Demystifying Blockchain Scalability: Sibling Chains with Minimal Interleaving

Jiangfeng Ma[1,2], Xuetao Zhang[1,2], and Xiangxue Li[1,2(✉)]

[1] Shanghai Key Laboratory of Trustworthy Computing, School of Software Engineering, East China Normal University, Shanghai 200062, China
[2] Shanghai Key Laboratory of Trusted Data Circulation and Governance and Web3, Shanghai 200240, China
xxli@cs.ecnu.edu.cn

Abstract. Blockchain provides alluring infrastructure for distributed ledgers supporting anonymous online payments. However, existing solutions for blockchain scalability have limitations of either being increasingly cumbersome in security analysis or inherent deficiencies (e.g., surviving on duplicate transactions). Moreover, current state-of-the-art scalable blockchains suffer from low throughput when used for larger transaction blockchains. To improve scalability, we propose SIBCHA, a novel protocol that equipped with k (power of 2) parallel sibling chains that correspond to k transaction pools (indexed by the rightmost $\log_2 k$ bits of transaction payers' addresses). In the protocol, i-th transaction (along with a Merkle tree path) would be announced to the i-th chain based on the rightmost $\log_2 k$ bits of the hashing determined in solving proof-of-work (PoW) puzzle (i is the exact value in decimal format represented by the $\log_2 k$ bits). To achieve parallel transactions, we design a inter-chain mechanism without other correlations (such as block ordering, inter-chain transactions, block updates, eventual atomicity decoupling, two-phase PoW puzzle solving, etc.), which makes SIBCHA considerably simpler than current state-of-the-art solutions (e.g., OHIE at IEEE S&P 2020 and Monoxide at USENIX Security 2019). SIBCHA has much less (e.g., $1.86\times \sim 3.16\times$) confirmation latency than OHIE. Prototype implementations also demonstrate that its throughput scales linearly with available bandwidth ($1.5\times$ that of Conflux).

1 Introduction

Bitcoin's central Nakamoto consensus features Proof of Work (PoW) mechanism characterized by simplicity and security [13,24] and combines a cryptographically-protected data structure with a reward scheme that encourages node participation. The blocks (containing multiple transactions) in Nakamoto consensus are linked sequentially as an append-only list (i.e., blockchain).

1.1 Scalability from Multiple Chains

Following the success of Bitcoin, the community notices its low transaction throughput that might hinder its practical applications and further development.

H. Duan et al. (Eds.): SecureComm 2023, LNICST 567, pp. 265–286, 2025.
https://doi.org/10.1007/978-3-031-64948-6_14

Bitcoin deals with about 7 transactions per-second and Ethereum 19 transactions, and both are less than 0.2% of the average bandwidth of their respective P2P networks [13]. Some designs (e.g., multiple chains) are proposed to improve throughput [3,10,17,18,21,30,31,33] with an attempt to maintain similar simplicity to Nakamoto consensus. Whereas, undesirable outcomes include that they either become more complex than Nakamoto consensus or find it increasingly difficult to prove their security (or even flawed). We take several seminal works [30,31] to see what exactly happen in these attempts on blockchain scalability.

Security Weakness in Heuristic Analysis. Monoxide [30] introduces interesting asynchronous consensus zones. Entire network state is divided into different zones. A zone is a chain, blocks and transactions are zone-specific (replicated and stored insides the zones), and consensus occurs independently within each zone. Unfortunately, this would make 51% attack much easier. Monoxide recommends that Chu-ko-nu mining enables a miner to create a single block in a designated zone by solving a single proof-of-work puzzle. Namely, miners maintain information about multiple zones and mine multiple blocks from these zones at the same time. It is shown, however, that Monoxide needs now 99% of the network's computing power to make each zone resistant to 50% attack, which eats asynchronous consensus zones away. On the other hand, plutonium mining is more like a solution for mining bigger blocks, and miners are inclined to perform selfish mining.

Ungainly Interleaving. OHIE [31] is a simple blockchain scaling scheme that provides formal security proof. To implement its confirmation rules, OHIE counts on a newly introduced *trailing* field in the block header and a *Block Attachment* structure. All miners should maintain the trailing fields to keep strict consistency of block ordering among multiple chains. Each miner updates its *trailing* value and resets its mining operation as long as one block is generated. For concurrent evolving of all chains, OHIE introduces two-phase confirmation and a block is claimed stable only if it is first partially confirmed (not being one of the last T blocks in each chain) and then fully confirmed (its *ranx* should be no greater than the minimal *ranx* value of all the last T blocks for each chain [31]). However, this would slow block confirmation of the whole network. Another issue is that transaction disorder could occur, i.e., OHIE blocks arriving later might be fully confirmed earlier than those arriving earlier. E.g., some payer's coin being spent in some confirmed transactions is not brought to its address in some to-be-confirmed transactions yet. All these identify inherent *architectural* limitations in the design.

1.2 Question

There exist several research lines for blockchain scalability (see related work in Sect. 2) and we focus in particular on the line of multiple chains. It is surely not a new idea to introduce multiple chains to blockchain platform, we aim however, especially at the following interesting problem:

Can we build a multi-chain consensus to optimize the interleaving of the chains and to be supported by formal security analysis in the meantime?

In multiple chain approaches, great attentions have to be paid to (payer-then-payee) inter-chain transactions (see Fig. 4 in Sect. 4.3). Plain handling of inter-chain transactions would make the resulting systems much involved. By optimizing the interleaving of the chains, we can reduce the system complicacy and avoid the introduction of cumbersome requirements such as block ordering, block update [31], eventual atomicity decoupling, two-phase PoW puzzle solving [30], etc. This would further reduce the probability of block disordering (arriving-later blocks being confirmed earlier than arriving-earlier blocks).

By formal security analysis (rather than heuristic analysis), we move away from design-break-patch cycle (e.g., Chu-ko-nu mining recommended in Monoxide [30]) that heuristic design and analysis generally have to confront and rule out "generic attacks" on the system. This can be done by providing a security reduction [4], showing that breaking a proposed protocol is as difficult as breaking a known secure protocol (e.g., Nakamoto consensus [24]) or as solving a mathematical hard problem (e.g., factoring, DLP [4]). However, it is rather tricky to program a correct security reduction using an adversary's adaptive attack.

1.3 Our Answers

We present SibCha, a simple blockchain consensus protocol with formal proofs. It takes a novel mining strategy of *mining(sub-block)-then-choosing(sub-block)*. More precisely, SibCha first collects a sub-block (see Fig. 3 in Sect. 4.1) from each chain, solves PoW puzzle (of all sub-blocks in a joint manner), and then picks up the sub-block indexed by the hashing of PoW puzzle to perform re-assembling and appending operations (see Sect. 4.2 for details). At the collecting stage, the miner already knows which chain each sub-block should be appended to, yet does not know which sub-block would be the lucky one until it obtains the PoW hashing.

In contrast, mining process of prior multiple chain approaches (e.g., OHIE) is characterized by *mining(block)-then-deciding(chain)*: each miner packs some transactions, solves PoW puzzle to obtain a block, and then decides certain chain to which the block is appended. This may lead to some *architectural* limitations, e.g., low block confirmation of the whole network, transaction disordering, etc.

SibCha has k (power of 2) sibling chains (indexed by the rightmost $\log_2 k$ bits of transaction payers' addresses). It allows any miner to mine in a parallel way the blocks announced to sibling chains. By the rightmost $\log_2 k$ bits of the hashing determined in solving PoW puzzle, corresponding transactions (along with a Merkle tree path and others) will be announced to i-th chain (i is the exact value in decimal format represented by the $\log_2 k$ bits).

1.4 Feature Summary of SibCha

SibCha is a PoW-based solution, similar to OHIE and Monoxide, and Table 1 clarifies their design characteristics. Besides concurrent evolving from expected uniform random distribution of the hashing, we summarize the major features of

SibCha below. Table 2 shows concrete comparisons between SibCha and some sharding-based protocols (see Sect. 2) from the perspectives of transaction mode, consensus, and resiliency.

Parallel Mining. The trick of headers Merkle tree path (HMTP) in SibCha block structure enables any miner to perform mining tasks in a parallel way without splitting up its computation performance. Our PoW mechanism compels any adversary to uniformly share out its computing power on all sibling chains.

Minimal Interleaving. In SibCha, minimal interleaving among sibling chains requires that a transaction (with some addresses being a payer) could only be deemed as legitimate if the latest transaction with the address being a payee has already been confirmed (with transaction-ordering guarantee) and no other correlations (such as block ordering, block update, eventual atomicity decoupling, two-phase PoW puzzle solving) are assumed.

Formal Evaluating. We provide theoretical security guarantees for SibCha, in particular chain-growth, chain-quality and consistency properties. The security model and proofs follow the line of provable security in cryptography community. We prove joint state of each chain for any honest node during entire execution of SibCha and that in Nakamoto consensus are strongly statistically close.

Experimental Analysis. We implement SibCha prototype to demonstrate its validity. Our experiments are configured similar to prior work. It is shown that SibCha scales linearly with available bandwidth, as in state-of-the-art protocols [10,17,19,31,33]. For example, the throughput of 1000–2500 transactions per second (500 bytes in size) can be achieved at the bandwidth of 8-20 Mbps. Such throughput is almost identical to that of OHIE and is 1.50× that of Conflux [18] at similar bandwidth. Experiments with the same settings as OHIE (same bandwidth, block interval and same number of chains) show that SibCha has much less confirmation latency than OHIE for same T (e.g., $T = 6$ for Bitcoin and $T = 10$–15 for Ethereum). The confirmation latency of OHIE is 3.16× that of SibCha for $T = 6$, and 1.86× that of SibCha for $T = 30$.

2 Related Work

Nakamoto Consensus. In 2016, Croman et al. [7] suggest that maximum block size should not exceed 4 MB considering current Bitcoin block-interval of 10 min, which produces a maximum throughput of 27 $txns/s$. There are several Blockchain protocols that apply this approach, such as Bitcoin-NG [10], Bitcoin Cash[1], and Bitcoin Classic[2]. Large block size solutions do not increase throughput significantly as large blocks result in higher block transfer latency. Chainweb [21] is an alternative to scale Nakamoto consensus, improving the throughput by maintaining k chains. Unlike SibCha, there are no measures in

[1] https://bitcoincash.org/.
[2] https://bitcoinclassic.com/.

Chainweb to deal with the adversary's full attack on any one of k chains. Chainweb [6,22] only considers few attack strategies. Fitzi et al. [11] show that some attacks on specific chains may increase confirmation latency time in Chainweb by k^2. SibCha has no this problem. Similar methods are proposed in [11] and [3] where of all k chains, one is designated as a special chain in the sense that the blocks on other chains are related to the blocks on the special chain.

Table 1. Comparisons among OHIE, Monoxide, and SibCha.

Characteristics	OHIE	Monoxide	SibCha
Consensus	PoW	PoW	PoW
Transaction mode	UTXO	Account	UTXO
Block structure	complex	simple	simple
Block confirmation rule	complex	simple	simple
Total resiliency	50%	50%[a]	50%
Block confirmation delay	high	low	low
Correlation among chains	high	low	low
Power consumption	low	high	low
Two-phase PoW solving	no	yes	no
Sequential order of blocks among chains	strict	loose	loose

[a] Chu-ko-nu mining shall be eventually required for all nodes to resist 50% attack.

Monoxide and OHIE are also multi-chain protocols and more details can be found in Sect. 1.1.

DAG-Based. DAG protocols differ from linear structures of traditional blockchains. Greedy Heaviest-Observed Sub-Tree [28] proposed by Sompolinsky et al. first introduces DAGs to blockchains, using tree rather than linear structure to process transactions. Several DAG-based blockchain protocols with high

Table 2. Comparisons among sharding-based blockchain protocols.

Protocol	Transaction mode	Consensus	Committee resiliency	Total resiliency
Elastico [19]	UTXO	PoW,PBFT	33%	25%
OmniLedger [17]	UTXO	BFT,PoW	33%	25%
RapidChain [33]	UTXO	BFT,PoW	50%	25%
Ostraka [20]	UTXO	PoW	50%	25%
Zilliqa [1]	Account	PoW,PBFT	33%	25%
Monoxide [30]	Account	PoW	N/A	50%
SSChain [5]	UTXO	PoW	N/A	25%
SibCha	UTXO	PoW	N/A	50%

throughput are designed, e.g., SPECTER [27], PHANTOM [29], Conflux [18]. However, DAG-based protocols have no security proofs or only provide heuristic proofs. It is also difficult for them to scale to thousands of nodes and more complicated to handle forks. SibCha ensures transaction ordering, provides formal proofs, has simple fork processing rules, and is scalable.

Sharding. Main idea of sharding is to divide or partition the network into subsets called shards; each shard handles only a different set of transactions. As the number of shards increases, the throughput and storage of the network achieve high efficiency with the possibility of being compromised [1,5,17,19,20,33]. Almost all sharding protocols [1,5,17,19,20,33] (except Monoxide [30]) are required to meet the byzantine validator limit, i.e., maximum ratio of malicious nodes is less than $\frac{1}{3}$, in order to ensure blockchain security. Maximum percentage of malicious validators/nodes that our system can tolerate is less than $\frac{1}{2}$, much larger than the fault tolerance of sharding protocols. Although maximum percentage of malicious nodes that Monoxide [30] can tolerate is less than $\frac{1}{2}$, its Chu-ko-nu mining mechanism will improve the effectiveness of selfish mining and double payment. Mining rules in SibCha do not suffer from this problem. Table 2 compares SibCha with existing sharding-based proposals from the perspectives of transaction mode, consensus, committee resiliency, and total resiliency.

3 System Model and Formal Security Syntax

System Model. We suppose system model and assumptions widely used previously [9,16,25,26,31,32]. $\lambda \in \mathbb{N}$ denotes security parameter (i.e., the length of the hashes) and ρ target difficulty (i.e., a block hashing must have $\log_2 \frac{1}{\rho}$ leading zeros). In a blockchain protocol, interactive Turing machines (nodes) interact with each other in a partial asynchronous network model (see components below).

Environment $\mathcal{Z}(1^\lambda)$: models any external applications running in the system or all external factors related to system execution. It activates nodes p_i as either honest or corrupted and provides system inputs, $1 \leq i \leq n$.

Honest nodes: strictly abide by blockchain protocol; each keeps a copy of its current view and tries to contribute to the chain by building a block.

Adversary $\mathcal{A}$: can deviate arbitrarily from the prescribed protocol. In particular, $\mathcal{A}$ directly controls all actions of each corrupted node. $\mathcal{A}$ can also selectively delay any message sent by honest party for δ seconds (δ defines the network delay).

Random oracle: All nodes can access random oracle $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ in performing and verifying PoW, decoupled as: $H(x)$–computes and outputs $H(x)$, $H.verify(x,y)$–output 1 if $H(x) = y$, and 0 otherwise. We assume that the collision (i.e., $H(x_1) = H(x_2), x_1 \neq x_2$) probability is $e^{-\Omega(\lambda)}$, which is also set as the probability that $H.verify(x,y)$ outputs 1 without calling $H(x)$ first.

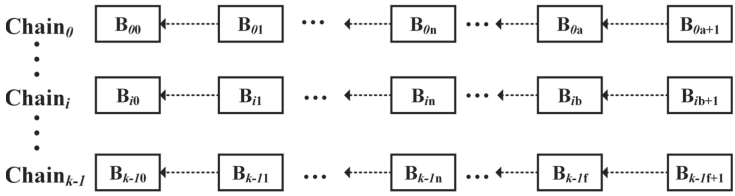

Fig. 1. SibCha chain structure.

Blockchain protocol executes from $\mathcal{Z}$ and in the system, overall computing power is $\mathcal{N}$. To formalize security analysis, we assume that there are a total of n nodes. An adversary could have at most a fraction μ ($<\frac{1}{2}$) of $\mathcal{N}$ computing power, i.e., μn corrupted nodes. Protocol execution consists of a sequence of ticks, where a tick is the amount of time needed in one single execution of random oracle H (queried by an honest node). In each tick, $\mathcal{A}$ can perform μn random oracle queries sequentially, and each honest node shall only send once.

Let δ denote network delay bound, i.e., a node can broadcast a block with maximum delay of δ to other nodes. Let Δ be the result of δ divided by the duration of once execution of H (a tick), e.g., $\Delta = 2 * 10^{12}$ for $\delta = 2$ *seconds* and *a tick* $= 10^{-12}$ *seconds*. Hence, all other honest nodes can receive a message sent by an honest node within Δ ticks. $\mathcal{A}$ and honest nodes are responsible for delivering messages they received to other parties. $\mathcal{A}$ cannot modify the messages broadcast by honest nodes, but it can record or delay them (as long as it finally delivers all messages within Δ ticks). At any time, $\mathcal{A}$ can corrupt an honest node n_i (e.g., control $n_i's$ local state) and its messages; corrupted node can also become honest (i.e., no longer controlled by $\mathcal{A}$) and execute from initial state. $\mathcal{Z}$ can interact with $\mathcal{A}$ or access n_i's local state at any time.

Blockchain Security. A blockchain protocol should ensure the following at any point: any node can output a sequence of total-ordered stable blocks that unlikely become orphaned (the probability ≈ 0). We call these stable blocks as the sequence of confirmed blocks (SCB). A block is stable if there are at least T (a system parameter) blocks following it in the chain. E.g., the SCB in Bitcoin is the remaining chain after we remove the last $T(= 6)$ blocks. A "robust" blockchain should satisfy the properties of *persistence* and *liveness* [15,25,26].

Persistence: once a system node announces a transaction tx_i in SCB of the chain, all remaining nodes will either report tx_i in the same position of that ledger or not report any transaction conflicting with tx_i as stable. A transaction is announced stable only if the block covering it is in SCB.

Liveness: after some pre-defined ticks (determined by confirmation mechanisms), those transactions announced to the chain could be included in their ledgers and will be reported as stable (if queried) by honest nodes.

It is shown [12,14,25] that *persistence* and *liveness* can be deduced from the following properties. **Chain-growth**: in protocol execution, there is a high

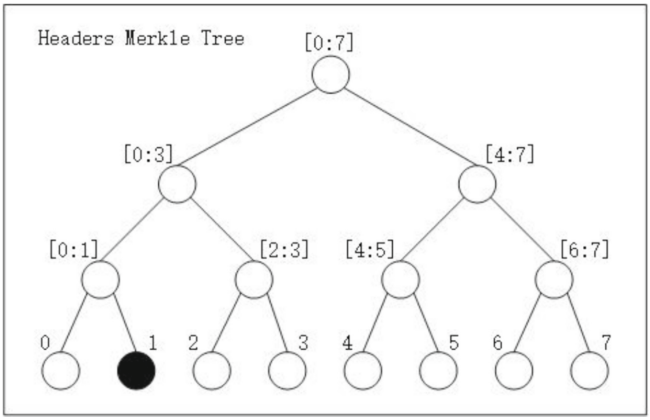

Fig. 2. Hash Merkle Tree (HMT). HMTP of sub-block sb_1 (the black node) is $\{H_0, H_{2:3}, H_{4:7}, H_{0:7}\}$

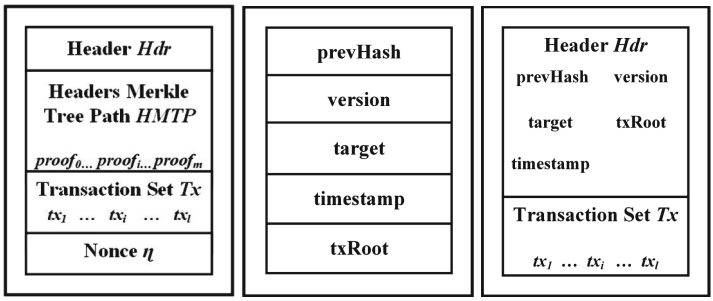

Fig. 3. Block and header structures.

probability on any honest node that the chain grows by T blocks in the last $O(T)$ ticks. **Chain-quality**: given any T consecutive blocks of the chain on any honest node at any time, $\Theta(T)$ blocks are honest (i.e., generated by an honest node; and malicious otherwise). **Consistency**: let n_1 and n_2 be any two honest nodes and C_i be node n_i's local state at time t_i, $i = 1, 2$, $t_1 < t_2$, then C_1 is a prefix of C_2 after removing the last T blocks from the two chains, for any T.

4 System Details

4.1 Addresses, Chain Structure, Block Structrue

In SibCha, an account (i.e., address) is represented by a public key hash of fixed size. SibCha is partitioned into k independent and parallel instances (i.e., consensus regions) based on the rightmost $\log_2 k$ bits of the account to scale the blockchain system. Namely, SibCha runs k independent and parallel instances of single-chain consensus system (see Fig. 1). All transactions of each chain have the same rightmost $\log_2 k$ bits of payers' accounts.

A block is of a data container structure used to store transaction information and ensures the authenticity of block data through PoW puzzle. SibCha block structure includes the following fields (see Fig. 3(a)). *A block header* Hdr (see Fig. 3(b)) contains the same fields as those in Bitcoin except the field *nonce*. *Headers Merkle Tree Path* $HMTP$ (see Fig. 2), contains $1+\log_2 k$ hash values. In particular, given a sub-block sb_i, we have a path from the root to the exact leaf node corresponding to sb_i. Now, for each node on the path, we have one sibling node on the tree. All hash values of sibling nodes (including the root) form a headers Merkle tree path (see Fig 3(a) where $m := \log_2 k$). *Transaction set* Tx is a set of the transactions to be included in the block (at the point, miners should check the legitimacy of these transactions before picking out them from transaction pools). *Nonce* η is used for PoW mechanism.

In SibCha, miners cram some blocks message for each chain into a sub-block (see Fig. 3(c)) which consists of a block header Hdr and a transaction set Tx.

4.2 Mining

In Bitcoin (or Ethereum), PoW puzzle decides a random number η satisfying $H(< Hdr, \eta >) \leq D_\rho$, where D_ρ $(= \log_2 \frac{1}{\rho})$ is PoW target difficulty, and the operator " $\leq$ " treats hash value as an integer.

There are k chains in SibCha and each miner must maintain k sibling chains at any time. Thus during the mining of SibCha, an miner needs to prepare k sub-blocks. E.g., in Fig. 1, the miner prepares one sub-block for each chain, sb_i for chain i. Afterwards the miner computes a Merkle tree using hash values of all sb_i's headers as tree leaves. Let $sb_i.Hdr$ denote the hash of the sub-block's header of ith chain. We write $MerkleTree(H(sb_0.Hdr), ..., H(sb_i.Hdr), ...)$. Let $bksRoot$ denote the root value of the Merkel tree. Finally, the miner computes a random number η_k such that $H(< bksRoot, \eta_k >) \leq D_{k\rho}$ with PoW target difficulty $D_{k\rho}$ $(= \log_2 \frac{1}{k\rho})$. Given η_k above, we have $H_b := H(< bksRoot, \eta_k >)$. Then we use H_b's rightmost $\log_2 k$ bits (corresponding to index j) to pick up 1-out-of-k sub-block (to be announced to the network): sb_j among $sb_0, sb_1, ..., sb_{k-1}$ is the lucky one. The rightmost $\log_2 k$ bits of H_b decides which one of the sub-blocks wins in parallel mining (the rightmost $log_2 k$ bits of $prevHash$ in the sub-block coincides with the rightmost $log_2 k$ bits of H_b). We use $HMTP$ to denote the Merkle tree proof of $sb_j.Hdr$. We mention that folklore Merkle proofs consist of $\log_2 k$ off-path hashing [23]. The sub-block (sb_j), $HMTP$ and η_k are all packed together to form a complete block that would be announced to conforming chain instantaneously.

The hash of a valid block in SibCha should begin with $\log_2 \frac{1}{k\rho}$ leading zeros, where ρ is consistent with that in the Nakamoto consensus protocol (i.e., $D_{k\rho}+k = D_\rho$). Meanwhile, the last $\log_2 k$ bits of the hash of a block in SibCha will be indexed to one of the k chains to which the block will be assigned and from which it will extend. The hash function can be modeled as a random oracle and $\log_2 \frac{1}{k\rho} + \log_2 k = \log_2 \frac{1}{\rho}$. Thus, the probability that a single hash operation will produce a block for any SibCha chain is exactly ρ, which is the same as in Nakamoto consensus. In this context, the choice of k's value does not affect

Table 3. Transaction structure.

Field Name		Size	Description
version		4B	transaction format version
#vin (l)		1–9B	number of transaction input entries $vin[\,]$
$i \in vin[l]$[a]	pre_tx_{id}	32B	hash of a previous transaction
	index	4B	Index of a transaction output within the transaction specified by hash
	sptSigLen	1–9B	length of scriptSig filed in bytes
	sptSig	var	script to satisfy spending condition of the transaction ouput (hash,index)
	nSequence	4B	transaction input sequence number
#vout (m)		1–9B	number of transaction output entries in $vout[\,]$
$o \in vout[m]$	value	8B	amount of btc
	sptPubkeyLen	1–9B	length of scriptPubkey filed in bytes
	sptPubkey	var	script specifying conditions under which the transaction output can be claimed
locktime		4B	timestamp past which transactions can be replaced before inclusion in block

[a]Each entry of $vin[l]$ is a 5-tuple $(pre\_tx_{id}, n, sptSigLen, sptSig, nSequence)$.

the security of SibCha, whereas it does have close connections with network bandwidth, block size, and block interval (see Sect. 6.2).

Similarly, the block interval for any given chain in SibCha will be the same as in the Nakamoto consensus, i.e., the block interval in Nakamoto consensus and any chain in SibCha is also $c \cdot \delta$ (c is a constant). Thus, SibCha has at least k blocks that would be generated every $c \cdot \delta$ times. Our experiments demonstrate (see Sect. 6.1) that the negative impact of propagating parallel blocks on block propagation delay δ is small when system network bandwidth is fully utilized.

All sibling chains use same PoW target difficulty so that they are at the same security level. Difficulty adjustment mechanism is similar to that in Bitcoin.

4.3 Transaction

Bitcoin uses unspent transaction output (UTXO) model [2] and a transaction spends the output of some previous transactions to produce new output that can be spent in future transactions. Table 3 illustrates Bitcoin's transaction structure. Now full nodes in Bitcoin can keep a copy of UTXO set [8] which might be used as a lightweight technique in verifying transactions or generating new transactions rather than arduous traversing of entire chain. UTXO set is stored in the chainstate, a database that provides persistent key-value storage. The key of a record is a 32-byte transaction hash. The value of a record stores metadata about the transaction (height and whether it is coinbase or not) and a

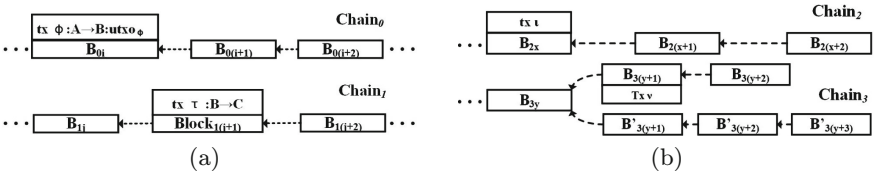

Fig. 4. Inter-chain transaction.

compressed representation of the UTXOs of the transaction. A transaction may output multiple UTXOs by adding prefix/suffix to the key.

In SibCha, there exist inter-chain transactions. Take chains 0 and 1 in Fig. 4(a) as example. Payment transactions of A and B are stored in chain 0 and chain 1, respectively. Block B_{0i} in chain 0 records the transaction ϕ from payer A to payee B, so an UTXO$_\phi$ of B is generated. Later, B spends UTXO$_\phi$ when paying C, and this transaction will be recorded in block $B_{1(j+1)}$ of chain 1. What's worse is that inter-chain transactions exist (see Fig. 4(b)). Consider Block B_{2x} in chain 2 and block $B_{3(y+1)}$ in chain 3. UTXO generated by transaction υ in block $B_{3(y+1)}$ is spent in transaction ι in block B_{2x}. But chain 3 has forked. Block $B_{3(y+1)}$ becomes an orphan block and there is no transaction υ in block $B'_{3(y+1)}$. This makes B_{2x} and all subsequent blocks in chain 2 invalid.

To ensure transaction security (and cope with above-mentioned potential vulnerability), we promote a shift towards UTXO records. We swap out block height in the field *value* for a block hash before carving the UXTOs of transaction output in UTXO set. Now UTXO records are of the format (key: tx_{id}, value: *utxo*, h_b,...), where *utxo* is exactly generated by the transaction with tx_{id} being its 32-byte hash, and h_b is the block hash that covers the exact transaction. On other hand, for each inter-chain transaction (say v), one can perceive all its previous transactions (by pre_tx_v in $v.vin[\,]$) and it is mandatory in checking v's legitimacy that all blocks covering these prior transactions should be in SCB.

To simplify mining rules, SibCha does not harshly require that each block must point to some blocks in each chain. Therefore the sequential nature of SibCha blocks is not as strictly consistent as Bitcoin's single-chain structure, whereas the sequential nature of blocks inside each SibCha chain is as strictly consistent as the single-chain structure of Bitcoin. To facilitate later verification of SibCha blocks (i.e., to facilitate transaction tracking in each block or checking entire blockchain) and strengthen block dependency, we adapt the input of transaction structure (Table 3). In particular, we introduce to each entry of $vin[\,]$ an extra field *pre_tx_block* besides existing 5-tuple. *pre_tx_block* takes as its value the block hashing that covers the transaction determined by pre_tx_{id}.

4.4 Validation

When a node receives a block broadcast by other miners, it shall verify the block to defend against malicious miners. SibCha has the following verification. The block provides a legitimate nonce that can answer the PoW puzzle with target difficulty: first compute $H(Hdr)$, then recover *bksRoot* by using $H(Hdr)$ and corresponding proof path $HMTP$, and thereby check its satisfiability. Header fields (i.e., version, target, and timestamp) are set correctly. All included transactions are correct and can reconstruct the exact *txRoot*.

One should identify inner-chain transactions and inter-chain transactions. For any transaction v: if the rightmost $\log_2 k$ bits of *pre_tx_block* in e ($\forall e \in v.vin[\,]$) and the rightmost $\log_2 k$ bits of the hashing of the block covering v are the same, then we view v as an inner-chain transaction; v is a inter-chain transaction if it

is not an inner-chain transaction. For a inter-chain transaction (we use similar steps to check inner-chain transactions, neglecting the last step), its input should be legitimate, i.e., we verify each item (ei) of the transaction $vin[\,]$ ($ei \in vin[\,]$): check the existence of the UTXO corresponding to ei in UTXO set based on pre_tx_{id} in ei; verify that the block corresponding to the pre_tx_block field in ei is in SCB of the chain in which it is located.

4.5 Confirmation Rule and Resolution

SibCha attempts to minimize the correlation among chains. Each transaction is supposed deliberately to be recorded (after mined) in some entitled chains based on the payer's address of the transaction. Our particular processing of inter-chain transactions and validation rules enable the miners not to be imprisoned in a crosschain-ordering straitjacket when performing mining tasks.

Our confirmation rule is as simple as that in Nakamoto consensus. There is no need in confirming a block to consider block ordering among the chains and the last T blocks of each chain. There is also no need to make particular effort into possible forks among sibling chains (which cannot be circumvented in OHIE) and we simply choose the longest one for each chain as in Nakamoto.

4.6 Optimized Interleaving

In Nakamoto consensus, all blocks have strict sequential nature due to the structure of append-only reverse chained table. For other schemes maintaining k Nakamoto consensus instances, all blocks generated by k chains also possess strict sequential nature, and in particular, the blocks between chains are organized in a strictly sequential way. For example, OHIE manipulates the field *trailing* to bound the sequential nature of blocks. This makes the chains difficult to achieve concurrent evolving.

SibCha reduces sequential correlation of generated blocks among k sibling chains and induces miners to participate honestly in mining on all sibling chains instead of one single chain alone. We do not strictly require the sequential nature of blocks among k chains but only within a single chain (i.e., miners who find multiple blocks with different last k bits of the hash value within propagation delay are all compliant blocks). Even if two miners mine blocks at almost same moment there is a high probability that both blocks are legitimate (the probability that only one of the two blocks is legitimate is $\frac{1}{k^2}$). Obviously, concurrent evolving capability of our system is strictly related to k. The larger the value of k, the higher the concurrent evolving capability and the higher the throughput.

The interleaving among SibCha sibling chains merely requires that a transaction (with some addresses being a payer) could only be deemed as legitimate if the latest transaction(s) with the exact address being a payee in sibling chains have already been confirmed (with ordering guarantee of payee-then-payer inter-chain transactions). For this, SibCha tweaks the transaction input and UTXO record format (without modifying the structures of blocks and chains or even

decoupling the transactions) and then hashing localization suffices to guarantee the ordering of payee-then-payer inter-chain transactions. We mention that OHIE [31] claps extra *trailing* and *root* fields on the block header (to keep the view of all chains up-to-date) and invents the structure *block attachment* (including *rank*, *next_rank* and the proof path of the Merkle tree for *root*) in solving multi-chain interleaving problem that inexorably links to block ordering and block update of the system view. Monoxide [30] decouples a cross-zone transaction into initiative transaction and relay transaction (each involves a single zone) and it may thereby take double incentive to succeed and at least $2T$ times to confirm a cross-zone transaction by two-phase PoW puzzle solving.

5 Formal Security

We use (ρ, λ, T)-Nakamoto to denote Nakamoto consensus protocol. We define $(k, \rho, \lambda, T)-$SibCha to denote our protocol. Let SCB_i denote the SCB of chain i in SibCha and $\text{SCB}_{\text{SibCha}}$ denote the SCB of SibCha, i.e., $\text{SCB}_{\text{SibCha}}=\text{SCB}_0 \cup ...\cup \text{SCB}_i \cup ...\cup \text{SCB}_{k-1}$. We have the following theorem.

Theorem 1. *Assume a constant $\mu < \frac{1}{2}$. For any n and Δ, there exists constant c such that $0 \leq \rho < \frac{1}{c\Delta n}$ holds, the probability that all the following properties are satisfied by $(k, \rho, \lambda, T)-$SibCha is at least $1 - e^{-\Omega(\lambda)} - e^{-\Omega(T)}$.*
***Chain-growth**: For any honest node and any sibling chain, the number of blocks increased in every $\frac{2T}{\rho n}$ ticks is $\geq T$.*
***Chain-quality**: For any honest node and any T consecutive blocks of each of k chains, at least $\frac{1-2\mu}{1-\mu}T$ blocks are honest.*
***Consistency**: For any T, at any two ticks t_1 and t_2, let C_1 (and C_2) denote the SCB_{SibCha} of any honest node n_1 (n_2, resp.) at t_1 (t_2, resp.). If $n_1 = n_2$ and $t_1 < t_2$, C_1 is a prefix of C_2. If $n_1 \neq n_2$ and $t_1 + \Delta \leq t_2$, C_1 is a prefix of C_2.*
***Chain-quality-growth**: Let S be all SCB in k sibling chains on any given honest node, i.e., $S=SCB_0 \cup ...\cup SCB_i \cup ...\cup SCB_{k-1}$. Then, at least $k\frac{1-2\mu}{1-\mu}T$ honest blocks will be added to S after every $\frac{2T}{\rho n}$ ticks.*

Proof Highlights. We have two main steps in proving Theorem 1. The first step (Lemma 1 along with Lemma 2 and Lemma 3) shows that the security of any chain in $(k, \rho, \lambda, T)-$ SibCha can be deduced from that of (ρ, λ, T)-Nakamoto [25,31,32]. More precisely, for any adversary $\mathcal{A}$ that attacks any given chain i ($0 \leq i \leq k-1$) in the execution of $(k, \rho, \lambda, T)-$SibCha, there exists some adversaries $\mathcal{A}'$ with the following properties in attacking (ρ, λ, T)-Nakamoto. Except for some exponentially small probability and after a proper mapping from blocks in $(k, \rho, \lambda, T)-$SibCha to blocks in (ρ, λ, T)-Nakamoto, chain i in the execution of $(k, \rho, \lambda, T)-$SibCha against $\mathcal{A}$ follows exactly the same distribution as the behaviour of the (single) chain when executing $(\rho, \lambda, T)-$Nakamoto against $\mathcal{A}'$. Such a reduction implies that the existing properties about (ρ, λ, T)-Nakamoto carry over directly to each individual chain in $(k, \rho, \lambda, T)-$SibCha.

The second step (i.e., Lemma 4) says consistency. It shows that the SCB generated in SibCha must satisfy the properties in Theorem 1 except for

some exponentially small probability, given that each individual chain in $(k, \rho, \lambda, T)-$SibCha maintains all existing properties (in Theorem 2 of Sect. A) of (ρ, λ, T)-Nakamoto.

Notations. Random variable $Exe(\text{SibCha}, k, \rho, \lambda, T, \mathcal{A})$ represents joint state of all nodes in entire execution process, resulted from $\mathcal{A}$ running SibCha. Random variable $C_i(Exe(\text{SibCha}, k, \rho, \lambda, T, \mathcal{A}))$ is defined as joint state of chain i for each honest node during entire execution. Similarly, we define random variables in Nakamoto consensus: $Exe(Nakamoto, \rho, \lambda, T, \mathcal{A}')$ and $C(Exe(Nakamoto, \rho, \lambda, T, \mathcal{A}'))$. $\mathcal{A}'$ is composed of $\mathcal{A}$ and middle-box. $\mathcal{A}'$ will treat $\mathcal{A}$ as a black-box, and middle-box simulates all honest nodes and invokes $\mathcal{A}$ to simulate the execution of $(k, \rho, \lambda, T)-$SibCha. For two random variables X and Y in finite field Z, we define their variation distance as $||X - Y|| := \frac{1}{2} \sum_{z \in Z} |Pr[X = z] - Pr[Y = z]|$. If λ is used to parameterize X and Y, $X(\lambda)$ is said statistically close to $Y(\lambda)$ iff $||X - Y|| = e^{-\Omega(\lambda)}$. We write $X(\lambda) \overset{\text{S}}{\sim} Y(\lambda)$.

Lemma 1. *Assume any $\mathcal{A}$ that attacks any given chain $i (0 \leq i \leq k-1)$ in $(k, \rho, \lambda, T)-$SibCha. Then (ρ,λ,T)-Nakamoto has $\mathcal{A}_i^{'}$, which makes random variables $C_i(Exe(\text{SibCha}, k, \rho, \lambda, T, \mathcal{A}))$ and $\sigma_i^{\tau}(C(Exe(Nakamoto, \rho, \lambda, T, \mathcal{A}_i^{'})))$ statistically close, where τ is the randomness in $Exe(Nakamoto, \rho, \lambda, T, \mathcal{A}_i^{'})$ and $\sigma_i^{\tau}()$ is a 1-to-1 mapping from each block B' in (ρ,λ,T)-Nakamoto to certain block $B (= \sigma_i^{\tau}(B'))$ in $(k, \rho, \lambda, T)-$SibCha. $\sigma_i^{\tau}()$ can guarantee that 1) if $\sigma_i^{\tau}(B')$ is honest, so is B', and 2) if B's parent block is A, the parent block of B' is A'.*

Lemma 1 can be proved by several hybrid protocol executions (adjacent executions are statistically close). We defer the details to Sect. A for ease of readability.

Lemma 2. *Each chain in $(k, \rho, \lambda, T)-$SibCha satisfies the properties of Theorem 2 (in Sect. A) with probability at least 1-$e^{-\Omega(\lambda)}$-$e^{-\Omega(T)}$. $(k, \rho, \lambda, T)-$SibCha further satisfies the properties of chain-quality and chain-growth in Theorem 1 with probability at least 1-$e^{-\Omega(\lambda)}$-$e^{-\Omega(T)}$.*

Proof. Lemma 1 guarantees that when the adversary's computing power is less than $\frac{1}{2}\mathcal{N}$, the characteristics of each sibling chain is consistent with that of Nakamoto chain. Therefore, each chain in SibCha satisfies Theorem 2. We can thus conclude that SibCha satisfies chain-growth and chain-quality in Theorem 1.

Lemma 3. *If each sibling chain has the properties in Theorem 2, $(k, \rho, \lambda, T)-$SibCha satisfies chain-quality-growth in Theorem 1 with probability at least 1-$e^{-\Omega(\lambda)}-e^{-\Omega(T)}$.*

Proof. It is obvious from Lemma 2.

Lemma 4. *$(k, \rho, \lambda, T)-$SibCha satisfies chain-consistency in Theorem 1 with probability at least 1-$e^{-\Omega(\lambda)}$-$e^{-\Omega(T)}$.*

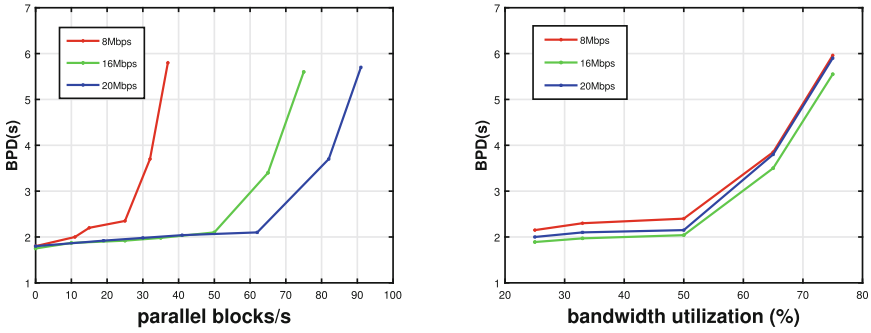

Fig. 5. BPD under node bandwidth configuration (8–20 Mbps).

Proof. C_1 (*and* C_2) denotes the $\text{SCB}_{\text{SIBCHA}}$ of any honest node p_1 ($p_2, resp.$) at tick t_1 ($t_2, resp.$). Lemma 2 shows that with probability$\geq$ 1-$e^{-\Omega(\lambda)} - e^{-\Omega(T)}$, each chain in SIBCHA satisfies Theorem 2. When $p_1 = p_2$ and $t_1 < t_2$, SCB_i $(0 \leq i \leq k-1)$ at tick t_1 must be the prefix of SCB_i at tick t_2. It is thereby easy to deduce that $\text{SCB}_{\text{SIBCHA}}$ at tick t_1 on any honest node is the prefix of $\text{SCB}_{\text{SIBCHA}}$ at tick t_2.

On the other hand, when $p_1 \neq p_2$ and $t_1 + \Delta < t_2$. According to Lemma 2, we can get that the SCB_i at tick t_1 must be the prefix of SCB_i at tick t_2. Since there may be multiple blocks (up to k) being mined at the same time, different nodes would have different views. Maximum network delay in SIBCHA is Δ tick, i.e., the messages published no later than t_1 would be received by all nodes in the network at tick t_2. Therefore, multiple blocks being announced at the same time will not affect SIBCHA consistency. Thus $\text{SCB}_{\text{SIBCHA}}$ at tick t_1 on any honest node is the prefix of $\text{SCB}_{\text{SIBCHA}}$ at tick t_2.

6 Experimental Evaluation

We deploy SIBCHA prototype on Amazon's EC2 using 1000 m 4.2 x large virtual machines (VMs), each with 8 cores and up to 1 Gbps of network throughput. For the experiments to determine the parameters in SIBCHA, 1,000 nodes are run on 20 EC2 instances. We can thus evaluate SIBCHA performance by running 10,000 to 50,000 SIBCHA nodes on 1,000 EC2 virtualizations, each connected to 8 randomly selected peers forming a P2P network with a bandwidth of up to 20 Mbps per node. Above settings match those of OHIE [31].

6.1 Block Size, Block Interval, and Parallel Propagation of Blocks

We first measure block propagation delay (BPD), i.e., individual block arrival delay for 99% of the node network bandwidth with 8 Mbs to 10 Mbps. We observe that for block sizes from 10 KB to 64 KB, BPD is about 1.6–2.2 s. It does not increase significantly with the number of nodes in the network.

Table 4. SibCha vs OHIE.

bandwidth(Mbps)	k	txns per second(tps)		txns throughput(Mbps)	
		OHIE	SibCha	OHIE	SibCha
8	250	960	958	3.84	3.83
12	370	1460	1454	5.84	5.82
16	500	1940	1946	7.76	7.78
20	620	2420	2435	9.68	9.74

For example, in our experiment with 50,000 nodes, the BPD values are about 3.0–3.3 s for 10–20 KB blocks. These results are similar to those in OHIE. We use the same configuration as OHIE for comparison purposes, e.g. a block size of 20KB, a time interval of about 10 s per chain to generate a block, $u = 0.43$, etc.

We further check whether propagating multiple parallel blocks in SibCHa has a significant impact on the BPD compared to propagating single block. Figure 5(a) shows the BPD of propagating in parallel blocks of 20KB size and the results we obtained are almost the same as those in OHIE: the number of parallel propagated blocks within a certain range does not show significantly negative impact on the BPD. Take the bandwidth of 20 Mbps as example and one hardly sees conspicuous variance for parallel propagation of 0–62 blocks.

Figure 5(b) depicts bandwidth occupied by parallel block propagation on BPD. The results are almost identical to those in OHIE (we use the same configuration to reproduce OHIE experiments). This demonstrates the significance of parallel chain design as with the same bandwidth configuration, parallel block propagation could efficiently utilize about 50% of the original bandwidth without strikingly negative impact on the BPD.

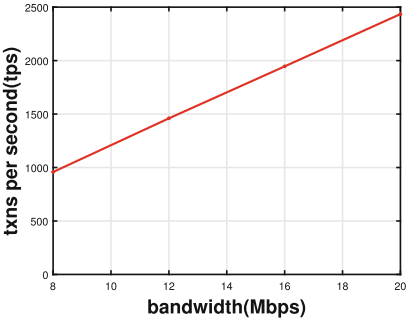

Fig. 6. SibCha throughout

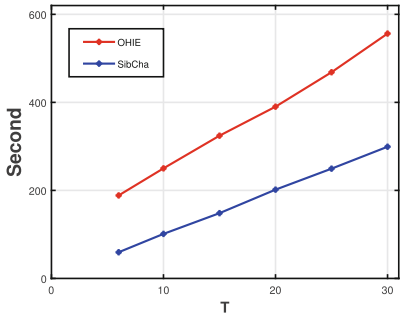

Fig. 7. Confirmation latency.

6.2 Throughput

Next we check the throughput of SibCha. As in OHIE, we run 12000 nodes on 1000 EC2 instances, with a bandwidth configuration of 8–20 Mbps per node, a block size of 20 KB, a block interval of 10 s per chain, and an average transaction size of 500 bytes (similar to Bitcoin). According to Sect. 6.1, parallel block propagation using 50% of available network bandwidth does not have a significant negative impact on the BPD, so we can set the k's values based on the available network bandwidth, block size and block interval time, i.e., $k * block\ size\ /\ block\ interval \approx bandwidth/2$. For 8/12/16/20 Mbps bandwidths, we set k = 250, 370, 500 and 620 respectively. As k is not a power of 2, the last 48 bits of block hash are used to determine chain index i to which the block is assigned, $i = H(\text{block}) \mod k$.

As shown in Fig. 6, the throughput of SibCha increases linearly with bandwidth. The throughput reaches about 2435 transactions (txns) per second at a bandwidth of 20 Mbps. A further detailed comparison with OHIE in Table 4 says that SibCha and OHIE contribute almost the same throughput. Thus, characterized by its minimal interleaving of sibling chains, simple block structure, simple confirmation mechanism, and simple fork processing, SibCha still retains the property of high throughput, as expected.

6.3 Confirmation Latency

Confirmation blocks are T blocks removed from the end of the chain, e.g., T=6 in Bitcoin, and T = 10 to 15 in Ethereum. T's value only has an impact on confirmation latency, but not on the throughput of SibCha. Our experiments measure time consumption required for a block to be confirmed in a 20 Mbps bandwidth configuration. Figure 7 shows confirmation latencies for T = 6–30 which demonstrates SibCha's much lower acknowledgement latency than OHIE. For example, the confirmation latency of OHIE is 3.16× that of SibCha for T = 6, and 1.86× that of SibCha for T = 30.

7 Conclusion

We propose SibCha, a PoW-based blockchain protocol consisting of multiple parallel instances of Nakamoto consensus. It implements simpler structure and confirmation rules and provides more security guarantees than existing similar solutions. SibCha scales linearly with available bandwidth and achieves much lower block confirmation latency.

Acknowledgement. The work is supported by Science and Technology Commission of Shanghai Municipality (23511100200), Shanghai Pilot Program for Basic Research (TQ20240212), Shanghai Key Laboratory of Trusted Data Circulation and Governance and Web3, and Shanghai Municipal Education Commission (2021-01-07-00-08-E00101).

A Proof of Lemma 1

From [25, Corollary 3] and [32, Theorem 2], we have the following theorem.

Theorem 2. *Let $\mu < \frac{1}{2}$ be a constant. Then for every n, Δ, there exists constant c such that $0 \leq \rho < \frac{1}{c\Delta n}$ holds, the probability that all the following properties are satisfied by (ρ,λ,T)-Nakamoto is at least $1 - e^{-\Omega(\lambda)} - e^{-\Omega(T)}$.*
***Chain-growth**: For any honest node, the length of the chain increased in every $\frac{2T}{\rho n}$ ticks is at least T blocks.*
***Chain-quality**: For any honest node and T consecutive blocks in the chain, at least $\frac{1-2\mu}{1-\mu}T$ blocks are honest.*
***Consistency**: For any T, at any ticks t_1 and t_2, let C_1 (C_2) denote the SCB of any honest node n_1 (n_2, reps.) at t_1 (t_2, resp.). If $n_1 = n_2$ and $t_1 < t_2$, C_1 is a prefix of C_2. If $n_1 \neq n_2$ and $t_1 + \Delta \leq t_2$, C_1 is a prefix of C_2. Combining **chain-growth** and **chain-quality**, we have **chain-quality-growth**: Let S be the SCB in the chain of any given honest node. Then, at least $\frac{1-2\mu}{1-\mu}T$ honest blocks will be added to S after every $\frac{2T}{\rho n}$ ticks.*

Based on Theorem 2, we can construct our proof of Lemma 1. It suffices to prove the correctness of Lemma 1 for $i = 0$. We will present the proof by designing several hybrid executions. Let H denote random oracle used in $Exe(\text{SibCha}, k, \rho, \lambda, T, \mathcal{A})$ and H' in $Exe(Nakamoto, \rho, \lambda, T, \mathcal{A}_0')$.
Hybrid1: $Exe(\text{SibCha}, k, \rho, \lambda, T, \mathcal{A})$.
Hybrid2: same as Hybrid1, except the invoking of random function. Three tables, T_1, T_2 and T_3, will be used in Hybrid2.
T_1: initially empty and with entries of the form (x, y).

In Hybrid2, $H(x)$ responds to any query x as below: 1) if $(x, y) \in T_1$ exists, return y directly (y must be unique), otherwise we choose a uniformly random binary string y of length λ, 2) check whether T_1 contains (x_2, y): if yes, we re-select a uniformly random y until there is no collision, and 3) insert (x, y) into T_1, and return y. Let bad_1 be the event that Hybrid2's execution includes some invocations $H(x)$ like $x_1 \neq x_2$ and $H(x_1) = H(x_2)$. We have $\Pr[\text{bad}_1] = e^{-\Omega(\lambda)}$.

For the query of $H.verify(x, y)$: if $(x, y) \in T_1$, return true, otherwise false. Define bad_2 to be the event that Hybrid2's execution includes some invocations $H.verify(x, y)$ s.t. $H(x)$ has not been invoked before, but $H.verify(x, y)$ returns true. We have $\Pr[\text{bad}_2] = e^{-\Omega(\lambda)}$.
T_2: It contains entries of the form $(B, B.hash)$ and is initialized as the genesis block and its hash value of chain 0.

For the query $H.verify(B, y)$: if $(B, y) \in T_2$, return true, otherwise false. Define bad_3 to be the event that Hybrid2's execution includes some invocations $H.verify(B, y)$ such that $H(B)$ has not been invoked before, but $H.verify(B, y)$ returns true. We have $\Pr[\text{bad}_3] = e^{-\Omega(\lambda)}$.
T_3: contains entries of the form (B, B') and is initialized as (B_0, B_0'), where B_0 is the genesis block of chain 0 and B_0' is constructed from B_0 (described below).

In Hybrid2, $H(B)$ responds to any query B as below: 1) if $(B, y) \in T_2$ exists, return y directly, 2) otherwise check if there is $(B, B') \in T_3$: if not, construct B'

based on B (described below) and put (B, B') into T_3, then invoke $H'(B')$ and obtain y and further add (B, y) to T_2, and return y as the answer to $H(B)$.

To make our reduction rigorous, we adapt the original Nakamoto consensus to fit the structure of SibCha (it can be assumed that the block structure of Nakamoto here is not different from that in SibCha). We construct B' based on B. First, set $B'.Tx' \leftarrow B.Tx$, and then construct Hdr of B' based on $B.Hdr$ (i.e., $B'.Hdr'.prevHash' = B.Hdr.prevHash$). Set $B'.HMTP' = B.HMTP$. Finally, let $B'.\eta'$ be a random nonce chosen uniformly.

Define bad_4 to be the event that during Hybrid2 execution, T_3 contains $(B_1, B_1^{'})$ and $(B_2, B_2^{'})$ s.t. $B_1^{'} = B_2^{'}$ or $B_1 = B_2$. Each B has a new nonce and output length of the random oracle is $poly(\lambda)$. Thus $\Pr[\text{bad}_4] = e^{-\Omega(\lambda)}$.

Hybrid1 $\overset{\text{S}}{\sim}$ **Hybrid2.** If none of the four events bad1 $\sim$ bad4 occur, joint distribution of the returned values of all the invoking ($H()$ and $H.verify()$) in Hybrid1 is the same as in Hybrid2. Thus Hybrid2 and Hybrid1 are very close. None of the four bad events occur with probability no less than $1 - e^{-\Omega(\lambda)}$, thus Hybrid1 and Hybrid2 are statistically close.

Hybrid3: same as Hybrid2, except that nodes in Hybrid3 run Nakamoto protocol and $\mathcal{A}_0^{'}$ consists of a middle-box and $\mathcal{A}$ in Hybrid2. Let p' be any honest node in Hybrid3 that runs $(\rho, \lambda, T)-$Nakamoto and maintains one chain, and p be honest in Hybrid2 that runs $(k, \rho, \lambda, T)-$SibCha and maintains k sibling chains. All honest nodes in Hybrid2 are simulated by the middle-box in Hybrid3.

In each tick, for any new block B' broadcast by node p', the middle-box: 1) simulates p running Mining() and returns $B.hash$ when p invokes $H(B)$; 2) adds $(B, B.hash)$ constructed by p during Mining() to T_2; 3) adds (B, B') in T_3; 4) can forward any message broadcast by p to $\mathcal{A}$.

In each tick, for any new block being not broadcast by node p': 1) the middle-box simulates p running Mining() and returns $B.hash$ (a uniformly random binary string of length λ) when p invokes $H(B)$, and the middle-box will recompute $B.hash$ until $B.hash$ does not have 2 leading zeros and k trailing zeros (i.e., $B.hash$ cannot solve PoW puzzle), 2) the middle-box adds $(B, B.hash)$ constructed by p during Mining() to T_2, 3) the middle-box constructs B' based on B, similar to the process of invoking $H(B)$ in Hybrid2, then adds (B, B') to T_3, 4) the middle-box can forward any message broadcast by p to $\mathcal{A}$.

In Hybrid3, there are two ways to add a new entry (B, B') in T_3, i.e., when either any honest node p or the adversary $\mathcal{A}$ invokes $H(B)$. The nonces in B and B' are uniformly random when p invokes $H(B)$. When $\mathcal{A}$ invokes $H(B)$, if T_3 doesnot contain any entry about B, new entry (B, B') will be added to T_3. Thus, B' always has a random nonce when it is in (B, B'). Let bad_5 be the event that in Hybrid3, T_3 contains $(B_1, B_1^{'})$ and $(B_2, B_2^{'})$ s.t. $B_1^{'}{=}B_2^{'}$ or $B_1{=}B_2$. Output length of random oracle is poly(λ) and we have $\Pr[\text{bad}_5] = e^{-\Omega(\lambda)}$.

Joint State of all Honest Nodes in Hybrid2 and Hybrid3 are Identically Distributed. It is easy to see that Hybrid3 is just the execution of (ρ, λ, T)-Nakamoto against $\mathcal{A}_0^{'}$. Moreover, it can be verified that $\mathcal{A}$'s view in Hybrid2 has the same distribution as $\mathcal{A}_0^{'}$'s view (composed of $\mathcal{A}$ and the middle-box) in Hybrid3, provided that neither bad_4 nor bad_5 occur. Therefore, under the same

condition, $\mathcal{A}$'s state in Hybrid2 follows the identical distribution as $\mathcal{A}_0^{'}$'s state in Hybrid3, and the states of all honest nodes in Hybrid2 and all honest nodes (simulated) in Hybrid3 have the same distribution.

B' is an honest block iff $\sigma_0^\tau(B')$ is an honest block and $B'.prevHash == \sigma_0^\tau(B').prevHash$. In the case that the event bad_5 does not occur in Hybrid3, the entries of T_3 are composed of 2-tuple $(\sigma_0^\tau(B'), B')$, $\forall B' \in SCB'$ on node p'. In addition, σ_0^τ can ensure that B' is an honest block if $\sigma_0^\tau(B')$ is an honest block and that $B'.prevHash = \sigma_0^\tau(B')$.prevHash.

Given all above, we have (bad1 $\sim$ bad5 do not occur): 1) Hybrid1=Exe (SibCha, $k, \rho, \lambda, T, \mathcal{A}$); 2) Hybrid1$\overset{S}{\sim}$Hybrid2; 3) joint state of all simulated honest nodes in Hybrid3 has the same distribution as joint state of all honest nodes in Hybrid2; 4) when an honest node in Hybrid3 adds B' to $chain'_0$, the corresponding simulated honest node in Hybrid3 adds σ_0^τ(B') (i.e., B) to its $chain_0$; 5) B' is honest if σ_0^τ(B') is honest; 6) Hybrid3 = $Exe(Nakamoto, \rho, \lambda, T, \mathcal{A}_0^{'})$.

This completes the proof of Lemme 1 for i=0, and above proof can be extended to all sibling chains.

References

1. The zilliqa technical whitepaper (2017). https://docs.zilliqa.com/whitepaper.pdf
2. Atzei, N., Bartoletti, M., Lande, S., Zunino, R.: A formal model of bitcoin transactions. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. LNCS, vol. 10957, pp. 541–560. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-58387-6_29
3. Bagaria, V.K., Kannan, S., Tse, D., Fanti, G.C., Viswanath, P.: Prism: deconstructing the blockchain to approach physical limits. In: Conference on Computer and Communications Security, pp. 585–602. ACM (2019)
4. Boneh, D., Shoup, V.: A graduate course in applied cryptography. https://crypto.stanford.edu/~dabo/cryptobook/draft_0_3.pdf
5. Chen, H., Wang, Y.: SSChain: a full sharding protocol for public blockchain without data migration overhead. Pervasive Mob. Comput. **59**, 101055 (2019)
6. Chitra, T., Quaintance, M., Haber, S., Martino, W.: Agent-based simulations of blockchain protocols illustrated via kadena's chainweb (2019). https://d31d887a-c1e0-47c2-aa51-c69f9f998b07.filesusr.com/ugd/86a16f_3b2d0c58179d4edd9df6df4d55d61dda.pdf
7. Croman, K., et al.: On scaling decentralized blockchains. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 106–125. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_8
8. Delgado-Segura, S., Pérez-Solà, C., Navarro-Arribas, G., Herrera-Joancomartí, J.: Analysis of the bitcoin UTXO set. In: Zohar, A., et al. (eds.) FC 2018. LNCS, vol. 10958, pp. 78–91. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-58820-8_6
9. Deuber, D., Magri, B., Thyagarajan, S.A.K.: Redactable blockchain in the permissionless setting. In: 2019 IEEE S&P, pp. 124–138. IEEE (2019)
10. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-ng: a scalable blockchain protocol. In: 13th USENIX NSDI, pp. 45–59 (2016)

11. Fitzi, M., Gaži, P., Kiayias, A., Russell, A.: Parallel chains: improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, Report 2018/1119 (2018)
12. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
13. Gencer, A.E., Basu, S., Eyal, I., van Renesse, R., Sirer, E.G.: Decentralization in bitcoin and Ethereum networks. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. LNCS, vol. 10957, pp. 439–457. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-58387-6_24
14. Kiayias, A., Panagiotakos, G.: Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019 (2015)
15. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
16. Kiffer, L., Rajaraman, R., Shelat, A.: A better method to analyze blockchain consistency. In: Conference on Computer and Communications Security, pp. 729–744. ACM (2018)
17. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: a secure, scale-out, decentralized ledger via sharding. In: IEEE S&P, pp. 583–598 (2018). https://doi.org/10.1109/SP.2018.000-5
18. Li, C., Li, P., Xu, W., Long, F., Yao, A.C.: Scaling nakamoto consensus to thousands of transactions per second. CoRR **abs/1805.03870** (2018), http://arxiv.org/abs/1805.03870
19. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: 2016 ACM CCS, pp. 17–30 (2016)
20. Manuskin, A., Mirkin, M., Eyal, I.: Ostraka: secure blockchain scaling by node sharding. In: IEEE European S&P, pp. 397–406. IEEE (2020)
21. Martino, W., Quaintance, M.: Chainweb: A proof-of-work parallel-chain architecture for massive throughput (2018). https://d31d887a-c1e0-47c2-aa51-c69f9f998b07.filesusr.com/ugd/86a16f_029c9991469e4565a7c334dd716345f4.pdf
22. Martino, W., Quaintance, M.: Chainweb protocol security calculations (2018). https://d31d887a-c1e0-47c2-aa51-c69f9f998b07.filesusr.com/ugd/86a16f_26d87f20cf8548d2927e28152babf533.pdf
23. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
24. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
25. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: EUROCRYPT, vol. 10211, pp. 643–673 (2017). https://doi.org/10.1007/978-3-319-56614-6_22
26. Pass, R., Shi, E.: Fruitchains: a fair blockchain. In: Principles of Distributed Computing, USA. pp. 315–324. ACM (2017)
27. Sompolinsky, Y., Lewenberg, Y., Zohar, A.: Spectre: serialization of proof-of-work events: confirming transactions via recursive elections (2016)
28. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in bitcoin. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 507–527. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_32

29. Sompolinsky, Y., Zohar, A.: PHANTOM: a scalable blockdag protocol. IACR Cryptol. ePrint Arch, p. 104 (2018)
30. Wang, J., Wang, H.: Monoxide: scale out blockchains with asynchronous consensus zones. In: 16th USENIX NSDI, pp. 95–112. USENIX Association (2019)
31. Yu, H., Nikolic, I., Hou, R., Saxena, P.: OHIE: blockchain scaling made simple. In: 2020 IEEE S&P, pp. 90–105. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00008
32. Yu, H., Nikolic, I., Hou, R., Saxena, P.: OHIE: blockchain scaling made simple. CoRR **abs/1811.12628** (2018). http://arxiv.org/abs/1811.12628
33. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: scaling blockchain via full sharding. In: 2018 ACM CCS, pp. 931–948 (2018)