A Generic Paradigm for Blockchain Design*

Phuc Thai
Virginia Commonwealth University
thaipd@vcu.edu

Laurent Njilla
Air Force Research Laboratory
laurent.njilla@us.af.mil

Tuyet Duong
Virginia Commonwealth University
duongtt3@vcu.edu

Lei Fan
Shanghai Jiao Tong University
fanlei@sjtu.edu.cn

Hong-Sheng Zhou
Virginia Commonwealth University
hszhou@vcu.edu

ABSTRACT

Cryptocurrencies have recently gained huge popularity. It is desirable to come up with effective approaches to constructing better blockchain protocols. In this paper, inspired by the 2-hop design by Duong et al (ePrint 2016/716), we put forth a generic paradigm for blockchain design, called n-hop blockchain. It includes one main chain, which is supported by $(n-1)$ supporting chains; hence, the main chain can achieve better security performance. In our paradigm, we show that our n-hop design can be easily extended to $(n+1)$-hop design. To demonstrate the power of our paradigm, we showcase two instantiations: *2-hop blockchain variant*, a combination of proof-of-stake and proof-of-work, and *3-hop blockchain variant*, which is extended from *2-hop blockchain variant* by adding Byzantine fault tolerance blockchain in 3rd hop.

CCS CONCEPTS

• **Cryptocurrency**; • **Blockchain**;

KEYWORDS

Cryptocurrency,Blockchain

ACM Reference Format:
Phuc Thai, Laurent Njilla, Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. 2018. A Generic Paradigm for Blockchain Design. In *EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '18), November 5–7, 2018, New York, NY, USA*, Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3286978.3286982

1 INTRODUCTION

Blockchain-based cryptocurrencies system like Bitcoin are increasingly popular and successful. The "core" is a global public distributed ledger, called *blockchain*, that records all transactions between *users* (stakeholders). Cryptocurrencies users (i.e., Bitcoin users) can create accounts to own bitcoins, and transactions enable a credit for one account and a corresponding debit to another account (i.e., transferring a certain number of digital currencies from a debit account to a credit account).

The Bitcoin-like systems are established by a chain of cryptographic puzzles (also called proofs-of-work (PoWs) [2, 7]), solved by a peer-to-peer network of nodes called *miners*. Each miner that successfully solves a cryptographic puzzle is allowed to extend the blockchain with a block of transactions, and at the same time to collect a reward. The chance of solving the puzzle first for each player is proportional to the amount of computing power the player invests.

To maintain the security of the Bitcoin blockchain, it requires that the majority of computing power is under the control of honest miners. Unfortunately, the majority of honest mining power assumption could be seriously challenged. Very recently, Duong et al [6] propose a very interesting design, called *2-hop blockchain*, via combining proof-of-work and proof-of-stake properly. The resulting blockchain protocol can be secure even if the adversary controls the majority of computing power. Also, Pass et al [25] introduce a hybrid consensus design, a combination of proof-of-work and Byzantine fault tolerance, which can be considered as 2-hop blockchain. That inspires us to design a generalized paradigm, called n-hop blockchain, which can achieve better performance than single blockchain. We will also showcase several of new instantiations to demonstrate the power of this paradigm.

1.1 Our Techniques

1.1.1 n-hop blockchain paradigm. Here, we present a n-hop paradigm for blockchain design, that is generalized from 2-hop block chain. In 2-hop blockchain protocol, proof-of-work miners in the *first* hop together maintain a proof-of-work blockchain which is then treated as a biased random beacon to elect a leader for the *second* hop in each round. That is, the first hop generates the information needed for the second hop, we therefore call the first hop support for the second hop. Naturally, we generalize this 2-hop design to an n-hop design. In this paradigm, the blocks in smaller hop may support the block in bigger hop; and $n-1$ hops, starting from the first hop to the $(n-1)$-th hop all together support for the last hop, i.e., n-th hop. We emphasize that these $(n-1)$ hops could be executed simultaneously.

1.1.2 Instantiations. Here, we introduce two instatiations of the paradigm.

2-hop blockchain variant. In 2-hop design in [6], a proof-of-work blockchain is utilized to support for a proof-of-stake blockchain.

*Approved for public release. Distribution unlimited. Case Number 88ABW-2018-4358 Dated 05 Sep 2018.

MobiQuitous '18, November 5–7, 2018, New York, NY, USA

ACM ISBN 978-1-4503-6093-7/18/11.
https://doi.org/10.1145/3286978.3286982

Here, our 2-hop blockchain variant is different in which a proof-of-stake blockchain support for a proof-of-work blockchain. Moreover, in our design, we choose the best chain in 2nd hop by number of supporting coin from 1st hop; the design in [6], on the other hand, choose the best chain in 2nd hop independently with the 1st hop. Our design also secure the system against the majority of malicious computing power with a different assumption.

Intuitively, we require honest stakeholders to utilize their resources (i.e., coins) to help select the best blockchain and utilize their coins to support their best chain. How can honest stakeholders utilize their coins to support their best chain? A possible way for an honest stakeholder to go is to include the stakeholder's best available blockchain as part of the transaction; this transaction is then signed by the stakeholder and bound to the stakeholer's current best available blockchain. When this transaction is broadcast into the network, all proof-of-work miners are informed with the stakeholer's best choice of blockchain. After collecting all stakeholders' best choices, miners will be able to derive and extend the best blockchain.

The next question is how to use coins to define the best chain. This is a tricky question. Recall that the adversary may control the majority of computing power; thus, choosing the most difficult chain as the best blockchain (as in Bitcoin) is not a good strategy anymore. We need to select the best blockchain based on a different strategy. As we discussed in the previous paragraph, each transaction is bound to a blockchain that is selected by a stakeholder, which allows us to define the balance of the output/credit account in the transaction as the "supporting coins" to endorse the blockchain. The immediate idea is to compare the supporting coins of two different blockchains, and then decide which one is better. However, this idea doesn't work directly: the adversary is able to create a blockchain with a huge amount of supporting coins by simply spending his coins repeatedly and frequently along the blockchain. We can avoid this issue by only counting the unspent coins which have not been transferred to any account in this blockchain. However, this strategy is still problematic, and we cannot count the unspent supporting coins from the beginning of the blockchain; note that the total number of supporting coins in each blockchain is the same since the coins are only moved from accounts to accounts. To address this issue, our final strategy is to *compare the unspent supporting coins in two branches from the block where the chains diverge (divergent position)*. The branch with more unspent supporting coins is defined as the better chain.

3-hop blockchain. Other than secure the system against the majority of malicious computing power, we here for the first time introduce a 3-hop protocol a secure combination of three chains to achieve a better performance for PoW-based blockchain. In this protocol, the first two hops are similar with the 2-hop protocol above that support for the third hop — a Byzantine fault-tolerant (BFT) blockchain [16]. A bit more concretely, the blocks in 1st hop sign to support the blocks in 2nd hop. Then, we can treat the second hop as a biased random resource to choose a small committees of registered participants, with have the voting power equal to the supporting coins from 1st hop, where each committee member one after the other generates a new block for the third hop. The resulting 3-hop blockchain can achieve significant improvement of PoW-based blockchains' throughput.

In this 3-hop blockchain, the best chain policy is almost the same as the Bitcoin blockchain. Here, we require the three blockchains are braided and have the similar structure making a chain-triple, and the best chain-triple is the one with the longest BFT chain. We note that, the underlying assumption of this 3-hop blockchain is similar to the 2-hop blockchain in [6] — the majority of collective resources is honest.

1.2 Related Work

Anonymous digital currency was introduced by Chaum [5].The first decentralized currency system, Bitcoin [20], was launched about 30 years later, by incentivizing a set of players to solve cryptographic puzzles (also called proofs-of-work puzzles [2, 7]).

It is very difficult to provide rigorous analysis for Bitcoin like protocols. Garay et al. [11] and then Pass et al [24] propose the first comprehensive *cryptographic* frameworks and show that, assuming the honest majority of computing power, Bitcoin achieves certain interesting security notions they defined. Their positive results fall apart if the adversaries' computational resources are not suitably bounded (strictly less than half). The blockchain security has also been investigated in rational setting (e.g.,[9, 10, 13, 21, 26, 27]).

The most closest to ours is the work by Duong et al [6], which formally extend the models of Garay et al [11]. and Pass et al. [24], and then propose a provably secure blockchain protocol against the majority of malicious computing power. We generalize their design paradigm and give two concrete instantiations: 2-hop blockchain variant and 3-hop blockchain. Please refer to Appendix B for the detail related works of these protocols.

2 OUR PARADIGM FOR BLOCKCHAIN DESIGN

In this section, we propose our generic paradigm for blockchain design. In this paradigm, we consider the last hop, i.e. the n-th hop, as the main hop in which all information (e.g. randomcoin, registered players) needed output from all blocks in the previous hops. We note that, to maintain the system, these hops should be linked properly so that the adversary cannot manipulate the structure of the whole blockchain. The blocks in a round are generated by the order of the hops. Let B_i^t be the t-th block of the i-th hop for $t > 0$. Other than a new block links to the previous block on the same blockchain, we require a secure n-hop blockchain is structured as follows.

- We require block B_n^t of the nth hop links to block $B_1^t, \ldots, B_{n-1}^t$. (block B_i^t is generated after B_{i-1}^t, for $1 \le i \le n$)
- We allow block B_i^t of the nth hop links to block $B_1^t, \ldots, B_{i-1}^t$.
- We require block B_1^{t+1} of the 1st hop links to the blocks $B_2^t, \ldots, B_n^t$
- We allow block B_i^{t+1} of the 1st hop links to the blocks $B_{i+1}^t, \ldots, B_n^t$

The blocks in the last hop are supported by the blocks in the first $n-1$ hop; hence, we can achieve better performance than a single blockchain, i.e. 1-hop blockchain.

We note that, in our design, the structure in a hop may not be a real blockchain. For example, in our 2-hop variant, in the first hop, users sign to support the transactions; there is no actual blockchain here.

We also note that, our design can easily extend n-hop blockchain to $n+1$-hop blockchain. We illustrate our blockchain design paradigm by a 3-hop blockchain structure (which is extended from our 2-hop blockchain) in Figure 1.

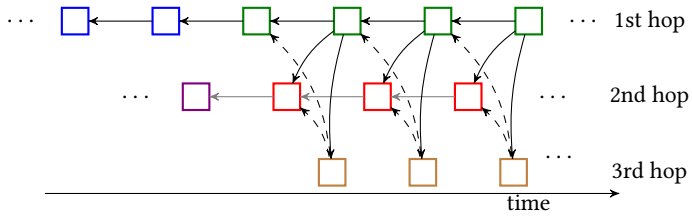

Figure 1: 3-hop blockchain structure

Here, solid arrows denote the first hops, gray arrows denote the second hops, and dot arrows denote the third hops. Green blocks denote the first hop's blocks, red blocks denote the second hop's blocks, and brown blocks denote the third hop's blocks. Note that the blue blocks are from the "mature blockchain", and violet blocks are from another "mature blockchain". The 2nd hop is extended from the 1st hop and the 3rd hop is extended from the 2nd hop.

3 2-HOP BLOCKCHAIN VARIANT

3.1 Our Model

The environment $\mathcal{Z}$ provides inputs for all miners and receives outputs from them. The environment may provide input to any miner in any round, and may modify the input from round to round. We denote by Input() the input tape, by Output() the output tape, and by Receive() the incoming communication tape, of each miner.

In our model, we specify two types of rounds that execute in turn: user-round and miner-round. In the user-round, the users are able to create accounts and sign transactions using their private keys. Moreover, they can utilize their coins to help the honest miners secure the system. On the other hand, in the miner-round, the miners solve PoW puzzles, and append a new block to the best chain on their local views. The users and miners are playing different roles in our model; however, without the collaboration of these two types of players, our model cannot be secure in the presence of the malicious majority computing power.

The user is another type of players in the system who does not solve the PoW puzzles to extend the blockchain, but spending coins in each round by signing transactions. A transaction is a statement to show that some coins are transferred from a source account to some destination accounts. A continuous transactions can be viewed as a chain of transactions. We only concern with the coins of destination accounts since the coins of the source account will become invalid once they are transferred. We assume that there is an initial blockchain, denoted $C^{\emptyset}$, which is distributed to all users before running the protocol. All of the legal coins must be traced back to $C^{\emptyset}$ by following the reverse order of the corresponding chain. Furthermore, a transaction must be verifiable and a coin cannot be transferred twice from the same source.

We consider the protocol executes round-by-round. We assume that the malicious users control at most $c_{\text{malicious}}$ coins, and in each round the honest users transfer $c_{\text{spend}} = \Theta(\kappa) \cdot c_{\text{malicious}}$ coins that originate from the blocks which were generated in $\Theta(\kappa)$ rounds earlier where κ is the security parameter. Note that c_{spend} denotes the *cash flow volatility* (v for short) of honest users. An honest user will sign his transaction to show his support to the bestchain on his local view.

We define a new comparison method for comparing two blockchains. In particular, the best chain is chosen based on the number of supporting coins. Given two distinct blockchains, the chain with more supporting coins from users after they are forked is better than the other one. We remark that once a coin is transferred, the supporting power is also transferred.

We refer to the above restrictions on the environment, the players, and the adversary as the (q, v)-synchronous setting [1] . The view of a player is denoted by $\text{VIEW}^{P,H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$, and the concatenation of all players' views is denoted by $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$. Similar to the framework by Garay et al [11], we will study security properties: *common-prefix* and *chain quality* of protocols Π in the (q, v)-synchronous setting. Such properties will be defined as predictions over the random variable $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$ by quantifying over all possible adversaries $\mathcal{A}$ and environments $\mathcal{Z}$.

3.2 Construction

We start by defining blocks, blockchains, proofs of work, a chain validation algorithm, and other related notations; we remark that many of the notations in this section are borrowed from [11]. Then, we define transactions in our protocol; here, new transaction format and properties are introduced. We then introduce new tools so that coins in a blockchain can be used to protect the blockchain. Finally, we present our main protocol.

Blocks. Let $G(\cdot)$ and $H(\cdot)$ be cryptographic hash functions with output in $\{0,1\}^{\kappa}$. A *block* B is a quadruple of the form $B = \langle s, X, ctr, w\rangle$ where $s \in \{0,1\}^{\kappa}$, $X \in \{0,1\}^{*}$, $ctr \in \mathbb{N}$, $w \in \{0,1\}^{\kappa}$, such that they satisfy the predicate $\text{validblock}^{D}_{q}(B)$, defined as $(H(ctr, w, G(s, X)) < D) \wedge (ctr \leq q)$ where the parameters $D \in \mathbb{N}$ and $q \in \mathbb{N}$. Here, D denotes the *difficulty level* of the block, and q denotes the maximum allowed number of hash queries in a round. We allow the size of q to be arbitrary as per backbone specification, as it bounds any player's ability to solve proofs of work in a round. Note that here, s should be the digest of the previous block, and w is a random nonce. Finally, we call X the *payload* of a block, and denote it as payload(B) which refers to the information stored in some block B. Specifically, in our protocol, we are interested in keeping *transactions* in X.

Blockchains. A *blockchain* C consists of a sequence of ℓ concatenated blocks $B_1\|B_2\|\cdots\|B_\ell$, where $\ell \geq 0$. For each blockchain, we specify several notations such as head, length, and payload as below: *blockchain head*, denoted head(C), refers to the topmost block B_ℓ

[1] For simplicity, we investigate the security of our designed protocols in the synchronous setting. We note that it is nontrivial to get a solution in the synchronous setting. Previously, [1, 12] provided solutions to (non-practical) consensus; the goal here is to get a blockchain *in the large-scale network*, against malicious 51%, which is much harder.

in blockchain C; *blockchain length*, denoted $\mathsf{len}(C)$, is the number of blocks in blockchain C, and here $\mathsf{len}(C) = \ell$; finally, *blockchain payload*, denoted $\mathsf{payload}(C)$, refers to the information we store in C, and we have $\mathsf{payload}(C) = ||_{i=1}^{\ell}\mathsf{payload}(B_i)$. For brevity, we use sub blockchain (or subchain) for referring to segment of a chain. For example, $C[1, \ell]$ refers to an entire blockchain, whereas $C[j, m]$, with $j \geq 1$ and $m \leq \ell$ would refer to a sub blockchain $B_j \| \cdots \| B_m$. We use $C[i]$ to denote the i-th block B_i in blockchain C. If blockchain C is a prefix of another blockchain C', we write $C \preceq C'$.

Please refer to Appenddix A for proof-of-work and chain validation of our protocol.

3.2.1 Transactions. Transactions are datagrams that are produced and consumed by the users. In a *transaction authentication scheme*, the users are able to create accounts and sign *transactions*.

Remark 3.1. *Our transaction authentication scheme is very similar to the digital signature scheme in Bitcoin except that, the transaction* $\tilde{\mathsf{tx}}$ *to be signed here includes an additional field B. This modification is critical in our protocol construction: there, we will use the additional field B to bind an unsigned transaction* $\tilde{\mathsf{tx}}$ *to a known block in a blockchain.*

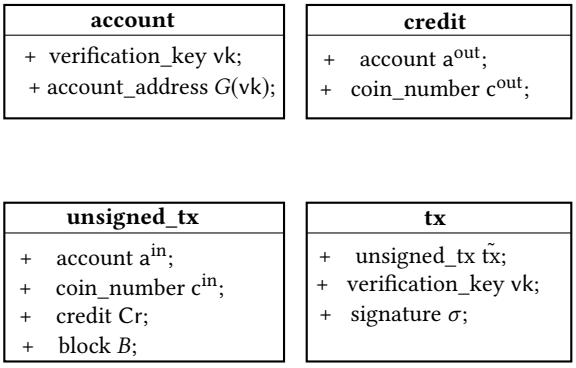

Figure 2: Data structure

Transaction format and data structure. We are now giving more details of our proposed data structures for account, credit, signed, and unsigned transactions in Figure 2. The account contains a verification key vk, and its address $G(\mathsf{vk})$. The credit stores an output account and the number of coins $\mathsf{c}^{\mathsf{out}}$; thus, $\mathsf{c}^{\mathsf{out}}$ coins will be transferred from an input account to $\mathsf{a}^{\mathsf{out}}$. The unsigned transaction includes a debited/input account and the number of debited coins c^{in} from the input account. Especially, this unsigned transaction also stores a field B to show its support to the corresponding block. The signed transaction consists of the corresponding unsigned transaction, a verification key, and the signature of this transaction.

In Figure 3, we present the format of a *regular transaction*. We employ the "dot" notation to indicate the access to an element of the data structure. For instance, if we want to access the block B in the data structure shown in Figure 3, we would refer to it as $\mathsf{tx}.\tilde{\mathsf{tx}}.B$. The signed transaction tx consists of three fields: a public key vk, an unsigned transaction $\tilde{\mathsf{tx}}$ for verification, and the signature σ of $\tilde{\mathsf{tx}}$. The unsigned transaction $\tilde{\mathsf{tx}}$ contains an input/debited account, the number of debited coins c^{in}, ℓ credited/output accounts, as well as the supported block B.

We remark that *coinbase transactions* are not signed and these unsigned transactions may be included in a blockchain via proof of work (as in Bitcoin).

We now define the transaction *wellformedness* in a blockchain. Intuitively, a well-formed transaction in a blockchain should be *verifiable* , i.e., verify predicate returns `True`), *traceable* and *not conflicting* . Before giving the detailed definitions for traceable transactions and non-conflicting transactions, we note that without loss of generality, the initial state of a blockchain C, denoted $C^{\emptyset}$, consists of multiple coinbase transactions.

Traceable transactions. Transactions represent the ownership of coins in cryptocurrency systems such as Bitcoin. In our system, transactions have a one-to-many relationship that represents the transfer of coin(s) from a single input account to potentially many output accounts. In order to make sure that a new transaction is valid, we need to ensure that the input account of that transaction has a valid balance (i.e., c^{in}). More precisely, the balance is from a previous transaction; therefore, we must trace back one step to the previous transaction input. We do this until a transaction can be traced back to a legal source, specifically, a coinbase transaction in $C^{\emptyset}$. In order to simplify our presentation, we assume that all accounts are unique and are used only once. We define a traceable transaction as follows.

Non-conflicting transactions. We define what it means for two transactions to be non-conflicting. We remark that, in our system, the debited accounts need to transfer all coins they have to credited accounts; therefore, those accounts cannot be debited accounts again in any other transactions. If there is a transactions in C such that its input account is the same as that in the considered transaction, then the examined transaction is a conflicting transaction. Otherwise, it is a non-conflicting transaction.

We are now ready to describe how to validate transactions in a blockchain. Valid transactions in a blockchain should be well-formed, i.e, the transactions should be verifiable, traceable, and non-conflicting. The validation of transactions is specified in algorithm validate_tx. The algorithm takes as input an initial chain $C^{\emptyset}$ and a chain C to be examined. If the chain C does not have the prefix $C^{\emptyset}$ or C is the same as $C^{\emptyset}$, then a bit b is set to `False`; otherwise, b = `True`. Next, if b is `True`, then every transaction in C will be checked by the three predicates: verify, trace, and nonconflict. If there exists a transaction that is not verifiable, traceable, or non-conflicting; then the algorithm outputs `False`; otherwise, it returns `True`. More details can be found in Algorithm 1.

Bound Transactions. As discussed in Section 1, each transaction in our protocol is bound to a certain block. We say that the transaction is bound to block B if its block section is set to be B. The definition of transaction-block binding is given as follows:

Definition 3.2 (Transaction-Block Binding). *Let* tx *be a transaction and* $\hat{B}$ *be a block. We say transaction* tx *is* bound *to block* $\hat{B}$ *if it holds that* $\hat{B} = \mathsf{tx}.\tilde{\mathsf{tx}}.B$.

Users make use of these bindings to show support for a certain blockchain. When a user signs a new transaction and broadcasts

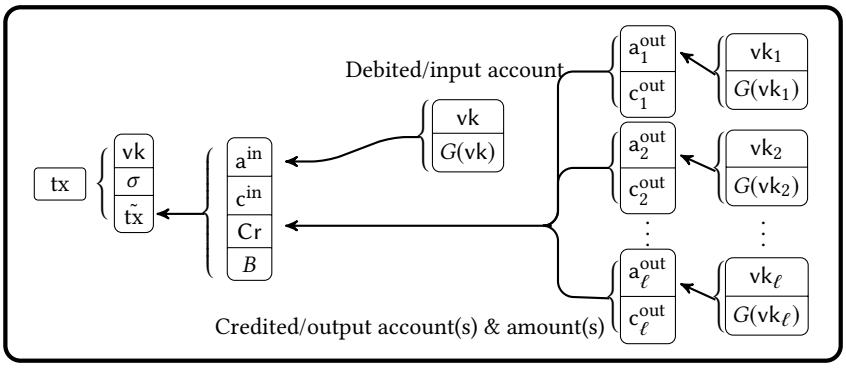

Figure 3: Transaction Data-structure Hierarchy, where $c^{in} = \sum_{i=1}^{\ell} c_i^{out}$.

Algorithm 1: The *transaction validation predicate*. The input is $(C, C^{\emptyset})$.

```
1  function validate_tx (C,C^∅)
2      b ← (C^∅ ≺ C)
3      if b = True then
4          ⟨s, X, ctr, w⟩ ← head (C)
5          s′ ← H(ctr, w, G(s, X))
6          repeat
7              ⟨s, X, ctr, w⟩ ← head (C)
8              {tx_1, tx_2, ..., tx_|X|} ← X
9              for i = 1 to |X| do
10                 if verify(tx_i) = False then
11                     b ← False
12                     break
13                 if trace(C, tx_i) = False then
14                     b ← False
15                     break
16                 if nonconflict(C, tx_i) = False then
17                     b ← False
18                     break
19             C ← C[1, len(C) − 1] ; /* Remove the head from
                   C */
20         until (C = C^∅) ⋁ (b = False);
21     return (b)
```

it to the miners, the support of this transaction is tracked in a transaction appendix of that chain. The transaction appendix is defined as follows.

Definition 3.3 (Transaction Appendix). *Consider a blockchain C and a set T of unique transactions where $T = \{tx_1, tx_2, \ldots, tx_{|T|}\}$ and $|T|$ is the number of transactions in T. We say T is a* transaction appendix *if it holds that tx_i is bound to block* head(C)*, for all $i \in [|T|]$. For convenience, we use $C.T$ to access the* transaction appendix *of C. If a subchain includes the* head(C) *of full chain, then it also includes the $C.T$, otherwise its T is ϵ.*

We now introduce two algorithms audit and virtualize that support our use of transactions and coins in the protocol. The audit is a blockchain auditing function (Algorithm 2, below). It appends each valid transaction in X to the transaction appendix of an input chain if and only if that transaction is bound to the head of the input chain. The second algorithm, called virtualize, invokes predicate audit over a set of chains, and updates the transaction appendix of each chain in the set; see Algorithm 3 for more details.

Algorithm 2: The audit function takes a chain C and transaction set $X = \{tx_1, tx_2, \ldots, tx_{|X|}\}$ as input.

```
1  Function audit (C, X)
2      B̂ ← head(C)
3      if X ≠ ∅ then
4          for i = 1 to |X| do
5              if TxAuth.verify(tx_i) = True ⋀ tx_i.t̃x.B = B̂ then
6                  if nonconflict(C, tx_i) = True ⋀ trace(C, tx_i) =
                       True then
7                      C.T ← C.T ⋃ {tx_i}
8      return
```

Algorithm 3: The virtualize function takes a set of chains $\mathbb{S} = \{C_1, C_2, \ldots, C_{|\mathbb{S}|}\}$ and a transaction set X as input.

```
1  Function virtualize (𝕊, X)
2      {C_1, ..., C_|𝕊|} ← 𝕊
3      for i = 1 to |𝕊| do
4          audit(C_i, X)
5      return
```

Transaction auditing. The transaction auditing function audit is parameterized by the nonconflict predicate, the trace predicate, and the TxAuth.verify function. This algorithm accepts a blockchain C and transaction set X as input. It serves to audit the chain C with the fresh transactions found in X that are bound to the head of C. In other words, it appends every transaction tx_i in X to the chain's transaction appendix $C.T$ if it satisfies that $tx_i.\tilde{tx}.B$ = head(C), tx_i does not conflict with C (nonconflict(C, tx_i) = True), tx_i is traceable on C (trace(C, tx_i) = True), and the authenticity of the signed transaction TxAuth.verify(tx_i) = True.

Virtual block extension. The virtualize is a chain extension algorithm that applies audit across a set of blockchains, denoted $\mathbb{S}$,

and updating each one in turn. The virtualize algorithm artificially extends each blockchain C_i stored in a set $\mathbb{S} = \{C_1, C_2, \ldots, C_{|\mathbb{S}|}\}$ for $1 \leq i \leq |\mathbb{S}|$ by generating or updating the transaction appendix $C_i.T$ of each chain C_i in $\mathbb{S}$ with the transactions from the input set X. It is parameterized by the previous algorithm audit. Each candidate chain C_i in $\mathbb{S}$ is audited with $\text{audit}(C_i, X)$. This updates each chain's appendix with any new binding transactions found in X.

Transactions carry balance from account to account and coins can be transferred between users' accounts. We now introduce *transaction value* which is the total amount of coins that have been transferred from the input account to the output accounts. Briefly, the balance of the input account being spent is the value of the entire transaction. This value must be equal to the sum of the balances of the output accounts. For the sake of simplicity in our presentation, we assume all accounts are unique.

Definition 3.4 (Transaction Value). *Let* $\text{value}(\text{tx})$ *be the transaction value for transaction* tx. *Then we have* $\text{value}(\text{tx}) = \sum_{k=1}^{\ell} \text{tx}.\tilde{\text{tx}}.\text{Cr}.\text{c}_k^{\text{out}}$ *where* ℓ *is the number of credited accounts in* tx *and* $\text{tx}.\tilde{\text{tx}}.\text{Cr}.\text{c}_k^{\text{out}}$ *refers to the amount of coins credited to account* a_k^{out} *for* $k \in [\ell]$.

We extend this definition to encompass the total value of a set of transactions. The naïve approach would be to sum the transaction value of each member of the set one-by-one. This would cause a serious problem with frequently or repetitively spent coins, which would be counted again and again. An adversary playing this strategy could significantly increase the value of his preferred set. Instead, we resolve this issue by only considering the coins that are not spent in any other transactions in the set. More precisely, if there are no input accounts in the other transactions that transfer some coins, then the coins will be contributed to the value of the considered set. The formal definition of value for a set of transactions is given by the following.

Definition 3.5 (Value for a Set of Transactions). *Let* $\text{value}(X)$ *be the value for a set* X *of transactions where* $X = \{\text{tx}_1, \text{tx}_2, \ldots, \text{tx}_m\}$ *and* $m \in \mathbb{N}$. *Then we have* $\text{value}(X) = \sum_{i=1}^{m} \sum_{k=1}^{\ell_i} (\text{tx}_i.\tilde{\text{tx}}.\text{Cr}.\text{c}_k^{\text{out}}) \cdot \phi_{ik}$ *where* ℓ_i *is the number of credited accounts in transaction* tx_i, *and* $\phi_{ik} = 1$ *if* $\text{tx}_i.\tilde{\text{tx}}.\text{Cr}.\text{a}_k^{\text{out}} \neq \text{tx}_{i^*}.\tilde{\text{tx}}.\text{a}^{\text{in}}$ $\forall i^* \in [m]$, *otherwise* $\phi_{ik} = 0$.

In our system, coins are moved from accounts to others and no new coins are generated in our system. This implies that the value of any blockchain is equal to the value of the initial blockchain $C^{\emptyset}$. Therefore, in order to compare two chains, we need to find the divergent position (e.g., j-th block) of the two examined chains, then compare the two subchains obtained by truncating all blocks from the initial block to the j-th block. In other words, we need to calculate and compare the values of the two branches instead of the entire chains. Again, note that we only consider coins that are not spent when determining the value of a subchain.

We remark that the value of a subchain is essentially based on the value of its payload which is a set of transactions. Therefore, for a subchain $C[j, m]$, $\text{value}(C[j, m]) = \text{value}(\text{payload}(C[j, m]))$. For simplicity, we use $\text{value}(C[j, m])$ to indicate the value of the subchain $C[j, m]$.

We further remark that, considering a subchain, coins spent over and over again are no more powerful than coins being spent once in that subchain.

Chain Comparison. With the definition of subchain value, we can compare the value of two chains as follows. Given two chains C_1 of length ℓ_1 and C_2 of length ℓ_2 with fork occurring after the j-th block. Specifically, $C_1[j+1]$ and $C_2[j+1]$ are not identical. We say the chain C_1 is better than C_2 if $\text{value}(C_1[j, \ell_1]) > \text{value}(C_2[j, \ell_2])$. If it happens that $\text{value}(C_1[j, \ell_1]) = \text{value}(C_2[j, \ell_2])$, we then say C_1 is better than C_2 if $H(\text{head}(C_1)) < H(\text{head}(C_2))$. We now provide a comparison algorithm, denoted compare, to implement this intuition.

The compare algorithm is introduced to compare blockchains (See Appendix C). It enables the honest players to deterministically select the best chain given any two chains. The bestvalid (Algorithm D) invokes the validate and compare functions over a set of chains, then returns the best valid blockchain.

At a high level, compare takes as input two chains C_1, C_2, and the divergent position j from their respective roots. We note that the algorithm uses a global public parameter $c_{\text{malicious}}$ which denotes the greatest number of coins controlled by the adversary, and a function max to find the maximum value of its two input values. The algorithm compare first computes the subchain values of C_1 and C_2 from the divergent position j. For the chain C_1, it computes the value of $C_1[j, \text{len}(C_1)]$, then adds the output of the max function on input $(0, \text{value}(C_1.T) - 1.1c_{\text{malicious}})$ where $C_1.T$ is the transaction appendix of C_1. Note that we need to subtract $1.1c_{\text{malicious}}$ coins from the appendix value in order to defend against the attack where the adversary adds his malicious coins to transaction appendix of his preferred chain. The computation of val_2 is analogous to val_1. The algorithm next compares the subchain values of each chain from the divergent position j, and then returns the chain with the greater value. In the event of a tie, the block hashes are used to make a decision by comparing the integer representation of both block's hash value.

Best valid chain. We further introduce the bestvalid algorithm that defines our strategy for selecting the best blockchain over a set of chains. The bestvalid algorithm is parametrized by the compare (Algorithm 8), validate_tx (Algorithm 1), validate_chain (Algorithm 7), and maxlen functions where maxlen takes as input a set of integers (lengths) then outputs the max length. The algorithm bestvalid takes as input a set of blockchains $\mathbb{S}$ and the initial blockchain $C^{\emptyset}$. It first finds the max length m which is the length of the longest chain, then creates an array T, and checks the validity of every chain in $\mathbb{S}$. Note that the boolean array T is used to mark the chains that either have not been compared or the best chain of any pair of chains. The bestvalid algorithm next scans every pair of chains in $\mathbb{S}$ to find the corresponding divergent position, then uses compare algorithm to compare the two considered chains from the divergent position. Once a better chain is determined, it then sets T[removed] = False where C_{removed} is the chain has less value so that the algorithm does not need to compare other chains with this chain later. The algorithm eventually finds the best chain in order from the newest to oldest forks. Once all chains are examined, the last C_{best} is the best chain among all chains in $\mathbb{S}$.

3.2.2 Our Main Protocol. As discussed in the introduction, we study a practical scenario where a cryptocurrency system has "grown up" such as the Bitcoin system, and then the adversary suddenly controls the majority of computing power. More precisely, the original system has already built a stable blockchain $C^{\emptyset}$, then the adversary's computing power increases and dominates the system. At this point, all users and miners start to switch to a new mode: our modified-backbone protocol.

The notations and algorithms described above lay the foundation for the modified-backbone protocol that we will describe in the following. Our modified-backbone protocol differs from that in [11], in that we rely on the miners *and the users*. Our protocol consists of two types of rounds: *miner-rounds* and *user-rounds*. For simplicity of presentation and without lost of generality, we consider an odd round as a miner-round, and an even round as a user-round; in a miner-round, only the miners are active; and in a user-round, only the users are active. All players have access to Broadcast that delivers messages to others in the next time step (round). Each player has an Input() tape and an Output() tape which are used to read from and write to the environment $\mathcal{Z}$, respectively. In our protocol, the players are granted (bounded) access to the random oracles $H(\cdot)$ and $G(\cdot)$. We now describe the details of our protocol.

Miners and users. The miners are the driving force of the protocol, who build blockchains; while the users are witnesses, who vote for blockchains by binding their transactions to blockchains in order to show their support. Clearly, the two roles have an interesting relationship. On one hand, we have the users who produce transactions with coins in the system; on the other hand, we have the miners who collect these transactions and include them on a blockchain.

Algorithm 4: Main Protocol: Miner.

$C \leftarrow C^{\emptyset}$; $\mathbb{S} \leftarrow \epsilon$; $round \leftarrow 0$;
while True **do**
  $\hat{\mathbb{S}} \leftarrow$ all chains found in Receive()
  $X \leftarrow$ all transactions found in Receive()
  while $\hat{\mathbb{S}} \neq \emptyset$ **do**
    $\hat{C} \leftarrow$ the first element in $\hat{\mathbb{S}}$
    $\hat{\mathbb{S}} \leftarrow \hat{\mathbb{S}} - \{\hat{C}\}$
    if validate_chain($\hat{C}$, $C^{\emptyset}$) $\bigwedge$ validate_tx($\hat{C}$, $C^{\emptyset}$) **then**
      $\mathbb{S} \leftarrow \mathbb{S} \cup \{\hat{C}\}$
  virtualize($\mathbb{S}$, X)
  $C \leftarrow$ bestvalid($\mathbb{S}$, $C^{\emptyset}$)
  $C_{\text{new}} \leftarrow$ pow($C.T$, C, $C^{\emptyset}$)
  if $C \neq C_{\text{new}}$ **then**
    $\mathbb{S} \leftarrow \mathbb{S} \cup \{C_{\text{new}}\}$
  $round \leftarrow round + 1$; Broadcast($C$);

Our protocol in miner-round is described in Algorithm 4. Here, several functions/predicates are used, including validate, virtualize, bestvalid, and pow. Now each player (i.e., miner) keeps a local storage $\hat{\mathbb{S}}$ to store a set of blockchains. Those blockchains are either received from the network via broadcast channels, or are mined by the miners themselves.

Likewise, our protocol in user-round is described in Algorithm 5. Here, function bestvalid and algorithms that related to transaction

Algorithm 5: Main Protocol: User.

$C \leftarrow C^{\emptyset}$; $round \leftarrow 0$; Account $\leftarrow \epsilon$; $command \leftarrow \epsilon$;
while True **do**
  $\hat{\mathbb{S}} \leftarrow$ any chain found in Receive()
  $C \leftarrow$ bestvalid($\{C\} \bigcup \hat{\mathbb{S}}$, $C^{\emptyset}$)
  $command \leftarrow$ Input()
  if *command* = *RegisterAccount* **then**
    (a, sk) $\leftarrow$ TxAuth.keygen(1^{κ})
    Account $\leftarrow$ {(a, sk)} $\bigcup$ Account
    Output(a)
  else if *command* = (*Credit*, *Argument*) **then**
    $(a^{in}, a^{out}, c^{in}) \leftarrow$ Argument
    if $(a^{in}$, sk) $\in$ Account **then**
      $\vec{tx} \leftarrow (a^{in}, c^{in}, (a_1^{out}, c_1^{out}), \ldots, (a_\ell^{out}, c_\ell^{out}), \text{head}(C))$
      tx $\leftarrow$ TxAuth.sign(sk, $\vec{tx}$)
  $round \leftarrow round + 1$;
  Broadcast(tx); Broadcast(C);

authentication scheme TxAuth are used. Now, each player (i.e, user) maintains a set of accounts.

In each miner-round or user-round, the players (both miners and users) carry out three major steps. In the first step, the miners and users first update their local views by reading the messages they have received. The miners and users update the chain set $\hat{\mathbb{S}}$ with any valid chain they find on the communication tapes Receive(). Let's consider a miner. If the miner finds a transaction tx in the Receive() tape, then it stores the transaction in a separate local storage called X. For every chain in $\hat{\mathbb{S}}$, the miner first checks whether the chain and all transactions on this chain are valid by using algorithms validate_chain (See Algorithm 7) and validate_tx (Algorithm 1). If a chain is not valid or there exists a transaction on this chain that is not valid, then the chain will be removed from $\hat{\mathbb{S}}$. The miner then has an updated set $\mathbb{S}$ where all invalid chains are removed. Further, the miner uses the virtualize algorithm to generate or update the transaction appendix T of each chain in $\mathbb{S}$ with the transactions from the input set X. After that, the miner runs bestvalid algorithm on input $(\mathbb{S}, C^{\emptyset})$ to find the best valid blockchain C. The computation in this step of each user is similar to that of the miner except that the user does not need to verify every chain in $\hat{\mathbb{S}}$ as well as run virtualize since he is only involved in the computation to support for his best chain. Moreover, the user will find the best chain among all chains in $\hat{\mathbb{S}}$ along with his local chain.

Each miner invokes the proof of work (See , Algorithm 6) with his best chain C, the transaction appendix $C.T$, and the initial chain $C^{\emptyset}$ as input in the second step. On the other hand, each user receives a command from the environment $\mathcal{Z}$ in its Input() tape; then executes the instruction. If the command is RegisterAccount, then the user only needs to create a distinct account in the system, and then outputs it to other participants. If the command is (Credit, Argument), the user will issue a new transaction. Argument is parsed to obtain accounts (a^{in}, a^{out}) and the number of coins c^{in} to be debited in the new transaction. The user next binds his transaction to the head of his best chain from the earlier step, and then signs this transaction.

Finally, the players (both miners and users) share their views with other players by broadcasting their local views to the network.

4 3-HOP BLOCKCHAIN

In this section, we present the basic idea of our 3-hop blockchain protocol. This is a combination of proof-of-stake, proof-of-work and Byzantine fault-tolerant (BFT), which is inspired from the Hybrid consensus [25], Byzncoin [14], Elastico [17]. Note that, those blockchains (Hybrid consensus, Byzncoin, and Elastico) are the special cases of the 2-hop blockchain design). Unlike these ideas; our 3-hop blockchain is bootstrapped from a PoS blockchain and a PoW blockchain with a different assumption — the majority of *collective* computing power.

We describe the procedure of each hop as follows.

(1) *First hop:* (corresponding to the proof-of-stake blockchain) the stakeholders are able to create accounts and sign transactions using their private keys and broadcast the transactions into network.
(2) *Second hop:* (corresponding to the proof-of-work blockchain) The miners take the transactions from the first hop as an input and try to solve the proof-of-work puzzle. Assume a honest miner have a local blockchain of size $size + \Delta$, where $size = \Theta(\Delta)$ denotes the size of the committee in the third hop and Δ is security parameter; the honest miner remove the last Δ blocks and call the miner of the first *size* block as the BFT committee. Due to the common prefix property, all the honest miner agree on the same BFT committee.
(3) *Third hop:* (corresponding to the Byzantine fault-tolerant blockchain) We use the committee, was selected from second hop. The voting power of a committee member relative with the number of supporting coin from the first hop of his PoW block . The players who produce these blocks become committee members and have the privilege to jointly generate new blocks for the BFT blockchain.

Similar to the 2-hop blockchain, chains in our 3-hop protocol are structured as chain-triples including proof-of-stake chain, proof-of-work chain and a BFT chain, and the best chain-triple is the one with the longest BFT chain.

We emphasize that, our protocol benefits significantly from the paradigm. The design is elegant and much simpler than the ones in [14, 17, 25]. In the full version, we will provide the complete 3-hop blockchain protocol and security analysis.

Acknowledgement. Phuc Thai and Tuyet Duong are in part supported by research gifts from Ergo Platform and IOHK Research. This work was done while Hong-Sheng Zhou visited Air Force Research Laboratory.

REFERENCES

[1] M. Andrychowicz and S. Dziembowski. PoW-based distributed cryptography with no trusted setup. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 379–399. Springer, Heidelberg, Aug. 2015.
[2] A. Back. Hashcash — A denial of service counter-measure. 2002. http://hashcash.org/papers/hashcash.pdf.
[3] I. Bentov, A. Gabizon, and A. Mizrahi. Currencies without proof of work. In *Bitcoin Workshop- Financial Cryptography and Data Security (FC)*, 2016.
[4] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld. Proof of activity: Extending Bitcoin's proof of work via proof of stake. Cryptology ePrint Archive, Report 2014/452, 2014. http://eprint.iacr.org/2014/452.
[5] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.
[6] T. Duong, L. Fan, and H.-S. Zhou. 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. In *Cryptology ePrint Archive, Report 2016/716*, 2016. https://eprint.iacr.org/2016/716.
[7] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, Aug. 1993.
[8] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, Aug. 2015.
[9] I. Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103. IEEE Computer Society Press, May 2015.
[10] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, Mar. 2014.
[11] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, Apr. 2015.
[12] J. Katz, A. Miller, and E. Shi. Pseudonymous broadcast and secure computation from cryptographic puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. http://eprint.iacr.org/2014/857.
[13] A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation (EC)*, pages 365–382, 2016.
[14] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, Austin, TX, 2016. USENIX Association.
[15] J. Kwon. Tendermint: Consensus without mining. 2014. https://tendermint.com/static/docs/tendermint.pdf.
[16] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.
[17] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. ACM.
[18] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490. IEEE Computer Society Press, May 2014.
[19] A. Miller, A. E. Kosba, J. Katz, and E. Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 680–691. ACM Press, Oct. 2015.
[20] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf.
[21] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. Cryptology ePrint Archive, Report 2015/796, 2015. http://eprint.iacr.org/2015/796.
[22] NXT Community. Nxt whitepaper. 2014. https://www.dropbox.com/s/cbuwrorf672c0yy/NxtWhitepaper_v122_rev4.pdf.
[23] S. Park, K. Pietrzak, A. Kwon, J. Alwen, G. Fuchsbauer, and P. Gaži. Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. http://eprint.iacr.org/2015/528.
[24] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, 2017. https://eprint.iacr.org/2016/454.
[25] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *Cryptology ePrint Archive, Report 2016/917*, 2016. http://eprint.iacr.org/2016/917.
[26] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In J. Grossklags and B. Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 515–532. Springer, Heidelberg, Feb. 2016.
[27] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden. Incentive compatibility of bitcoin mining pool reward functions. In J. Grossklags and B. Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 477–498. Springer, Heidelberg, Feb. 2016.
[28] P. Vasin. Blackcoin's proof-of-stake protocol v2. 2014. http://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf.

A PROOF-OF-WORK

Chain extension: Proof of work. The pow algorithm attempts to extend a chain via solving a proof of work (POW). This algorithm is parameterized by two hash functions $H(\cdot), G(\cdot)$ (which in our analysis will be modeled as random oracles); as well as two positive integers q and D where q represents the number of times

the algorithm is going to attempt to brute-force the hash function inequality that determines the POW instance, and D determines the "difficulty" of the POW. The algorithm works as follows. Given a chain C, an initial chain $C^{\emptyset}$, and a value X to be inserted in the chain C; the POW algorithm samples a random initial string w of length κ, it then hashes these values to obtain h with a counter ctr. Subsequently, it increments ctr and checks if $H(ctr, w, h) \leq D$; if a suitable (ctr, w, h) is found then the algorithm succeeds in solving the POW and extends chain C by one block. Note that the payload X, the initial string w, and the counter ctr will be inserted to the new block. If no suitable (ctr, w, h) is found, the algorithm simply returns the chain unaltered. We suppose that we have an initial blockchain $C^{\emptyset}$ as the starting point, and $C^{\emptyset}$ is known to all miners and users. We remark that, to ensure that each attempt is independent, we explicitly ask each miner to include a random string w in the proof of work.

Algorithm 6: The *proof of work* function, parameterized by q, D and the hash function $G(\cdot)$, $H(\cdot)$. The input is $(X, C, C^{\emptyset})$.

function pow $(X, C, C^{\emptyset})$
  if $C^{\emptyset} \preceq C$ **then**
    $\langle s', X', ctr', w' \rangle \leftarrow \mathsf{head}(C)$
    $s \leftarrow H(ctr', w', G(s', X'))$
    $ctr \leftarrow 1$
    $w \leftarrow \{0,1\}^{\kappa}$
    $B \leftarrow \epsilon$
    $h \leftarrow G(s, X)$
    while $ctr \leq q$ **do**
      if $H(ctr, w, h) < D$ **then**
        $B \leftarrow \langle s, X, ctr, w \rangle$
        break
      $ctr \leftarrow ctr + 1$
    $C \leftarrow C \| B$
    return C
  else
    return $C^{\emptyset}$

Chain validation. The next algorithm, called validate_chain, is introduced to verify the structure of an input chain C. This algorithm is nearly identical to the validate algorithm in [11] except that it only considers the chain that has a prefix $C^{\emptyset}$ and a random nonce w is introduced. Algorithm validate_chain is parameterized by values q and D, as well as access to the hash functions $H(\cdot)$ and $G(\cdot)$. It first checks whether the considered chain C has prefix $C^{\emptyset}$. If $C^{\emptyset}$ is the prefix of C, then starting from the head of C, for every block, the algorithm validate_chain ensures that the corresponding proof of work is properly solved where the counter ctr does not exceed q, a hash pointer s of the previous block is properly referenced, and the proof of work satisfies the difficult D. If all blocks verify is true then the chain is valid, otherwise it is rejected. Please refer to Algorithm 7 for more details.

Algorithm 7: The *chain validation predicate*, parameterized by q, D, and the hash functions $G(\cdot)$, $H(\cdot)$. The input is $(C, C^{\emptyset})$.

function validate_chain $(C, C^{\emptyset})$
  $b \leftarrow (C^{\emptyset} \prec C)$
  if $b =$ `True` **then**
    $\langle s, X, ctr, w \rangle \leftarrow \mathsf{head}\ (C)$
    $s' \leftarrow H(ctr, w, G(s, X))$
    repeat
      $\langle s, X, ctr, w \rangle \leftarrow \mathsf{head}\ (C)$
      if $\mathsf{validblock}_q^D(\langle s, X, ctr, w \rangle) \wedge H(ctr, w, G(s, X)) = s'$ **then**
        $s' \leftarrow s$; `/* Retain hash value */`
        $C \leftarrow C[1, \mathsf{len}(C) - 1]$; `/* Remove the head from C */`
      else
        $b \leftarrow$ `False`
    until $(C = C^{\emptyset}) \vee (b =$ `False`$)$;
  return b

B SUPPLEMENT FOR RELATED WORK

Pure proof-of-stake is a relatively inexpensive technique that is related to our 2-hop blockchain variant [3, 4, 15, 22, 28]. In [14, 17, 25], the proof of work and proof of stake can be combined together to improve the blockchain throughput.

On another front, attempts to limit a single entity from controlling too much computing resource is being made through complex, ASIC (Application Specific Integrated Circuits) resistant, and memory hard puzzles [2, 7] as an alternative to Bitcoin's SHA-256 proof-of-work puzzle. Other attempts aim to prevent pools from forming via non-outsource puzzles [19]. In addition, other important approaches [8, 18, 23] have been explored by the research community.

C THE CHAIN COMPARE FUNCTION

Algorithm 8: The chain compare function takes $\{C_1, C_2, j\}$ as input.

Function compare (C_1, C_2, j)
  $temp \leftarrow C_1$
  $val_1 \leftarrow$ value $(C_1[j, \mathsf{len}(C_1)]) +$ $\max(0, \mathsf{value}(C_1.T) - 1.1c_{\mathrm{malicious}})$
  $val_2 \leftarrow$ value $(C_2[j, \mathsf{len}(C_2)]) +$ $\max(0, \mathsf{value}(C_2.T) - 1.1c_{\mathrm{malicious}})$
  if $val_1 = val_2$ **then**
    if $H(\mathsf{head}(C_1)) > H(\mathsf{head}(C_2))$ **then**
      $temp \leftarrow C_2$
  else if $val_1 < val_2$ **then**
    $temp \leftarrow C_2$
  return $temp$

D THE BEST VALID FUNCTION

Algorithm 9: The bestvalid function takes $(\mathbb{S}, C^{\emptyset})$ as input.

Function bestvalid $(\mathbb{S}, C^{\emptyset})$
  Let m = maxlen(len(C_1), . . . , len($C_{|\mathbb{S}|}$))
  Create array T where T[i] = True for all $i \in [|\mathbb{S}|]$
  for $i = 1$ *to* $|\mathbb{S}|$ **do**
    if validate_chain(C_i, $C^{\emptyset}$) $\bigwedge$ validate_tx(C_i, $C^{\emptyset}$) **then**
      $\hat{\mathbb{S}} := \hat{\mathbb{S}} \bigcup C_i$
  $\{C_1, \ldots, C_{|\hat{\mathbb{S}}|}\} \leftarrow \hat{\mathbb{S}}$
  for $j = m$ *to* 1 **do**
    for $i = 1$ *to* $|\hat{\mathbb{S}}|$ **do**
      for $k = i + 1$ *to* $|\hat{\mathbb{S}}|$ **do**
        if $C_i[j] = C_k[j] \neq \perp \bigwedge$ T[i] = T[k] = True **then**
          $C_{\text{best}} \leftarrow$ compare(C_i, C_k, j) ;
           /* best, removed $\in \{i, k\}$ */
          T[removed] = False ; /* If best = i then
           removed = k, and vice versa */
  return C_{best}