TapSTARK - Commit Polynomial with Bitcoin Taptree

This article aims to elucidate the use of taptree as a Polynomial Commitment. The Taptree Polynomial Commitment is constructed primarily based on the concepts of Taptree, bitcommitment, and BitVM2.

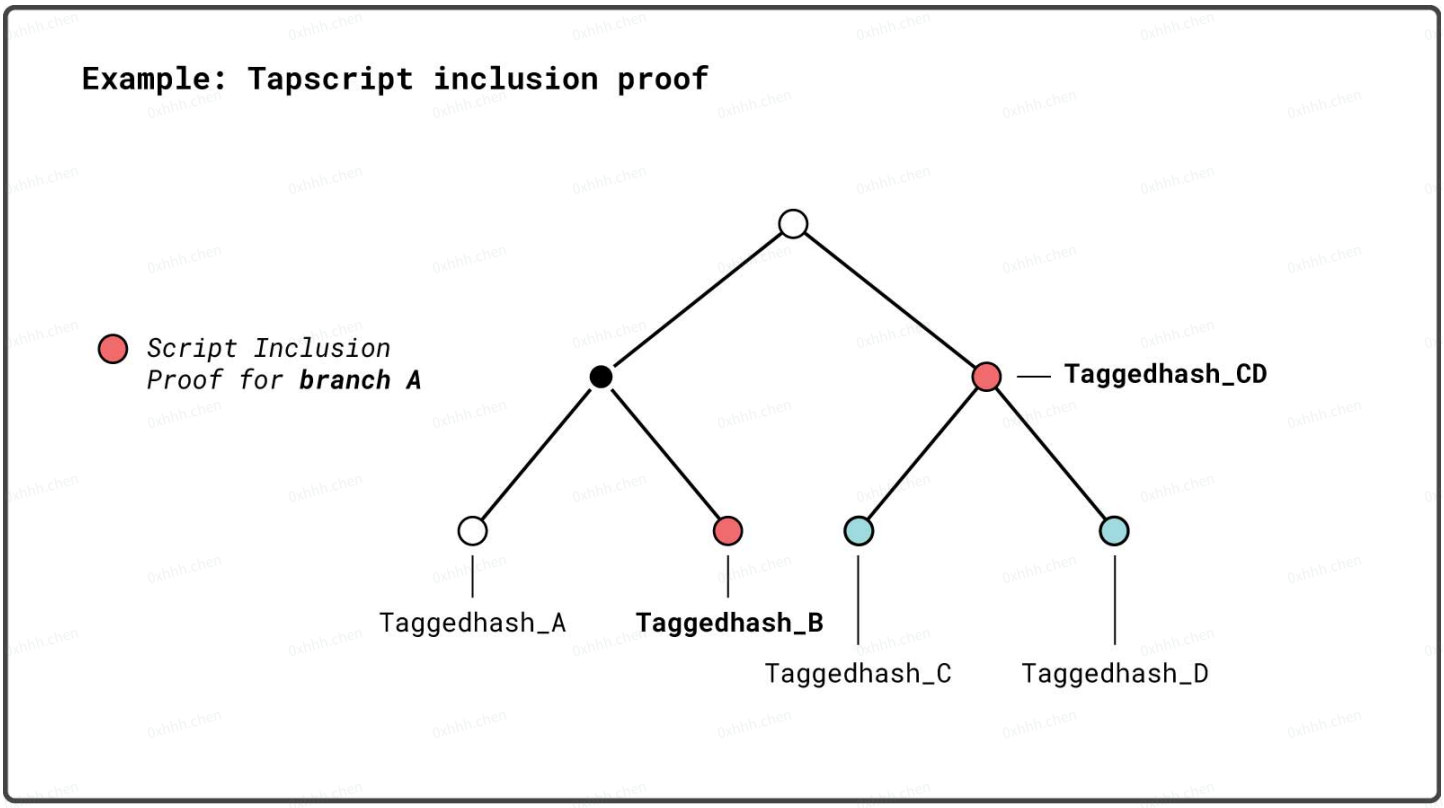
The structure of the article is as follows:

1. **Fundamental Concepts**: This section covers the essential background knowledge required to understand the subsequent content.
2. **Transition from Polynomial Merkle Tree Commitment to Polynomial Taptree Commitment**: Here, we detail the process of converting a Merkle Tree Commitment into a Taptree Commitment.
3. **Interactive Taptree Polynomial Commitment on the Bitcoin Blockchain**: This section presents a comprehensive example of an interactive Taptree Polynomial Commitment on the Bitcoin blockchain, followed by its transformation into a non-interactive version using the Fiat-Shamir heuristic.

1. Fundamental Concepts

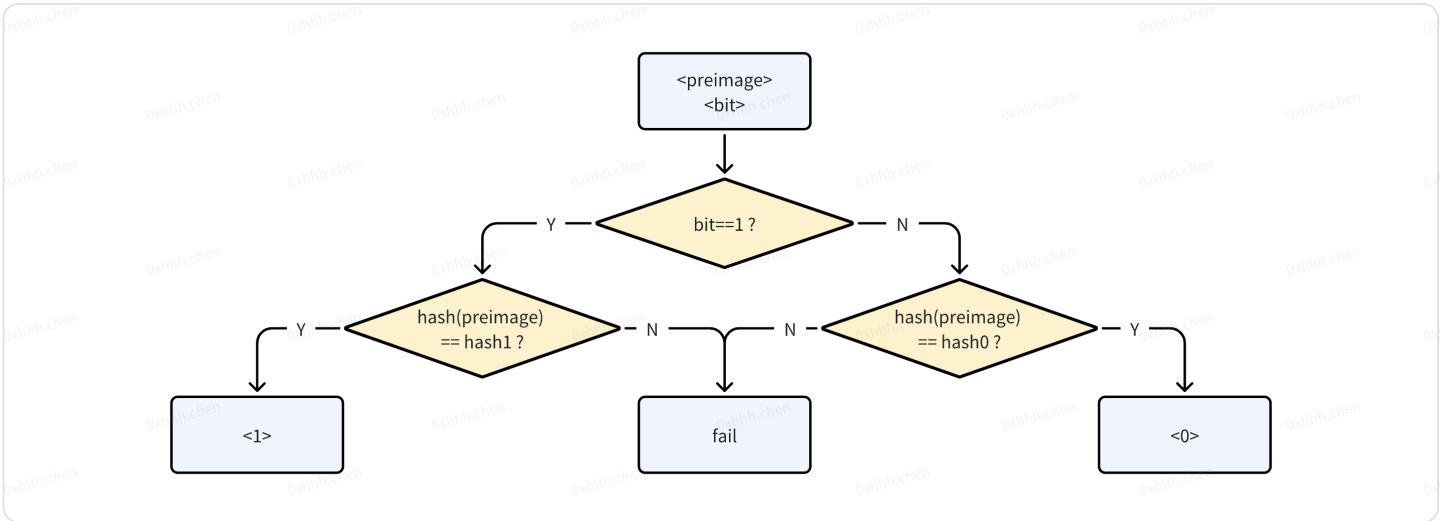
1.1 Taptree

Taptree is a MerkleTree that places executable bitcoin-script in leaf nodes. Anyone who can provide the input of any leaf script so that the program can be executed successfully and provide the corresponding valid merkle path can construct a transaction to unlock the taptree.



1.2 Bitcommitment

The concept of bit commitment is a variation derived from Lamport signatures. It involves two hashes, *hash0* and *hash1* (with the corresponding preimages being *preimage0* and *preimage1*). The prover then can determine the value of the bit: either "0" by disclosing *preimage0*, which corresponds to *hash0*, or "1" by disclosing *preimage1*, corresponding to *hash1*. Bit commitments enable the prover to set a variable's value across various scripts and UTXOs, extending the execution runtime of Bitcoin's VM through transaction splitting across multiple transactions.

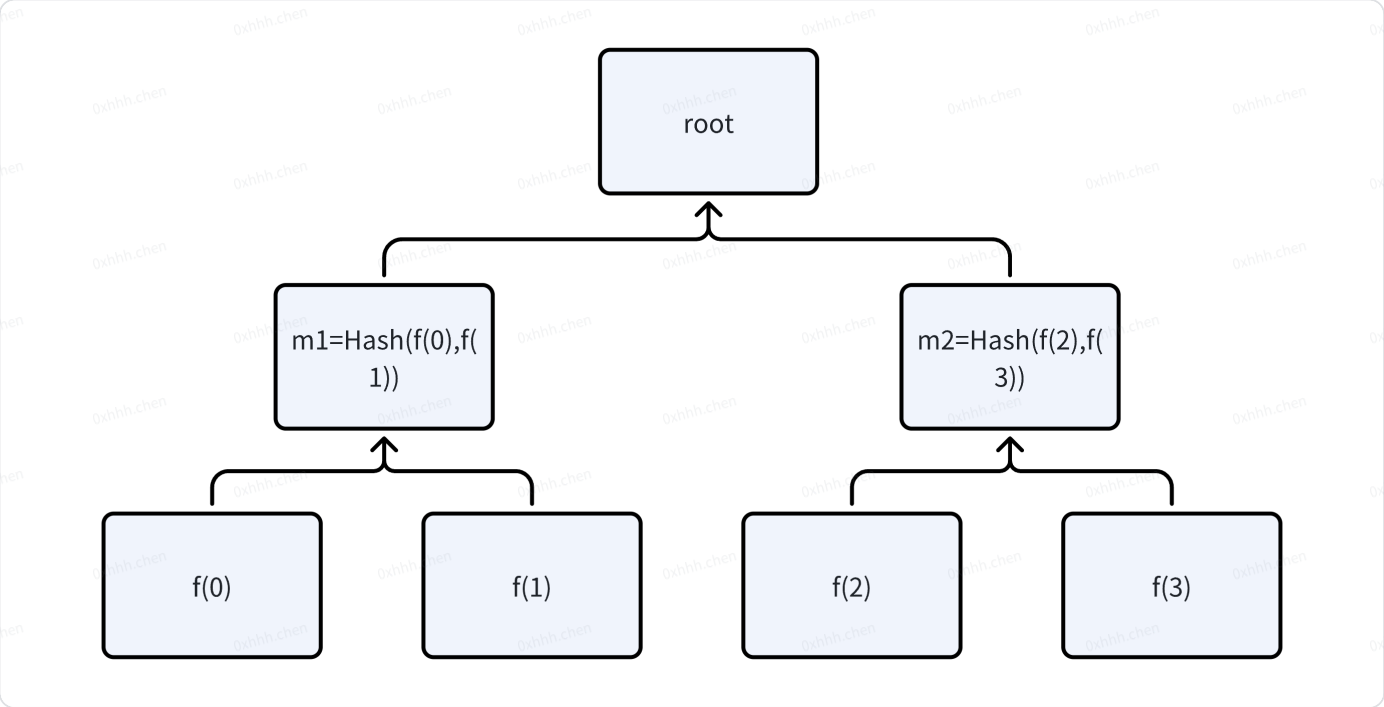


Process of Bit Commitment Script

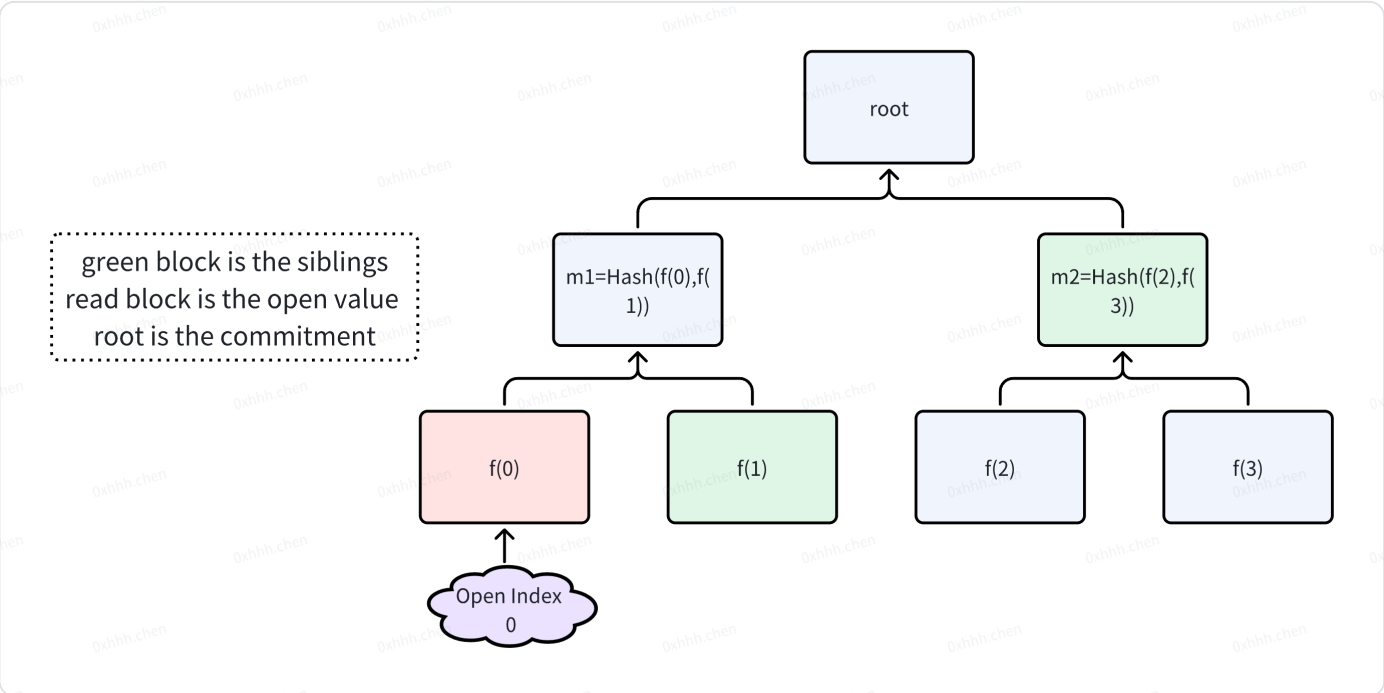
2. Merkletree as Polymial commitment

Commit to a univariate $f(X)$ in $F_p^{(<=d)}[X]$ using Merkletree.

1. Commits Polynomial as a Merkletree root and sends the root to Verifier.



2. Opens an entry of the merkletree:
   a. Verfier specifies a leaf node to open.
   b. The Prover sends the specified leaf node along with the sibling hashes of all nodes on the path from the root to the leaf.



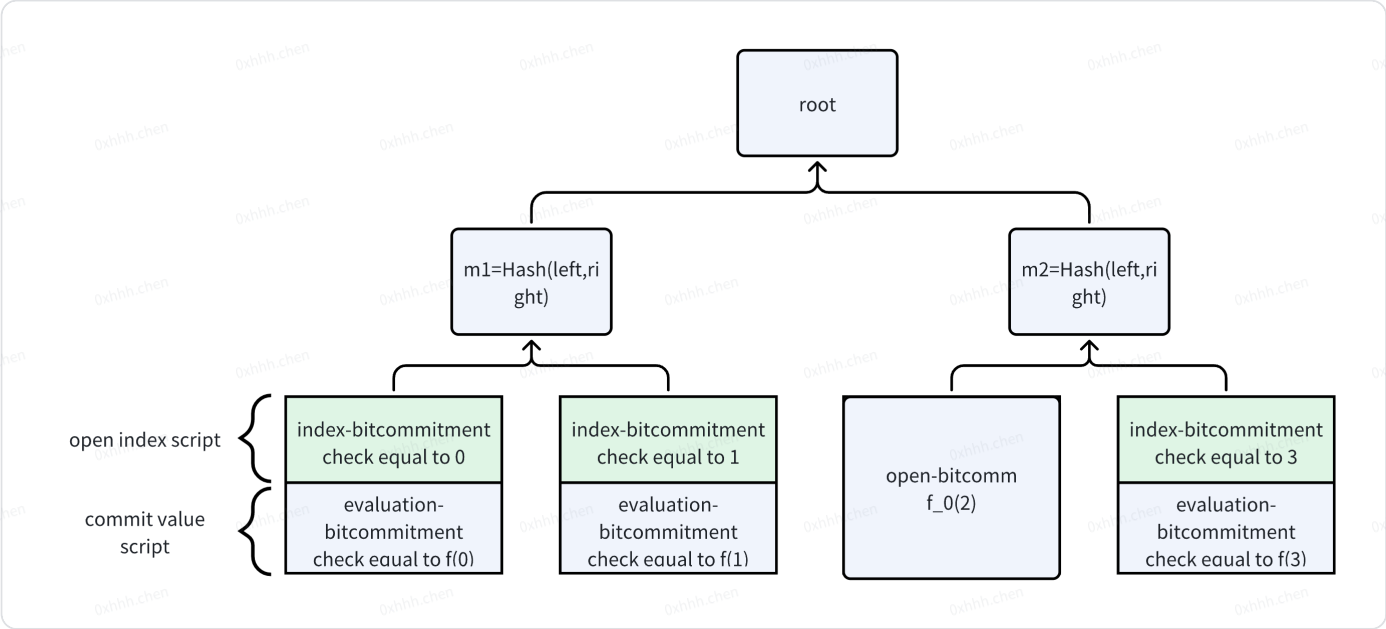
   c. Verifier checks the validity of the MerklePath.

3. Taptree as Polynomial Commitment

The leaf-script within tapleaf can not observe self index on chain, so we need commit the index into leaf-script.

Commit to a univariate $f(X)$ in $F_p^{(<=d)}[X]$ using Taptree.

1. Commit Polynomial as a Taptree root and sends the root to Verifier.



   a. Open Index Script: This part of the script is used to simulate opening a leaf of the Merkletree. The input for open-index-script is the preimage of index-bitcommitment. If the verifier wants to open the first leaf, he needs to provide a preimage of index-bitcommitment corresponding to 0.

   b. Commit Value Script: This part of the script is used to commit the polynomial evaluate (f(index)) at corresponding point. The input for commit-value-script is the preimage of evaluation bitcommitment. If the verifier wants to open the f(0), the prover needs to provide the preimage of evaluation-bitcommitment corresponding to f(0)-value.

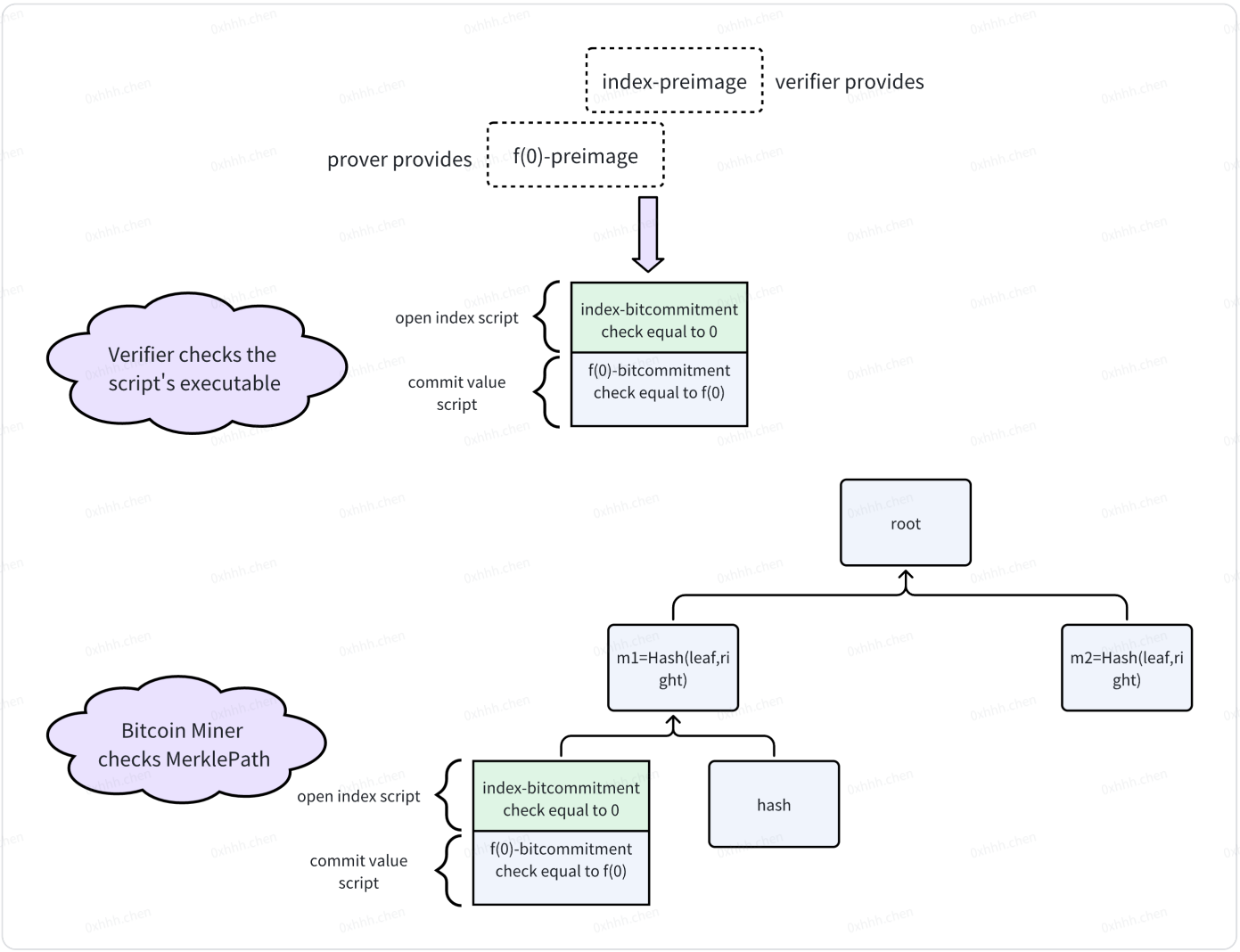
2. Open a leaf of the Taptree root.

   a. Verfier specifies a leaf node to open。

   b. Prover sends tapleaf and sibling hashes of all nodes on root-to-leaf path.

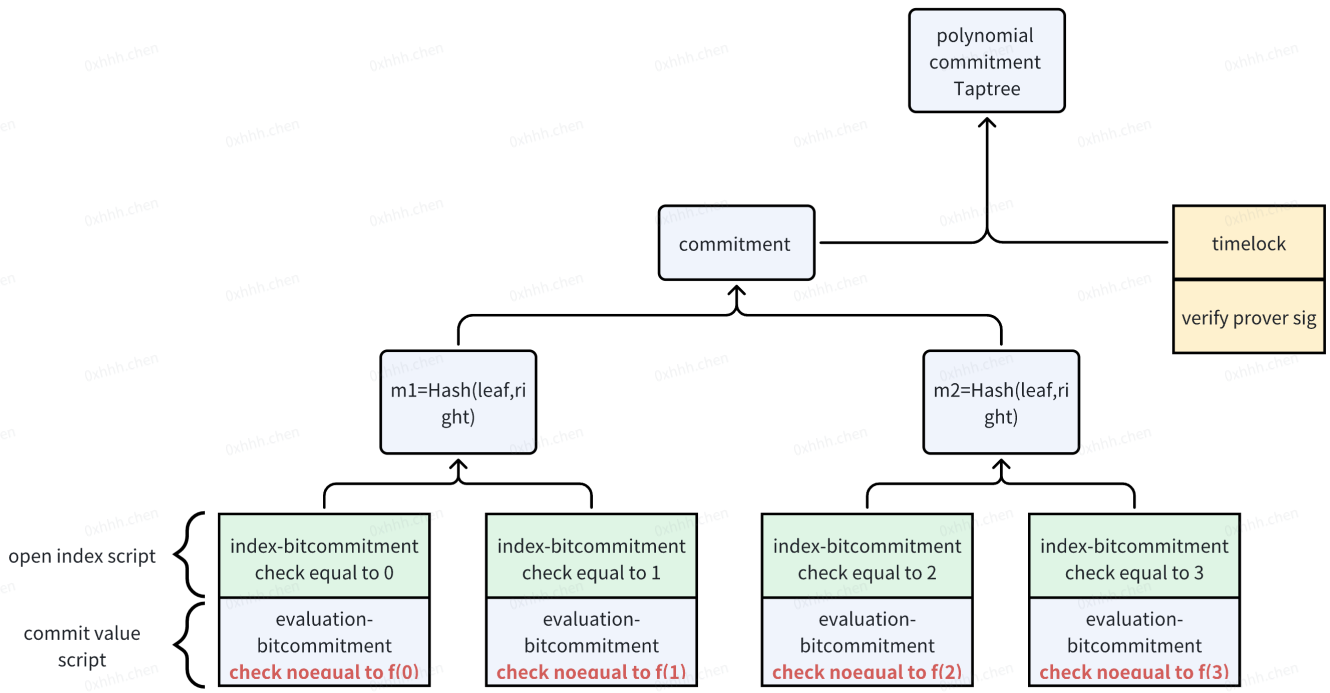
   c. Verifier executes the script of the given tapleaf with the input of preimage of bitcommiment to check the committed value.

   d. Bitcoin miner verifies the merklepath prover provided.



3.1 Using NoEqual Script

We update the commit value script of the polynomial commitment taptree as no_euqal script. It seems that at the above the taptree polynomial commitment, the prover needs to execute the open leaf script successfully to prove using the correct committed value. But if we use the no_euqal script, the prover does not need to execute the script, the prover just deposits some btc at the taptree. The prover can refund the deposit after waiting for a timelock if the prover is honest. The verifier can refund the prover's deposit if they observe the malicious prover error computation.

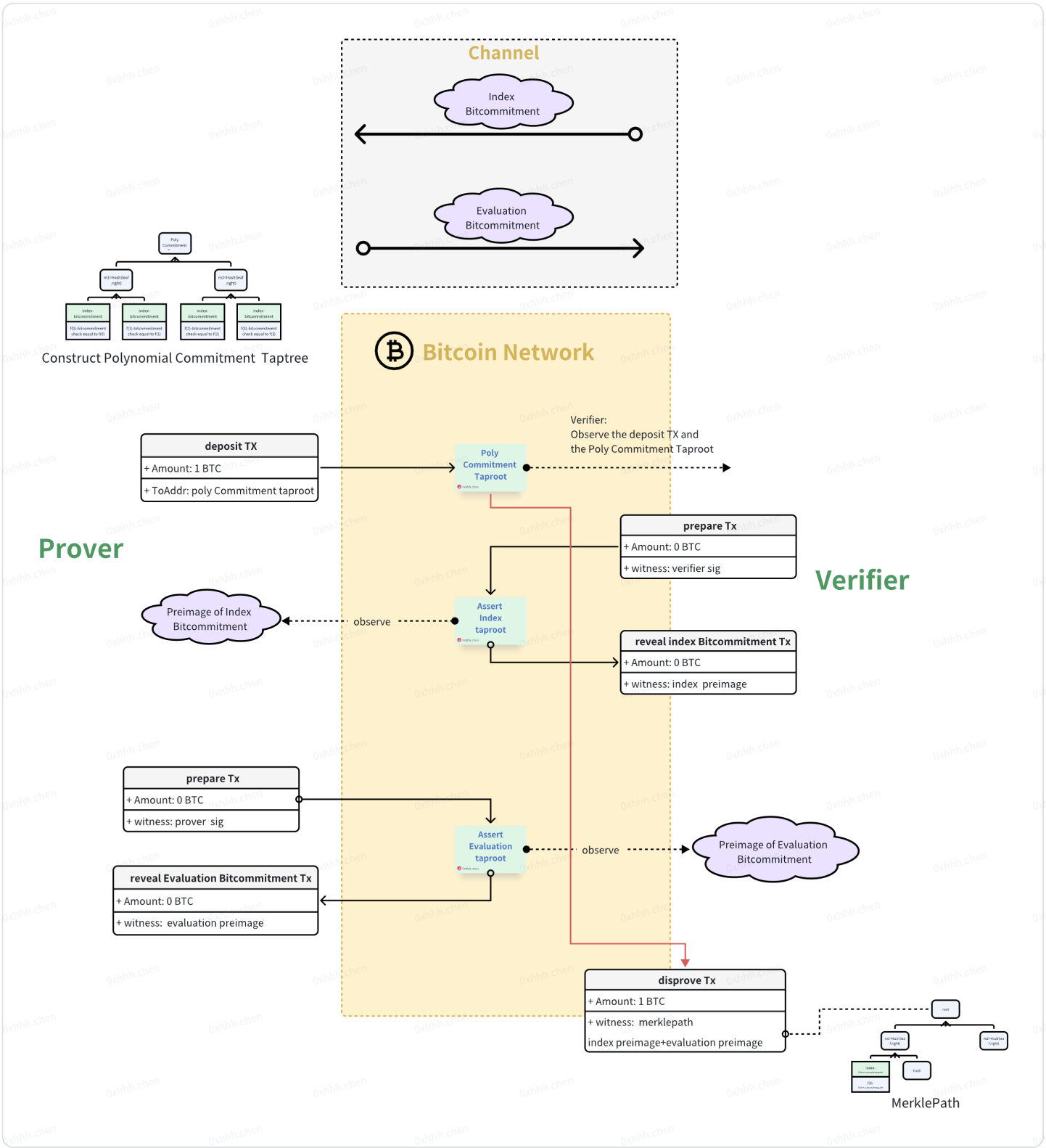


4. Formalization of On-chain Polynomial Taptree Commitment

The basic idea of the *On-chain Polynomial Taptree Commitment* is that if the prover is malicious, the verifier can slash the prover.

4.1 Interactive protocol

1. Verifer sends the index bitcommitment to Prover.
2. Prover sends the evaluation bitcommitment to Verifier.
3. Prover constructs the Polynomial Commitment Taptree and sends the UTXO to the corresponding taproot address on bitcoin network as the desposit.
4. Verifier claims the value of the index through revealing the preimage of the index commitment to open the polynomial at the corresponding index.
5. Prover claims the value of the evaluation at the open point through revealing the preimage of the evaluation bitcommitment to open the polynomial at the corresponding index.
6. If the evaluation of the prover reveals is not correct, the verifier can unlock the corresponding leaf-script and get the prover's deposit.



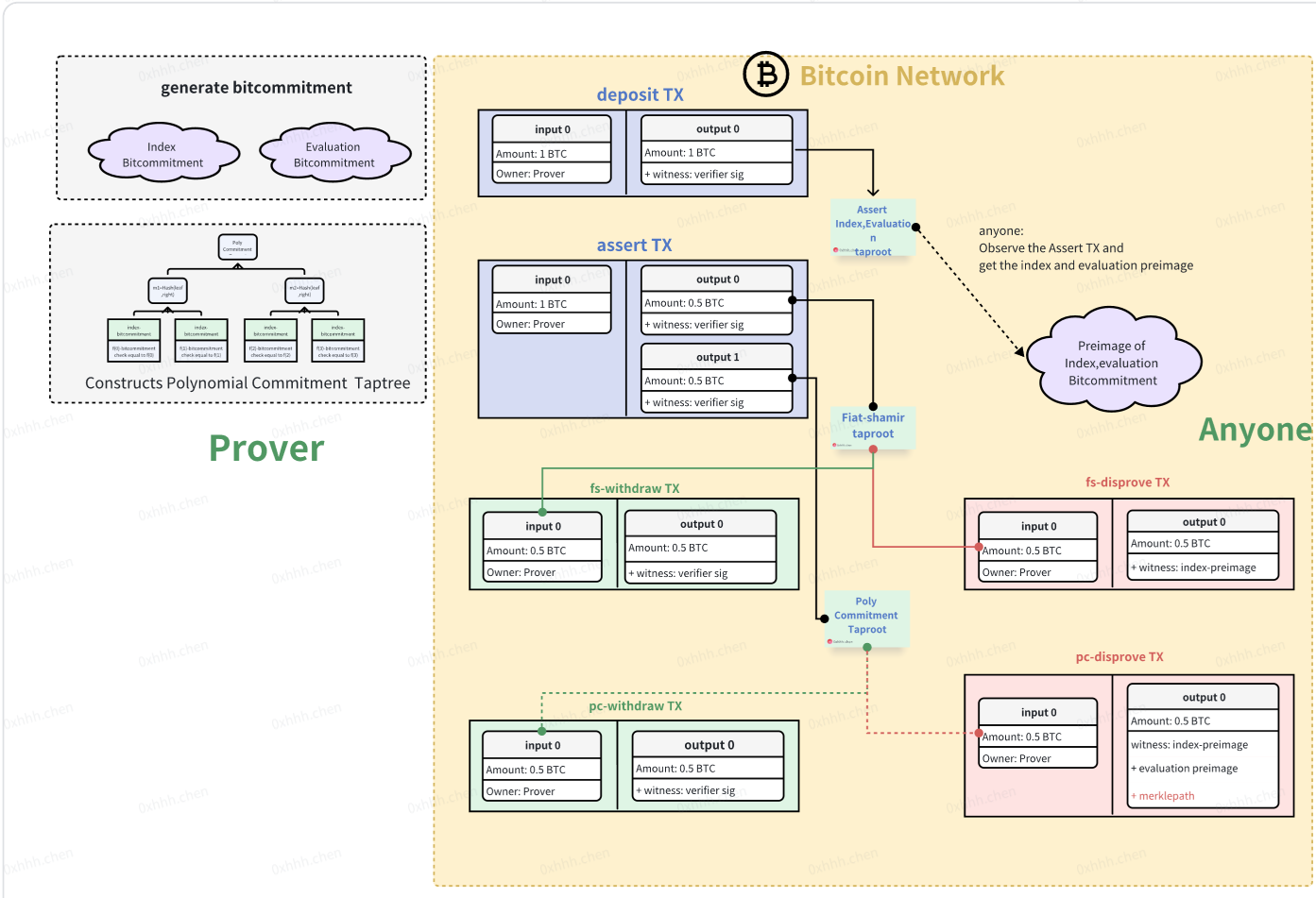
4.2 Non-interactive protocol

In an interactive protocol, the process relies on the Verifier selecting the corresponding challenge points on-chain, to which the Prover subsequently responds. However, practical applications often necessitate multiple rounds of challenges. Consequently, we aim to replace the Verifier’ s challenges with the Fiat-Shamir transformation to streamline the protocol.

The verifier-query is simulated by the Fiat-shamir Script.

1. The prover generates the index-bitcommitment and the Evaluation-Bitcommitment

2. Prover constructs the polynomial commitment Taptree offchain.
3. Prover sends the deposit transaction to deposit 1 btc to the Assert-Index-Evaluation taproot
4. Prover sends the assert transaction to claim the value of the index and the evaluation through revealing the preimage of the index Bitcommitment and the evaluation Bitcommitment to get the evaluation of the polynomial at the corresponding index.
5. If the assert index is not consistent with the Fiat-shamir output , anyone can create a fs-disprove transaction to find the issue and claim the prover's deposit.
    a. If the assert index is consistent with the Fiat-shamir output, the prover can create a fs-withdraw transaction to refund the deposit after waiting for a timelock.
6. If the assert value of the evaluation is incorrect, anyone can create a pc-disprove transaction to find the issue and claim the prover's deposit.
    a. If the assert value of the evaluation is correct, the prover can send a pc-withdraw transaction to refund the deposit after waiting for a timelock.



Fiat-Shamir Script

The details for Fiat-Shamir Script:

```
1 // input: preimage-of-index
```

```
2
3 <poly_commitment_taproot>
4 HASH
5 <preimage-of-index>
6 <index_bitcommitment>
7 Recover_Index_From_Bitcommitment
8 OP_NOEQUAL
9
```

We need to notice the Fiat-shamir script must input the polynomial commitment taproot to hash and sample an index to query. So the polynomial commitment taproot is **hardcode** within the Fiat-shamir script.

Federation

However, on the Bitcoin blockchain, only the Taproot structure is actually on-chain. Consequently, it is challenging for any participant to verify that the Prover’ s construction of the Fiat-Shamir Taptree and the Assert Index Evaluation Taptree is correct. Additionally, we need to ensure that the Prover has provided the correct MerklePath for the sampled points. Furthermore, without multisig, it is practically impossible to restrict a UTXO to be spent to a predefined address on Bitcoin.

To address these challenges, we propose introducing a federation of n members. This federation would require n-of-n signatures for transaction execution, ensuring that all transactions are authenticated and validated by all members. This means that all unlocking transactions shown in our diagram must include the signatures of the n federation members in the witness field for execution.

While this might seem impractical due to the need for communication with the entire federation to obtain signatures before each transaction, Bitcoin’ s pre-signature mechanism offers a solution. These transactions can be pre-signed at the time when the polynomial is determined off-chain, ensuring smooth and efficient execution of the entire process.

Gas Comparison

Polynomial Commiment	2^4	2^8	2^12	2^14	2^18
taptree	800	800	800	800	800
merkletree(op_cat)	103	191	283	329	424

https://github.com/Bitcoin-Wildlife-Sanctuary/bitcoin-circle-stark/blob/main/src/merkle_tree/bitcoin_script.rs#L91

- 16 query * 16 folding