Taproot Consensus yellow paper

Chapter 1: Abstract

The non-Turing complete nature of the Bitcoin network limits its ability to directly implement Layer2 scalability solutions similar to Ethereum Rollups. The Bitcoin network's script contract layer can only perform simple “Transfer” operations and cannot support more complex smart contract functionalities. Therefore, constructing a Layer2 scalability solution solely from the Bitcoin script layer is not feasible.

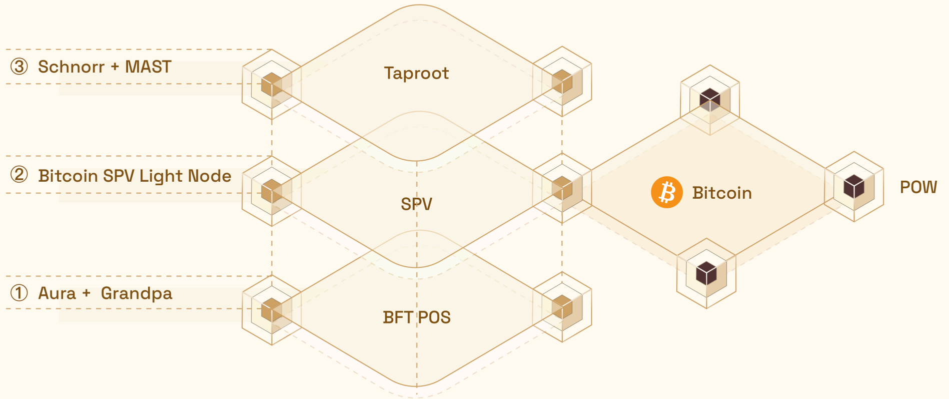
All of the existing Bitcoin Layer2 solutions, such as the Lightning Network, rely on external decentralized nodes to maintain the ledger of state channels. This compromise is a fundamental reason why the Lightning Network has not seen widespread adoption, as the lack of Byzantine Fault-Tolerant(BFT) trust mechanisms makes it difficult to attract a large number of users to the network.

To address this issue, the best approach is to integrate all existing capabilities of Bitcoin to construct a fully decentralized BTC Layer2 scalability solution.

BEVM's Taproot Consensus is an innovative design that achieves this goal. It combines Bitcoin's Taproot technology (Schnorr and MAST), Bitcoin SPV lightweight nodes, and BFT PoS consensus mechanism to create a decentralized, highly consistent Layer2 network.

By utilizing Bitcoin SPV lightweight nodes, BEVM can efficiently and decentralized synchronize the Bitcoin network state. Through a threshold signature network constructed using Taproot technology and BFT PoS consensus mechanism, BEVM can decentralized synchronize its state back to the Bitcoin network, thereby forming a fully decentralized bidirectional channel. Taproot Consensus achieves a fully decentralized Layer2 scalability based on highly consistent BFT PoS consensus, leveraging the inherent security of the Bitcoin network.

Chapter 2: Architecture



Taproot Consensus

①: Threshold Signature generated by Taproot Musig2

②: The Bitcoin SPV light client + Bitcoin Network based on POW = Decentralization. SPV is written on L2 consensus layer.

③: The BFT POS consensus node merges ② and ③ modules, allowing Taproot Consensus to be decentralized without any intermediary participation.

BEVM

2.1 Component

The Taproot Consensus of BEVM is composed of three main components: Schnorr + MAST, Bitcoin SPV, and Aura + Grandpa.

2.2 Schnorr + MAST and Bitcoin SPV

In the BEVM system, each validator holds a BTC private key used for Schnorr signatures. The properties of Schnorr signatures enable efficient signature aggregation, thereby enhancing the system's security and efficiency. A large MAST (Merkelized Abstract Syntax Tree) is formed using the aggregated public key $Pagg$Pagg generated by the Musig2 multi-signature scheme.

After generating the root hash of the MAST tree, validators conduct BTC transfers and inscription operations to the threshold signature address generated by the MAST tree, enabling the Bitcoin mainnet to submit data to the BEVM network. Additionally, each validator acts as a Bitcoin SPV (Simplified Payment Verification) lightweight node, allowing them to securely and permissionlessly synchronize the BTC network state.

2.3 Schnorr+MAST and Aura+Grandpa

Blockchain is a special type of distributed network that solves the problem of decentralized trust by introducing Byzantine Fault Tolerance (BFT) mechanisms. Essentially, blockchain can be viewed as a decentralized network with Byzantine fault-tolerant properties.

In the BEVM system, advanced PoS consensus mechanisms such as Aura and Grandpa are used to ensure network consistency and security. Aura and Grandpa are advanced PoS consensus protocols that implement Byzantine fault tolerance, ensuring high consistency among network nodes through distributed protocols. To prevent malicious behavior by validators in the PoS consensus, the system incorporates a BTC staking governance mechanism, using BTC to ensure the security of the BEVM network.

By introducing the Aura and Grandpa consensus mechanisms, the decentralized threshold signature network within the BEVM system gains Byzantine fault-tolerant properties, forming a unique Layer2 blockchain structure. This threshold signature blockchain not only ensures the security of the BEVM network but also enables the decentralized synchronization of BEVM states back to the BTC network.

2.4 tBTC and Taproot Consensus

The underlying technical structure of Mezo is based on the tBTC protocol. tBTC utilizes Bitcoin multi-signature to construct a threshold signature network, which offers stronger consistency compared to traditional decentralized networks.

However, to achieve a truly decentralized and Byzantine fault-tolerant blockchain solution, Mezo needs to further combine this multi-signature network with a BFT PoS (Byzantine Fault Tolerance Proof of Stake) consensus mechanism.

In contrast, the Taproot Consensus adopts a more advanced design. It combines Schnorr, MAST, Musig2 multi-signature schemes, Bitcoin SPV lightweight nodes, and the Aura and Grandpa Byzantine fault-tolerant consensus mechanisms to construct a highly consistent and secure decentralized Layer2 scalability solution. This integration not only enhances the scalability and usability of the Bitcoin network but also ensures the security and consistency of the BEVM network.

Chapter 3: Threshold Signatures

The Taproot upgrade is a significant improvement to the Bitcoin network, encompassing Schnorr signatures (BIP 340), Taproot (BIP 341), and Tapscript (BIP 342). The primary goals are to enhance Bitcoin's privacy, efficiency, and flexibility. The Schnorr signatures allows multiple signatures to be aggregated into one, thereby reducing transaction fees and memory overhead. Additionally, Merkle Abstract Syntax Trees (MAST) hides complex Bitcoin Scripts in the structure of Merkle trees, improving the anonymity and privacy of transactions.

MuSig2 is an advanced multi-signature scheme that enables multiple participants to jointly sign a single document or transaction, ensuring that the final signature is indistinguishable from one generated by a single entity to external observers. This scheme is specifically designed to facilitate secure communication among multiple parties in insecure network environments, providing an efficient and secure multi-signature solution. It ensures the security of operations while maintaining the privacy of the participants.

3.1 Schnorr Signatures

- **Public Key Generation:** Select a random number (d) as the private key, and let (G) be the base point. Then the public key (P) is:

$$P = d * G$$

- **Signature Generation:** Select a random number (r) as the nonce, and let ($message$) be the message to be signed. The signature (r, s) is generated as follows:

$$R = r * G$$

$$e = Hash(R, P, message)$$

$$s = r + e * d$$

- **Aggregated Public Key and Signature:** The public keys and signatures of Schnorr signatures can be aggregated.

  Given：

$$d_{agg} = \sum_{i=1}^{n} d_i$$

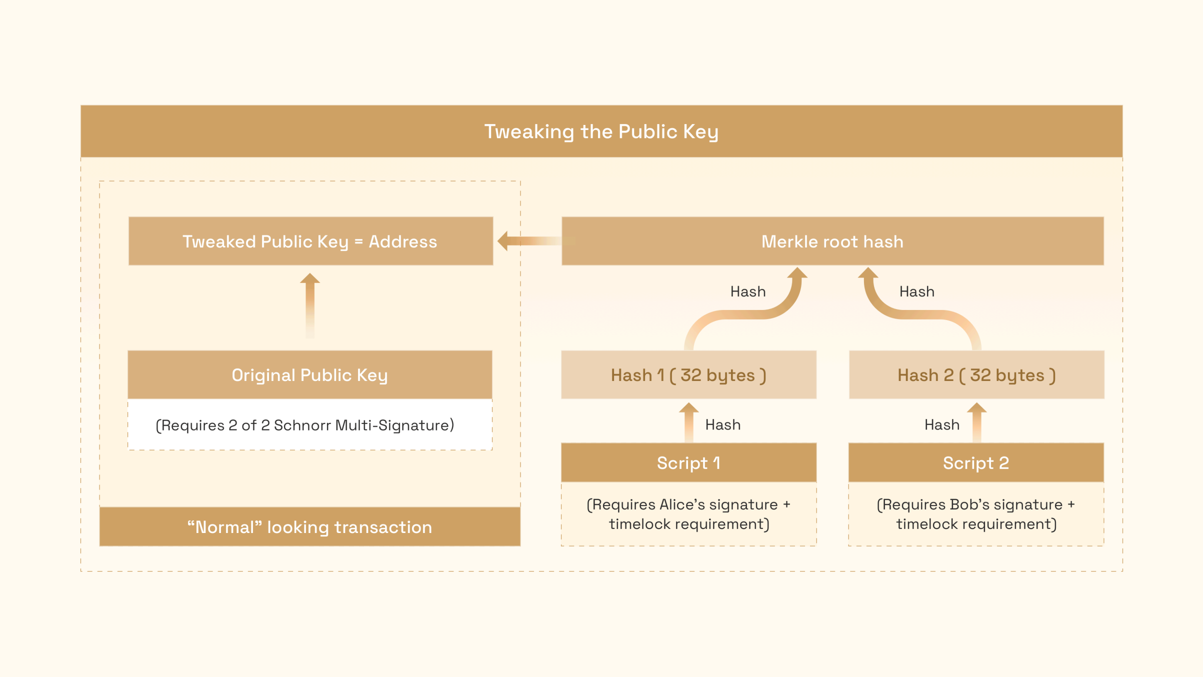
  Then：

$$P_{agg} = \sum_{i=1}^{n} P_i$$

$$s_{agg} = \sum_{i=1}^{n} s_i$$

Schnorr signature is a key technology in Bitcoin's Taproot upgrade, replacing the previous ECDSA signature algorithm. From the specific formulas of Schnorr signatures, it is easy to prove that the private key (d), public key (P), and signature elements (r, s) all possess linear additivity. This unique property allows multiple signatures to be efficiently aggregated into a single signature, significantly reducing the size of transaction data. Consequently, this enhances transaction processing speed and lowers fees.

3.2 Merkle Abstract Syntax Trees


Tweaking the Public Key
Tweaked Public Key = Address
Merkle root hash
Hash
Hash
Original Public Key
(Requires 2 of 2 Schnorr Multi-Signature)
Hash 1 (32 bytes)
Hash 2 (32 bytes)
Hash
Hash
Script 1
(Requires Alice's signature + timelock requirement)
Script 2
(Requires Bob's signature + timelock requirement)
"Normal" looking transaction

MAST (Merkelized Abstract Syntax Tree) is a data structure that combines the advantages of Merkle trees and abstract syntax trees. It is designed to optimize the privacy and efficiency of Bitcoin Network. MAST allows complex smart contract conditions to be hidden within the structure of a Merkle tree, revealing only the relevant script parts (Script 1 or Script 2) when these conditions are triggered.

MAST trees can be used to organize and verify complex scripts and conditions. The steps to construct a MAST tree are as follows:

- **Script Splitting:** Divide the complex smart contract script into multiple sub-scripts, with each sub-script representing a possible execution path.
- **Hash Calculation:** Compute the hash value for each sub-script. Let the set of sub-scripts be denoted as $S = \{S_1, S_2, \ldots, S_m\},$ The corresponding set of hash values is $H = \{H(S_1), H(S_2), \ldots, H(S_m)\}$。

- **Tree Node Construction:** Use the set of hash values as leaf nodes and iteratively compute the parent node hash values through a binary tree structure, ultimately generating the root hash value H_{root}。 This process follows the construction method of a Merkle tree. Specifically: H_{left} and H_{right} represent the hash values of the left and right child nodes, respectively, and ($||$) denotes the concatenation operation.

$$H_{\text{parent}} = H(H_{\text{left}} || H_{\text{right}})$$

- **MAST Root Hash:** The final root hash value ($H_{root}\}$) is generated as the root hash of the MAST tree, representing the overall state of the smart contract.

3.3 MuSig2

Musig2 is a multi-signature scheme and a variant of the MuSig signature scheme. Musig2 allows multiple signers to create an aggregated public key from their respective private keys and then jointly create a valid signature for that public key. The aggregated public key created in this manner is indistinguishable from any other public key.

Musig2 is a simple and highly practical two-round multi-signature scheme with the following advantages:

i. It is secure under concurrent signing sessions.

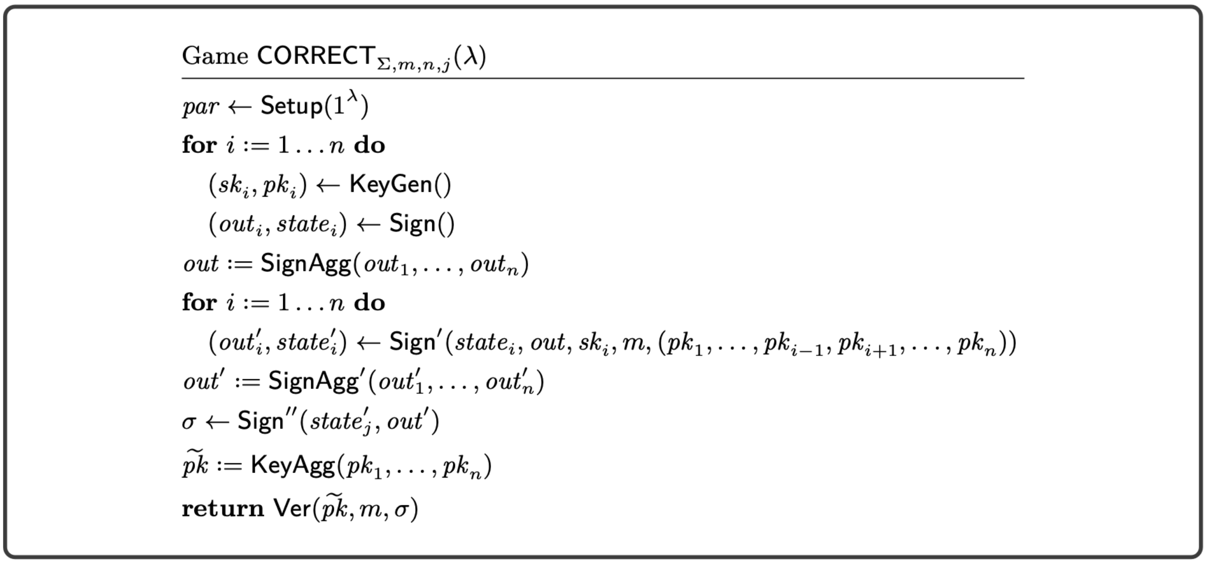
ii. It supports key aggregation.

iii. It outputs standard Schnorr signatures.

iv. It requires only two rounds of communication.

v. It has signer complexity similar to that of regular Schnorr signatures.

The Musig2 signing process involves two main stages: first, the generation of keys and temporary key pairs, and second, the aggregation of signatures through two rounds of communication.



```
Game $\mathsf{CORRECT}_{\Sigma,m,n,j}(\lambda)$
$par \leftarrow \mathsf{Setup}(1^\lambda)$
for $i := 1 \ldots n$ do
    $(sk_i, pk_i) \leftarrow \mathsf{KeyGen}()$
    $(out_i, state_i) \leftarrow \mathsf{Sign}()$
$out := \mathsf{SignAgg}(out_1, \ldots, out_n)$
for $i := 1 \ldots n$ do
    $(out'_i, state'_i) \leftarrow \mathsf{Sign}'(state_i, out, sk_i, m, (pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n))$
$out' := \mathsf{SignAgg}'(out'_1, \ldots, out'_n)$
$\sigma \leftarrow \mathsf{Sign}''(state'_j, out')$
$\widetilde{pk} := \mathsf{KeyAgg}(pk_1, \ldots, pk_n)$
return $\mathsf{Ver}(\widetilde{pk}, m, \sigma)$
```

1. **Key and Temporary Key Pair Generation:**
   - At the beginning, each participant uses the `KeyGen()` function to generate their own public-private key pair.
   - Each participant also creates a temporary public-private key pair (nonce) using the `Sign()` function. The temporary public key ((out_i)) is then sent to the other participants in the first round of communication.
2. **Signature Aggregation Through Two Rounds of Communication:**
   - **First Round:**
     - Each participant sends their temporary public key ((out_i)) to the other participants.
     - Once all participants have received the temporary public keys from others, they can use the `SignAgg()` and `Sign'()` functions to generate their own signature fragments ((out'_i)).
   - **Second Round:**
     - These signature fragments are exchanged among the participants.
     - After receiving all the signature fragments from the other participants, each participant uses the `SignAgg'()` and `Sign''()` functions to combine these fragments and generate the final unified signature (σ).

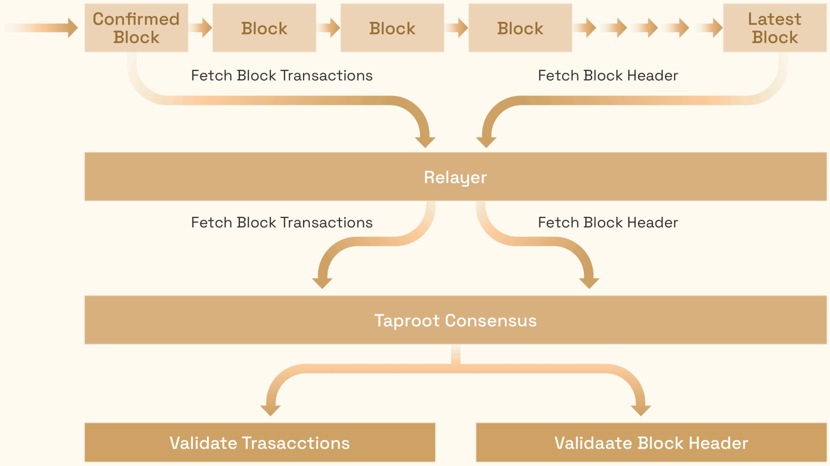
By following this process, Musig2 allows multiple participants to collaboratively create a single, aggregated Schnorr signature that is indistinguishable from a regular Schnorr signature, while ensuring security and efficiency in concurrent signing sessions.

Chapter 4: BTC SPV

In the Bitcoin network, BTC Simplified Payment Verification (SPV) lightweight nodes introduce an efficient mechanism that allows for the verification of BTC transactions without downloading the entire blockchain. This feature enables Taproot Consensus to synchronize BTC states in a fully decentralized and permissionless environment. By synchronizing only the relevant transactions, SPV lightweight nodes significantly reduce storage requirements without compromising security, providing a lightweight yet secure mechanism for BTC state synchronization.

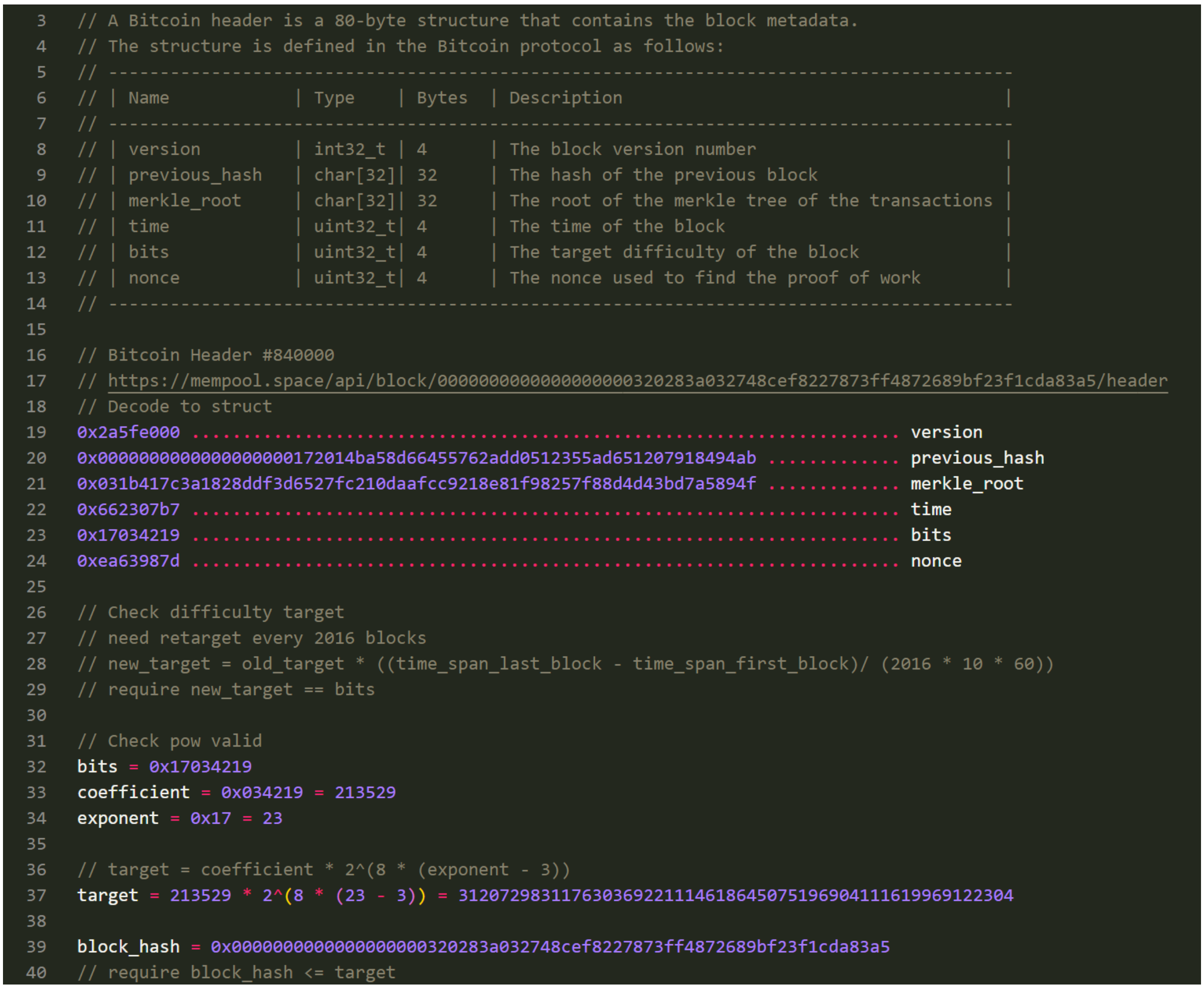
4.1 How It Works

Permissionless relayers automatically fetch the latest block headers from the BTC network and push the confirmed transactions from these blocks to the nodes. The nodes then verify the transactions and blocks according to SPV lightweight node verification rules, facilitating interaction with the BTC network. The block header contains sufficient information for transaction verification, such as the hash of the previous block, timestamp, proof of difficulty (nonce), and Merkle root. It allows nodes to store only the synchronized block headers to confirm the validity of transactions without needing to store the entire content of the blocks.



4.2 Block Header Verification

Block header verification involves checking the validity and integrity of the block header information.

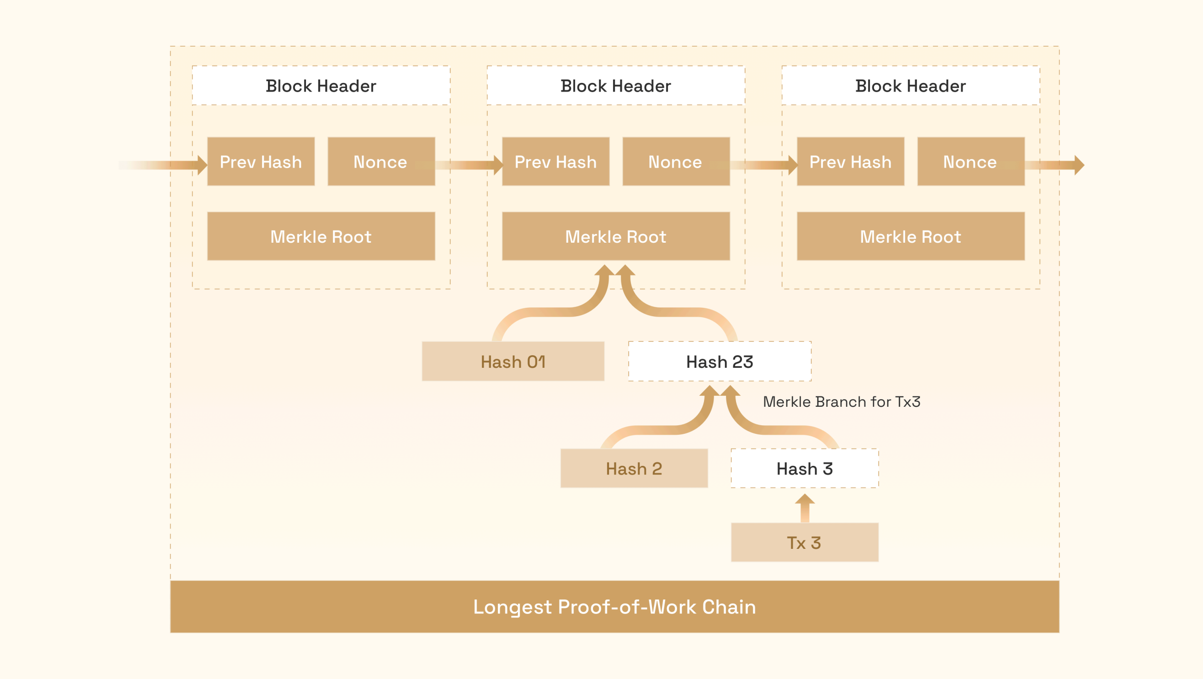


```
// A Bitcoin header is a 80-byte structure that contains the block metadata.
// The structure is defined in the Bitcoin protocol as follows:
// ----------------------------------------------------------------------------------------
// | Name            | Type    | Bytes  | Description                                     |
// ----------------------------------------------------------------------------------------
// | version         | int32_t | 4      | The block version number                        |
// | previous_hash   | char[32]| 32     | The hash of the previous block                  |
// | merkle_root     | char[32]| 32     | The root of the merkle tree of the transactions |
// | time            | uint32_t| 4      | The time of the block                           |
// | bits            | uint32_t| 4      | The target difficulty of the block              |
// | nonce           | uint32_t| 4      | The nonce used to find the proof of work        |
// ----------------------------------------------------------------------------------------

// Bitcoin Header #840000
// https://mempool.space/api/block/0000000000000000000320283a032748cef8227873ff4872689bf23f1cda83a5/header
// Decode to struct
0x2a5fe000 ..................................................................... version
0x0000000000000000000172014ba58d66455762add0512355ad651207918494ab ............. previous_hash
0x031b417c3a1828ddf3d6527fc210daafcc9218e81f98257f88d4d43bd7a5894f ............. merkle_root
0x662307b7 ..................................................................... time
0x17034219 ..................................................................... bits
0xea63987d ..................................................................... nonce

// Check difficulty target
// need retarget every 2016 blocks
// new_target = old_target * ((time_span_last_block - time_span_first_block)/ (2016 * 10 * 60))
// require new_target == bits

// Check pow valid
bits = 0x17034219
coefficient = 0x034219 = 213529
exponent = 0x17 = 23

// target = coefficient * 2^(8 * (exponent - 3))
target = 213529 * 2^(8 * (23 - 3)) = 312072983117630369221114618645075196904111619969122304

block_hash = 0x0000000000000000000320283a032748cef8227873ff4872689bf23f1cda83a5
// require block_hash <= target
```

1. **Download Block Header and Check Previous Block Hash:** Retrieve the block header information, including the version number, previous block hash, Merkle root, timestamp, difficulty target, and nonce.Confirm that the previous block hash included in the current block header is correct, ensuring that the current block is properly linked to the previous block.
2. **Verify Difficulty Target:** Check that the difficulty target of the current block meets the requirements of the Bitcoin network, ensuring that the computation difficulty is consistent with the network standards.
3. **Verify Proof of Work (PoW):** By checking the hash value of the block header, ensure that it is less than or equal to the current difficulty target, thereby validating the proof of work.

4.3 Transaction Verification

Transaction verification involves checking whether a transaction is validly included in the block. The process is as follows:



1. Construct the Merkle path of the target transaction.
2. The Merkle root is calculated using the Merkle path and transaction hash and compared with the Merkle root in the block header.
3. Confirm transaction format and signature validity to ensure transactions are not double-spended.

The verification process utilizes the Merkle tree structure of the Bitcoin blockchain to ensure the efficiency and accuracy of verification. This method is widely used in light nodes and other scenarios where the existence of transactions needs to be verified.

Chapter 5: BEVM's BFT PoS Consensus (Aura + Grandpa)

BEVM uses a combination of two consensus mechanisms, Aura (Authority Round) and Grandpa (GHOST-based Recursive Ancestor Deriving Prefix Agreement), to enhance the efficiency, security and finality of the blockchain network. By combining the Byzantine Fault Tolerance (BFT) mechanism, the reliability and attack resistance of the network are further improved.

5.1 Aura consensus

Aura (Authority Round) is a consensus algorithm based on Proof of Authority (PoA), mainly used for block generation. Here's how it works:

- **Taking turns to generate blocks**: Aura adopts a permission-based rotation scheduling mechanism. In each time slot (slot), a predetermined group of permission nodes (validators) take turns to generate blocks. Each validator generates a new block in the time slot of its turn. The block producer is determined according to the following formula:

  $$Validator_{current} = Validator_{(slot \mod N)}$$

  Among them, $Validator_{current}$ is the current block producer, N is the total number of validators, and slot is the current time slot number.

- **Fixed interval:** The block generation time is fixed (for example, one block every 6 seconds), which makes the block generation process more predictable and stable. Specifically:

  $$T_{block} = T_{slot}$$

  Among them, T_{block} is the block generation time, and T_{slot} is the fixed length of the time slot.

- **Fast block generation**: Because the block producers are predetermined, Aura can achieve fast and efficient block generation.

The main advantage of Aura is its simplicity and efficiency, which is suitable for blockchain networks that need to produce blocks quickly.

5.2 Grandpa consensus

Grandpa (GHOST-based Recursive ANcestor Deriving Prefix Agreement) is a consensus algorithm used for block finality. Here's how it works:

- **Multi-round voting**: Grandpa uses a multi-round voting mechanism to enable the verifiers in the network to agree on a certain prefix (prefix) of the blockchain. Each validator votes in each round for the blockchain it believes is optimal until more than two-thirds of the nodes in the network reach consensus. The statistical formula for voting results is:

$$\mathrm{VoteCount}(b) = \sum_{v \in V} \mathrm{vote}(v, b)$$

V is the set of all validators, b is the set of candidate blocks, and $\mathrm{vote}(v, b)$ represents the vote of verification node v for block b.

- **Block Confirmation:** Once more than two-thirds of validators reach consensus on a block, the block and all its ancestor blocks are considered final and irreversible. The conditions for reaching consensus are:

$$\mathrm{VoteCount}(b) > \frac{2}{3}N$$

  Among them, N is the total number of validators.

- **Efficiency and security**: Grandpa combines a Byzantine fault-tolerant algorithm to ensure that the system can still run stably even in the presence of malicious nodes.

Grandpa's main advantage is its efficient finality, allowing for rapid consensus even in the event of a network fork.

Chapter 6: Integration of BTC SPV, Threshold Signatures and BFT PoS Consensus

The BEVM network is an effective integration of BTC SPV light nodes, threshold signatures and BFT POS consensus, including:

6.1 Generate threshold signature address

Assume that the set of verifiers is $V = \{V_1, V_2, \ldots, V_n\}$, and each verifier V_i has a public key P_i. Any m nodes use the Musig2 multi-signature scheme to generate the aggregate public key P_{agg}, which is used as the script of the MAST tree to build a large MAST tree and thereby generate a threshold signature address. The set of all validators creates a robust and efficient threshold signature network through this MAST structure, allowing m of any n validators to complete signing and executing scripts through Musig2's two-round signature mechanism.

6.2 Synchronize BTC mainnet

The portability and efficiency of BTC SPV make it possible for each validator to act as an independent BTC SPV node in BEVM. The collection of all validators builds a BTC SRV distributed network. Due to the characteristics of BTC SPV, the BEVM network naturally has the ability to synchronize the status of the BTC network safely and without permission. By sending transfers or engraving transactions to the threshold signature address, the BEVM network can synchronize BTC mainnet data.

6.3 Submit BEVM data

BEVM combines threshold signature technology with Aura + Grandpa, an advanced Byzantine fault-tolerant consensus mechanism, to build a highly consistent and secure decentralized distributed network. By introducing this Byzantine fault-tolerant consensus, BEVM's threshold signature distributed network can achieve true decentralization. When a withdrawal transaction is initiated on BEVM, m of the verifiers in the set use the Musig2 multi-signature scheme to communicate, complete the signature, and complete the submission of BEVM data to the BTC network.

6.4 BTC Governance

The integration of threshold signature technology and Byzantine Fault Tolerance (BFT) Proof of Stake (PoS) consensus mechanism is achieved through a multi-layered key management and governance structure that ensures the decentralization and security of the system.

6.4.1 Multi-level key management

In this architecture, the election and governance of validators are carried out through the PoS governance system, involving the management of multiple key keys, including governance keys, block generation keys and threshold signing keys. These keys are set up and managed through permissionless user voting.

1. **Governance Key**: used to participate in network governance and the election of validators. Governance keys allow holders to vote on network parameters and rules, thereby affecting the overall governance structure of the network.
2. **Block Key**: Specially used to generate new blocks and verify transactions. The block generation key ensures the authority and security of the validators in the block generation process.

3. **Threshold Signature Key**: used to implement a distributed threshold signature scheme. The threshold signature key is generated and managed through the cooperation of multiple validators to ensure that the signature operation can only be performed when a certain number of nodes agree. This mechanism enhances the security and fault tolerance of the system and prevents single points of failure and malicious behavior.

6.4.2 Governance Roles

- **Ordinary users (Stakers)**: Users holding BTC can pledge Bitcoin to the network and participate in the election and governance decisions of validators. By staking BTC, users receive governance tokens, which give them voting and proposal rights.
- **Validators**: Validators are obtained through election, hold governance keys and block keys, and participate in threshold signature operations. The election and management of validator nodes are carried out through the PoS governance system, ensuring its decentralization and security.
- **Council**: elected by governance token holders, responsible for proposing and reviewing proposals to ensure the efficiency and rationality of the governance process. Board members hold governance keys and have higher voting weight on key decisions.
- **Technical Committee**: elected by the Board of Directors, responsible for technical decision-making and upgrades in emergencies. Members of the technical committee have the power to make quick decisions in response to emergencies.

6.4.3 Governance Process

- **Proposal Phase**: Any user holding governance tokens can propose governance proposals. These proposals can involve the adjustment of network parameters, the management of validators, the setting of key keys, etc.
- **Review Phase**: After the proposal is submitted, it is reviewed by the Board of Directors. Board members can modify, accept or reject proposals. Proposals that pass review enter the voting stage.
- **Voting Phase**: All governance token holders can vote on proposals. Voting adopts a weighted voting mechanism, and users holding more governance tokens have greater voting weight.

- **Execution Phase**: Proposals that pass the vote enter the execution phase. Execution involves corresponding technical operations, such as key generation and distribution, adjustment of network parameters, etc.

6.4.4 Combination with BTC staking

The security of the BFT PoS consensus depends on the participants' pledged funds. In order to enhance the security and reliability of the network, BEVM has introduced a Bitcoin staking governance mechanism that requires validators to pledge a certain amount of BTC to the network.

- **Staking and Rewards**: Users holding BTC can pledge Bitcoin to the network and participate in the election and governance decisions of validators. By staking BTC, users can earn block rewards and transaction fees in return.
- **Governance Rights**: Users who stake BTC also enjoy the governance rights of the governance token, including making proposals and participating in voting. Stakeholders who support passing proposals receive additional rewards, while stakers who submit malicious proposals will lose their staked Bitcoins as a penalty.
- **Security Guarantee**: Once malicious behavior occurs, these pledged funds will be slashed, effectively curbing the motivation for evil. Therefore, the greater the amount of BTC staked, the more secure and reliable the BEVM network will be.

In general, BEVM has created a decentralized, secure and efficient Bitcoin Layer2 solution by integrating multiple advanced Bitcoin technologies. This not only protects the security of BTC assets, but also injects new vitality into the future development of the Bitcoin ecosystem.

References

[1] Nakamoto, S. (2008, October 31). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf

[2] Wuille, P., Nick, J., & Ruffing, T. (2020). Schnorr Signatures for secp256k1. Retrieved from https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki

[3] Wuille, P., Nick, J., & Ruffing, T. (2020). Taproot: SegWit version 1 spending rules. Retrieved from https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki

[4] Wuille, P., Nick, J., & Ruffing, T. (2020). Validation of Taproot scripts. Retrieved from https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki

[5] Poelstra, A., Ruffing, T., & Seurin, Y. (2020). MuSig2: Simple Two-Round Schnorr Multisignatures. Retrieved from https://eprint.iacr.org/2020/1261

[6] Parity Technologies. (n.d.). Aura: Authority Round Consensus Algorithm. Retrieved from https://openethereum.github.io/Aura

[7] Parity Technologies. (n.d.). GRANDPA: A Byzantine Finality Gadget. Retrieved from https://github.com/w3f/consensus/blob/master/pdf/grandpa.pdf