Article

Modeling and Verifying the CKB Blockchain Consensus Protocol[†]

Meng Sun¹ , Yuteng Lu¹ , Yichun Feng¹ and Qi Zhang¹ and Shaoying Liu^{2,*}

¹ School of Mathematical Sciences, Peking University, Beijing 100871, China; sunm@pku.edu.cn (M.S.); luyuteng@pku.edu.cn (Y.L.); yichunfeng@pku.edu.cn (Y.F.); zhang.qi@pku.edu.cn (Q.Z.)

² Graduate School of Advanced Science and Engineering, Hiroshima University, Higashi Hiroshima City 739-8527, Japan

* Correspondence: sliu@hiroshima-u.ac.jp

[†] This paper is an extended version of our papers published in Proceedings of SEKE 2021, pp. 150–153, KSI Research Inc. and Knowledge Systems Institute, 2021, and Proceedings of BlockSys 2020, CCIS 1267, pp. 3–17, Springer, 2020.

Abstract: The Nervos CKB (Common Knowledge Base) is a public permissionless blockchain designed for the Nervos ecosystem. The CKB consensus protocol is the key protocol of the Nervos CKB, which improves the limit of the consensus's performance for Bitcoin. In this paper, we developed the formal model of the CKB consensus protocol using timed automata. Based on the model, we formally verified various important properties of the Nervos CKB to provide a sufficient trustworthiness assurance. Especially, the security of the Nervos CKB against the selfish mining attacks to the protocol was investigated.

Keywords: Nervos CKB; consensus protocol; model checking; UPPAAL

Citation: Sun, M.; Lu, Y.; Feng, Y.; Zhang, Q.; Liu, S. Modeling and Verifying the CKB Blockchain Consensus Protocol. *Mathematics* **2021**, *9*, 2954. <https://doi.org/10.3390/math9222954>

Academic Editor: Frank Werner

Received: 25 October 2021

Accepted: 12 November 2021

Published: 19 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Blockchains are distributed digital ledgers for which there are numerous benefits such as decentralization, persistency, and anonymity. A continuously growing ledger of transactions being represented as a chain of blocks is provided in a blockchain, where the transactions are distributed and maintained over a peer-to-peer network [1]. Blockchain has become a popular technology since it was first proposed by Satoshi Nakamoto in 2008 to support Bitcoin [2] and has been successfully applied in many scenarios due to its power to create, transfer, and own assets in crypto-economy networks. Ethereum [3] extends the application range of blockchain and allows developers to write smart contracts and create different decentralized applications. Both Bitcoin and Ethereum have shown their exciting potential for building a powerful crypto-economy network and have attracted much attention from governments and industry.

Developing a secure and trustworthy blockchain is highly challenging because of the vulnerabilities and the complexity of the distributed execution environment. In addition to the security issues, the processing speed is also an important concern. However, both Bitcoin and Ethereum have a limit of processing large transactions per time unit. In other words, their processing capability is severely limited by the scalability. To alleviate this problem for long-term sustainability, the Nervos team proposed the Common Knowledge Base (CKB) [4], which uses a decentralized and secure layer and provides common knowledge for the peer-to-peer network.

Since the CKB has become the trust root of the decentralized secure crypto-economy system, guaranteeing the security and consistency of the CKB consensus protocol have become very important. In fact, there are some protocols in which vulnerabilities were discovered after they have been taken as correct and used for a long time [5]. In the literature, there are some existing works for formal modeling and verification of blockchain systems.

For example, the work in [6] proposed a novel approach for verifying the properties of Ethereum smart contracts using statistical model checking, and a formal model of the Bitcoin protocol was proposed and verified with the UPPAAL model checker [7,8] in [9].

The CKB *consensus protocol* [10] is the key protocol being used in the CKB to build the secure and optimal crypto-economy system [4]. The protocol aims to overcome the two drawbacks of Bitcoin consensus: the *low transaction processing throughput* and the *vulnerability to selfish mining attacks*. It limits the time of connecting the sender in the search of a lost transaction. Such a restriction improves the efficiency of transaction processing without compromising the security of the blockchain. Furthermore, the protocol adopts a novel “*two-step confirmation*”, which can be used for selfish mining attack resistance.

Since the CKB is becoming more and more popular and its applications are constantly increasing, the security properties of the CKB should have more attention paid to them. The security of the CKB calls for a detailed investigation, and its ability to resist selfish mining attacks has not been formally checked. In this paper, we propose the formal model of the CKB consensus protocol using timed automata. Based on the formal model, we verified the corresponding important properties with mathematical rigor with UPPAAL, which is a model checker that has been successfully used in various case studies [9,11,12]. Model checking [13] is a formal method of verification, which requires mathematical formalisms for both the desired properties and systems and assures system correctness w.r.t. the properties specified in given specifications automatically. Meanwhile, model checking is also helpful for finding and fixing bugs in the system implementation.

The work in this paper is an extension of our previous studies [14,15], where we initially discussed the formal models of the CKB block synchronization protocol and consensus protocol, respectively, and the verification of some important properties of these two protocols. In this paper, we further improved the formal models of the CKB consensus protocol and investigated its robustness against malicious attacks, especially selfish mining attacks. The main contributions of this paper are as follows:

- The formal models of the CKB consensus protocol in timed automata are proposed;
- A family of properties is formally defined and verified in UPPAAL;
- The ability of the CKB to resist malicious selfish mining attacks is proven.

The rest of this paper is organized as follows. The Nervos CKB and CKB consensus protocol are briefly described in Section 2. Section 3 presents the formal model of the CKB consensus protocol. Then, a family of properties of the protocol is formally defined and verified using UPPAAL in Section 4. Section 5 discusses the ability of the CKB to resist selfish mining attacks. Related work is provided in Section 6. Finally, Section 7 concludes the paper and discusses possible future studies.

2. Preliminaries

This section gives an introduction to the Nervos CKB, the CKB consensus protocol, and attacks.

2.1. The Nervos Network

The CKB is an open, public, and PoW-based blockchain, which was proposed in the Nervos Network [16]. It was inspired by Bitcoin, but provides higher scalability and lower transaction costs compared to Bitcoin. There are mainly two ways to improve the scalability of blockchain: increasing the block space to store more transactions and moving part of the operations off-chain for execution. The Nervos Network [16] uses the second approach and creates a two-layer environment. Figure 1 shows its layered architecture, which separates the state and computation and provides better scalability and more flexibility to each layer.

Figure 1. The CKB layered architecture.

The CKB layer, designed as a public permissionless blockchain for a layered crypto-economy network, is the first layer in the Nervos Network. It is responsible for providing the decentralized and secure infrastructure. In addition, it also includes the operation of state verification. In order to settle the assets that come in and out of the second layer, the CKB layer ensures the decentralization and sustainability of the entire blockchain. The second layer is the environment of generating transactions and calculating and is mainly responsible for generating states and protecting privacy. For different needs, it can be designed separately to match various scenarios. The encryption of the CKB layer protects the activities in the second layer. The second layer's operation can be expanded to a large extent under the security provided by the CKB layer.

Applications on the second layer can choose the proper generation methods based on their particular needs. The CKB layer provides common knowledge custody for the crypto-economy network, and its design target focuses on states. Common knowledge refers to states verified by global consensus, and crypto-assets are examples of common knowledge. CKB can generate trust and extend this trust to the second layer, making the whole network trustworthy.

The operations of the Nervos Network consist of three parts: state generation executed off-chain, the state-verification-based CKB virtual machine, and storing the states in the cell. Once a new state is generated by the second layer, it will be placed into the transaction. Then, the transaction will be broadcast to the whole network. To overcome the shortcomings of Bitcoin and Ethereum, as mentioned above, the CKB consensus protocol increases the output and enhances the security. The two-step confirmation is used for transaction verification where the two steps are defined as the proposal step and the commitment step. All transactions must go through the two-step confirmation. In the Nervos Network, users can participate in activities as three types of nodes: The *mining nodes*, which are responsible for collecting transactions and generating blocks, the *full nodes*, which responsible for the verification, and the *light nodes*, which only focus on the data they need and use the least resources. All nodes can freely enter or exit the blockchain.

2.2. CKB Consensus Protocol

The CKB consensus protocol is a variant of Nakamoto Consensus (NC) and complies with the PoW mechanism. While retaining the advantage of NC, the CKB consensus protocol improves the performance limit and resistance to selfish mining attacks by adopting a two-step confirmation, as shown in Figure 2. The block structures in the CKB include the proposal zone and the commitment zone [4,10]. When a blockchain user wants to record a transaction on the blockchain, a new transaction is generated. Based on the design of the CKB, miners put these new transactions into the proposal zone of a block. The proposal step starts once the proposal zone receives the transactions. This step will mainly go through two operations. The first one is to check *txpid*, which is defined as the first few

bits of the transaction ID. In the second operation, full nodes confirm whether they have received the transaction and then verify it. When a transaction passes the above operations, it is considered to be “proposed”. Next, the commitment step starts once the transaction is put into the commitment zone by the miner. In this step, full nodes confirm that the transaction is not a duplicate and it would not conflict with previous transactions. It is assumed that the transaction’s *txpid* appears in the proposal zone of one block and the commitment zone of another block, then full nodes confirm that the distance between these two blocks on the chain is kept within a predefined range. The transaction is “committed” after the commitment step is complete.

Figure 2. Two-step confirmation.

The block propagation mechanism adopted in the consensus protocol checks whether the transaction in a block is lost while avoiding extra round trips. In selfish mining attacks, some transactions are concealed by the malicious miners. If such missing transactions are continuously requested, an extra round trip will occur. The block propagation mechanism regulates the maximum number of steps for the round trip through the following two operations. In the first operation, when a committed transaction is previously unknown, its sending node will be requested. There exist some transactions that are indeed proposed, but they are not broadcast. The sending node must provide these transactions and put them in the prefilled transaction list. If the sending node and the receiving node have the same list that contains these nonbroadcast transactions, these transactions can be considered valid. In the second operation, if a transaction is still missing, the sending node will be queried again. When the sending node does not provide this transaction within the time limit, this node will be included in the blacklist. Just as Bitcoin consensus, the CKB cannot resist majority attack (51% attack) either.

2.3. Selfish Mining Attack

Some blockchain systems including Bitcoin have suffered from the selfish mining attack. In the worst case, the malicious miners can occupy the dominant position in the mining, and the decentralized characteristics may disappear. Then, the original advantages of the blockchain no longer exist.

The way to gain illegal benefits in a selfish mining attack is that the malicious miners create nonpublic blocks and use these blocks to replace the blocks created by the honest miners. When one malicious miner generates a new block and launches a selfish mining attack, he/she hides the block and waits for the opportunity to announce it. In general, multiple malicious miners join together to form a malicious group and share the computing power to improve the probability of success. The more blocks these malicious miners possess, the higher the profits they can obtain. Meanwhile, the other honest miners are

competing with the malicious group for mining. When the computing power of honest miners far exceeds that of the malicious group, it will be difficult for the malicious group to gain an advantage to obtain benefits.

The key idea of the selfish mining attack is to create a secret branch from the chain, and the miners in the malicious group will work only on this private chain. When competing with the honest miners for mining, the malicious group waits until the private chain is longer than the public chain. By the time the private chain gains the upper hand, the malicious group announces it to the public, causing a fork. Since the newly announced chain is longer than the original chain, other miners choose the longer one to follow. Furthermore, the blocks added thereafter are successors of this private chain. Therefore, the private chain replaces the original one as the main blockchain. Since the original chain is abandoned, all the mining rewards of honest miners go down the drain. The malicious group is more profitable when the newly announced private chain becomes longer.

3. The Formal Model of the CKB Consensus Protocol

In this section, we propose the formal model of the consensus protocol using timed automata. To accurately simulate the operations of transaction verification in the CKB consensus protocol, this model formalizes both the verification process and the interactions among different nodes. Such a model consists of four automata: *two-step*, *miner*, *full node*, and *block-propagation*.

All the variables in the model are used to specify whether the operations are successful or not. The default initial values of the variables are all 0. Once the operations are complete, the corresponding values are assigned to the variables according to the results. The assigned value is 1 if the operation is successfully completed and greater than 1 if the operation is abnormal. The assigned variables are taken as parameters in the guard conditions on transitions.

3.1. Two-Step Automaton

The two steps in the two-step confirmation are “*proposal*” and “*commitment*”, respectively. All the transactions that pass the two-step confirmation are taken as legal. After a node generates a new transaction, a miner collects the transaction and completes the PoW to generate a new block. The transaction is firstly written in the proposal zone of one block, and then, the proposal step begins.

The initial state of the two-step automaton in Figure 3 is T_0 , which represents the generation of a new transaction. The channel *collectP!* simulates the operation of mining and is used to synchronize with *collectP?*, which is a channel in the miner automaton. The variable c denotes the global time, which represents the time interval of each mining epoch. Variable cp is used to specify whether the transaction has been put into the proposal zone. According to the CKB consensus protocol, the difficulty of the PoW and the time interval will be adjusted to make full use of the hardware performance, maintaining high-efficiency production. Although the time interval in the protocol is not constant, setting time c to a fixed value in this model does not affect the simulation of verification and propagation.

The function of the proposal zone is to announce new transactions being processed by a miner to all nodes. Transactions that have not yet passed subsequent verification are not considered to be valid. Therefore, these transactions in the proposal zone do not affect the legality and spreadability of the blocks. The state T_1 captures “start of proposal confirmation”. There are 4 operations in the proposal confirmation: (1) to confirm that a transaction exists in the proposal zone; (2) to check the *txpid* of the transaction; (3) to confirm that the transaction has been received by the full node; (4) to verify the transaction content. The value of variable *checkT* is used to specify whether the transaction successfully passes the *txpid* check. The value of *checkT* is zero by default before the check, and a forced state transition will be made by the invariant.

Figure 3. The two-step automaton.

In the CKB consensus protocol, *txid* checking is performed by the full nodes, so the channel *checkTxid!* is used to synchronize with the channel *checkTxid?* in the full node automaton. The full node automaton assigns the checking result to *checkT*. The value of variable *x* denotes the corresponding block height on the blockchain. Whenever a new block is added into the blockchain, the value of *x* increases by 1. The height of the block in which the transaction exists is recorded using the value of variable *hp*.

The process continues to move forward if all operations in the two-step confirmation process are successful. If any verification fails, the state transfers to *T9*, indicating that it is impossible to broadcast the transaction. *T2* is the state for “verification of transaction”. The channel *ReceiveVerify!* simulates the verification performed by a full node. Once the verification is finished, the full node automaton assigns value 1 to the variables *checkR* and *checkV*. The values of these two variables are used to indicate whether the transaction is successfully received and verified by the full node, respectively. *T3* is the state in which the transaction is ready for “mining of the second step”. Variable *cc* marks whether the transaction is put into the commitment zone. A transaction τ that has been verified in the first confirmation step is regarded as a “proposed transaction”. If τ is in the proposal zone of a block with height *hp*, we say that τ is proposed at height *hp*.

Miners can collect all the transactions that have completed the first confirmation step and write them into a new block’s commitment zone. The channel *collectC!* synchronizes with the channel *collectC?* in the miner automaton to simulate the mining behavior. The mining operations in the two steps are different in the locations in which the miners write the transaction. There are two blocks in which the transaction exists, and the height interval between these two blocks is limited in a previously defined range.

T4 is the state for “start of the commitment confirmation”. It is reached once the transaction has been denoted as proposed and put in the commitment zone. The value of variable *checkC* shows whether this transaction conflicts with other transactions on the chain. Channel *committed!* synchronizes with channel *committed?* in the full node automaton and simulates the confirmation of the proposed transaction. A proposed transaction τ must meet the constraint $cc \geq 1$ once it enters the confirmation. This constraint means that τ has been put into the commitment zone. The current height of this block on the chain is captured by the variable *hc*.

T5 is the state that conforms to the invariant $close \leq hc - hp \leq far$. The transaction appears in the proposal zone and the commitment zone of two different blocks. The time spent in the two-step confirmation process creates a difference in *hc* and *hp*. The values of the constants *close* and *far* are predefined according to the efficiency of the hardware equipment. The height interval between the two blocks can be regarded as the time

required for the first step of confirmation. The setting of *close* is to ensure the time interval is long enough for the transaction to be propagated to the entire network. Each node has limited memory space in the local device, and the value of *far* is decided on the basis of the number of proposed transactions that can be stored in its device.

The state transfers to *T6* once the constraint $checkC == 1$ is satisfied, while the channel *propagating!* is triggered simultaneously. All transactions that reach this state are regarded as “committed transactions”. In the two-step confirmation process, if any of the variable values in *checkT*, *checkR*, *checkV*, and *checkC* is greater than 1, the verification is a failure. Transactions that fail in verification directly go to *T9*, which is defined as “transaction invalid”. Invalid transactions do not undergo subsequent verification steps.

The channel *propagate!* synchronizes with *propagate?* in the block-propagation automaton. If there is a transaction τ in the commitment zone of a certain block that is either proposed or committed, then τ can be spread to the network. If a transaction is missing, the block-propagation automaton initiates contact and requests the missing part from the miner automaton. The miner should respond within a short time. Otherwise, he/she is disconnected and blacklisted.

T7 and *T8* are the states that indicate “authorization of broadcast” and “prohibition of broadcast”, respectively. The value of variable *p* denotes whether the transaction is propagable. If the transaction is legal and can be propagated, then $p == 1$. Otherwise, the value 2 indicates that the transaction cannot be propagated. When the state reaches *T7*, *T8*, or *T10*, it means the end of the transaction verification. When the next transaction is born, the automaton state returns to *T0*, and the global time and variables are reset.

3.2. Full Node Automaton

Once a new block is generated, the legitimacy and the PoW of blocks are checked by the full node before they are broadcast. Since the two-step confirmation is transaction-oriented rather than block-oriented for the verification process, the full node automaton is also transaction-oriented. In this automaton, all operations aim at a single transaction.

Figure 4 (The state marked *c* is committed. A state is committed if any of the locations in the state are committed. A committed state cannot delay, and the next transition must involve an outgoing edge of at least one of the committed locations.) depicts the full node automaton. In the first confirmation step, the full node performs the checking of transaction *txid* and the verification of contents, which are captured by the channels *checkTxid?* and *ReceiveVerify?*, respectively. State *F1* is “checking of *txid*”, and the variable *checkT* is the result. States *F2* and *F3* correspond to “confirmation of receiving” and “transaction verification”, respectively. The variables *checkR* and *checkV* are used to note the results of these two operations.

When the full node reaches the second confirmation step, it becomes responsible for committing the transaction. Once the channel *committed?* synchronizes with the channel *committed!* in the two-step automaton, the state *F4* is reached. The assignment of the variable *checkC* marks whether the operation is successful. The state moves to *F5* once any operation fails. In this case, the transaction becomes invalid.

3.3. Miner Automaton

A miner’s behavior is specified in Figure 5, where *M1* captures the standby state. Once new transactions are generated, miners package these transactions and generate new blocks through the PoW. This automaton simulates the behavior of honest miners, so the mining results are all public. State *M2* means “new block generation”, which is reached after mining.

Figure 4. The full node automaton.

Figure 5. The miner automaton.

The automaton synchronizes with channel *connecting!* in the block-propagation automaton through the channel *connecting?* if a transaction is missing. Then, the state transfers to *M3*, which represents “the contact with the miner”. Channel *requesting?* describes the process in which the miner is asked for the missing transaction. After that, the state transfers to *M4*, which stands for “response to the request”, and the miner sends the requested transaction back. The assignment *checkRe := cc* uses the operation result after the transaction is written in the commitment zone as the miner’s reply.

When the transaction is still missing, the inquiry will be launched again. Channel *querying!* in the block-propagation automaton synchronizes with *querying?* in the miner automaton. State *M5* means “reply to query”. Variable *checkQ* represents the answer of the miner. Similarly, the value of *checkQ* is assigned to *cc*. The miner is taken as suspicious and blacklisted after two failed requests for the transaction. The channel *disconnecting?* is used to simulate this operation, which synchronizes with the channel *disconnecting!* in the block-propagation automaton and transfers to state *M6* for “disconnection”.

3.4. Block-Propagation Mechanism

The process of the block-propagation mechanism is described in Figure 6, which starts from the standby state *P0* by synchronizing the channel *propagating?* with the channel *propagating!* in the two-step automaton. State *P1* checks if the transaction is in the commitment zone or not. The value of variable *p* indicates whether the transaction can be propagated. The transaction can be broadcast if *p := 1*, and the broadcasting is forbidden if *p := 2*. If the value of *cc* is different from 1 ($cc \geq 2$), the transaction is not in the commitment zone of any public block, and the channel *connecting!* should be activated to synchronize with channel *connecting?* in the miner.

Figure 6. The block-propagation automaton.

State $P2$ means that the transaction is previously unknown. If $checkRe \geq 2$, which means that the transaction is still not acquired, the state transfers to $P3$ for “failure in request”. The channel *querying!* is used to synchronize with the channel *querying?* in the miner, which must reply in a short duration ($t < 3$ in Figure 6). State $P4$, which means “transaction invalidation”, is reached if the missing part is still unknown. Then, the miner is blacklisted and disconnected. Such an operation is simulated by the transition labeled by the channel *disconnecting!* and an assignment $p := 2$, which tells the two-step automaton that this transaction should not be propagated. This transition leads the automaton back to the initial standby state.

4. Verification of the CKB Consensus Protocol in UPPAAL

We conducted a series of experiments to explore the credibility and consistency of the consensus protocol by formalizing and verifying its key properties. In this section, we did not consider properties related to the malicious attackers. The presence of malicious attackers will be discussed in the next section. In the following, we define a family of properties that should be satisfied in the CKB consensus protocol. Based on the proposed formal model, we conducted some experiments using the UPPAAL model checker to check the correctness and consistency of the protocol.

First of all, based on the design of the CKB consensus protocol, newly generated transactions must go through a process of being put into the proposal zone. We define this process as $P1$, in which $T1$ represents that the transaction is in the proposal zone. Subsequently, the information of such transactions will be received by other nodes, and the legality of the blocks and the propagation will not be affected by the validity of the transactions. The verification result in UPPAAL demonstrates that the protocol satisfies (1).

$$A \langle \rangle TwoStep.T1 \quad (1)$$

Only after a transaction successfully passes the *txid* check, it can be considered as proposed. Therefore, transactions that have not completed the *txid* check are not “proposed transactions”. This property is formalized as (2), and the verification result shows that the protocol satisfies (2) as well.

$$A [] TwoStep.T4 \text{ imply } (checkT == 1)$$

$$A [] \text{ not } checkT == 1 \text{ imply not } TwoStep.T4 \quad (2)$$

$P3$ formalizes the following property: the full node should receive and verify a transaction before it is proposed. On the other hand, the transaction cannot be considered proposed if the full node has not received the transaction or completed the verification of its content. In the proposal step, the transaction *txid* is processed first, and then, a notification is sent to the full nodes. As mentioned earlier, the transaction cannot be

considered proposed until it passes the check $txid$ ($checkT == 1$). Once the check fails, the transaction will never be considered as proposed. Furthermore, the transaction must have been received ($checkR == 1$) and verified ($checkV == 1$) by the full nodes. The state $T4$ in the two-step automaton indicates that the transaction is proposed. The verification in UPPAAL shows that (3) is satisfied.

$$A [] TwoStep.T4 \text{ imply } (checkR == 1 \text{ and } checkV == 1)$$

$$A [] (not\ checkR == 1) \text{ or } (not\ checkV == 1) \text{ imply not } TwoStep.T4 \quad (3)$$

Before the transaction is put into the commitment zone, it must have been received and verified by the full node. A transaction that has not been received or verified by the full node cannot appear in the commitment area. We formalize this property as (4). The state $T5$ means that the transaction is put in the commitment zone. In fact, the second step of the two-step confirmation will be activated if and only if the miner finishes placing the transaction in the commitment area. Only through the verification of the proposal step, the transaction will be put into the commitment zone. (4) is satisfied based on the verification result.

$$A [] TwoStep.T5 \text{ imply } checkR == 1 \text{ and } checkV == 1$$

$$A [] not\ (checkR == 1 \text{ and } checkV == 1) \text{ imply not } TwoStep.T5 \quad (4)$$

A transaction must be located in the commitment zone with height hc and satisfy the condition: $close \leq hc - hp \leq far$ when it is committed. Such a property is formalized as (5), in which $T6$ means commitment of the transaction. The value of $checkC$ is used to indicate whether the transaction is in the commitment area. This property is satisfied according to the verification in UPPAAL.

$$A [] TwoStep.T6 \text{ imply } checkC == 1$$

$$\text{and } (close \leq hc - hp \text{ and } hc - hp \leq far) \quad (5)$$

If a transaction is missing and cannot be obtained by the miner after the requesting and querying operations, the miner will be blacklisted and disconnected. This property is formalized as (6).

$$A [] BlockPropagation.P3 \text{ and } BlockPropagation.P4 \text{ imply MiningNode.M6} \quad (6)$$

The model is repeatable, and there is no deadlock, formalized as (7).

$$A [] not\ deadlock \quad (7)$$

Both (6) and (7) are satisfied based on the verification.

5. Consistency and Robustness Analysis with Attacks

In reality, malicious attacks are always inevitable. In this section, we added attacks to our models and checked the security properties of the protocol.

The security of CKB consensus protocol against selfish mining attacks is discussed in this subsection. In the attack scenario, the other automaton models remain the same, but the miner's behavior is different. The malicious miner deliberately hides a block when generating it. We verified whether the protocol can defend against selfish mining attacks. The security properties are specified in the CTL formula and were proven in UPPAAL.

Figure 7 offers an automaton that simulates the behavior of a malicious miner. Compared to the honest miner in Figure 5, this automaton has an additional state $M7$, while the remaining states and transitions stay unchanged. When the malicious miners collect proposed transactions and put them into the commitment zone of the new block, the state

transfers to $M7$, which is defined as “attack start”. When $M7$ transfers to state $M2$, which is “new block generation”, the automaton synchronizes channel *attack!* with *attack?* in the SelfishMining automaton to perform the attack. According to the result of the attack, the selfish mining automaton returns the parameter cc , and the two-step automaton decides whether the transition should be fired based on the value of cc . The condition $cc > 0$ means that the transaction has been collected by the miner and put into the block.

Figure 7. The malicious miner automaton.

The mining competition between the malicious group and the honest miners can be described as the following three scenarios. In the first scenario, the malicious group leads the honest miners and generates blocks more quickly. As a result, the private chain has an absolute advantage. If the length of the private chain is already longer than the public chain by two, the malicious group can choose to announce the private chain immediately. At the moment, the public chain is shorter, so it will be discarded. The malicious group can also choose not to publish the private chain and continue mining. When the length of the public chain is about to catch up with the length of the private chain, that is to say, the gap between the two chains is only one, the malicious group will announce the private chain. In the second scenario, honest miners take the lead to find the new block and put it in the public chain. Once length of the private chain lags behind the length of the public one, the malicious group will directly abandon the private chain. In the third scenario, the malicious group has the same computing power as the honest miners, namely, the honest miners and the malicious group would find blocks at the same time. There is no advantage in the private chain. At this time, the malicious group could announce the private chain, and then, the full nodes would choose the public chain or private chain to follow.

The malicious group could also continue to bet until the game is over. In the first scenario, the selfish mining attack succeeds, and the malicious group will receive rewards for all blocks on the private chain. In the second scenario, the attack fails, and the malicious group receives nothing. In the last scenario, if a subsequent block is added to the private chain, the malicious group can still obtain the reward corresponding to the blocks on the private chain. Conversely, if the public chain is chosen, all blocks in the private chain will be discarded, and the malicious group will not be able to profit.

Figure 8 is a rough attempt to illustrate the selfish mining algorithm. $S0$ is the initial state. When the channel *attack?* fires synchronously with *attack!* in the malicious miner automaton, the state transfers to $S1$, which is regarded as “start of attack”. In state $S1$, there are two nondeterministic branches. The upper branch represents the first scenario of mining competition, while the lower branch moves toward the second and third scenarios of mining competition. The variable *private* represents the private chain’s length held by the malicious group, and the variable *public* is the length of the public chain maintained by other honest miners. Note that *public* is not the length of the main blockchain; it only represents the length of the public branch when the private branch is generated. The default values of *private* and *public* are both initially zero.

Figure 8. The selfish mining automaton.

It is indicated that the malicious group and honest miners are competing for mining at the same starting point. The variable δ is the difference in length between the private chain and the public chain and is used to distinguish the current competition between the two. When the automaton fires the transition from S1, the variable δ is updated first. At the beginning of the attack, since $private$ and $public$ are both defaulted to zero, the value of δ should be zero regardless of whether the state transfers to the upper or lower branch. If the automaton chooses the upper branch, the state transfers to S2. At this time, the malicious group successfully generates a block and adds it to the private chain. The assignment $private := private + 1$ implies that the length of the private chain increases by one. While the malicious group is mining, honest miners are also competing. At this time, if other malicious miners of the same group find the second block, the private chain is determined to be ahead of the public chain. Then, the private chain could be announced, and the selfish mining attack is successful. The state S6 after the transition indicates “a successful attack”. In state S3, if the malicious group is unable to obtain a new block faster than honest miners twice in a row, the state transfers back to S1, and the malicious group continues to compete with honest miners.

For simplicity, we did not consider the case that the malicious group holds a favorable position in computing power and keeps the private chain longer than the public chain. Therefore, the state invariant was set to enforce the transition. When $private$ is greater than two, the private chain should be announced. The automaton can select the lower branch to state S4, which implies “honest miners generate new block”. When the new block is added to the public chain, the variable $public$ increases by one. Then, the second or third scenario of competition may occur. In the second scenario, the guard condition $\delta == 0$ indicates that the private chain has fallen behind the public chain of the honest miner, so the malicious group can only immediately discard the private chain. In this case, the state transfers to S7, indicating “attack failure”. In the third scenario, the guard $\delta == 1$ means that the private chain and the public chain have the same length at this time, and the state transfers to S8, which implies that the two sides are equal in strength. Hence, the malicious group would like to compete again until the outcome is clear.

Unlike the honest miners, a malicious miner hides the blocks it generates. Such a behavior of hiding a block can happen in both the proposal and commitment steps, so we discuss the possible selfish mining attacks in these two steps separately. First, we assert that the attack launched at the proposal step will prevent malicious miners from gaining benefits. According to property P3, if the transaction cannot be received by the full nodes, it will not be regarded as proposed. In other words, this transaction is not a “valid transaction”; the state of the two-step automaton directly transfers to T9, and this transaction is not adopted. Some transactions may pass the proposal verification, but they

are not broadcast. In this case, these transactions are placed in the prefilled list and sent to the miners during the commitment step.

Next, we explored the scenario of starting a selfish mining attack during the commitment step and analyzed whether the CKB consensus protocol can effectively combat the attack. All the following properties were successfully verified in UPPAAL.

The property “all transactions must have appeared in the proposal zone before the commitment process” is formalized as (8). State *T5* is the first stage of the commitment process, and *cp* is a sign representing “the transaction already exists in the proposal zone”. The commitment process must only be the second step of the two-step confirmation. In other words, no transaction can skip the first step. According to (1), all transactions are processed by being put into the proposal zone. Before a transaction performs the second step, the nodes were informed of the transaction in the proposal zone of a block.

$$A [] \text{TwoStep.T5 imply } (cp == 1) \quad (8)$$

Before a transaction is committed, the full nodes should receive this transaction and verify its validity. This property is formalized as (9). State *T6* means “transaction is regarded as committed”; the condition *checkR* == 1 denotes that the full nodes have received the transaction; *checkV* == 1 is the sign that the full nodes have verified the transaction. The property (9) indicates that when the transaction is proposed, the full nodes have been informed of the transaction and the content of this transaction is confirmed. According to (8) and (9), a transaction cannot remain unknown after it is generated. Attributed to the role of the proposal step, the transaction must be announced in the first step.

$$A [] \text{TwoStep.T6 imply } (checkR == 1 \text{ and } checkV == 1) \quad (9)$$

Assuming that the malicious miner wants to hide the block in the second step, we have the following property (10): as long as the selfish mining attack is successful and a block and its included transactions are hidden in the commitment process, the block will not be propagated. State *S6* represents a successful selfish mining attack, and state *T7* stands for “block-propagation”. There is a case in which a transaction is regarded as proposed, but it does not appear in the commitment zone. This case only happens when the malicious miners launch a selfish mining attack. According to the protocol, the full nodes will request these missing transactions. If the malicious miner does not disclose the private blocks and transactions in time, the protocol prohibits the propagation of these blocks.

$$A [] \text{SelfishMining.S6 imply not TwoStep.T7} \quad (10)$$

The properties (8)–(10) together reveal then that CKB consensus protocol could prevent malicious miners from making unfair profits in selfish mining.

6. Related Work

There have been some results in the literature on the verification of blockchains and smart contracts. Based on these studies, we can see the practical meaning of applying formal verification techniques to blockchains.

Model checking approaches have been successfully applied in industry, especially for verification of hardware and communication systems, and also adopted recently in the verification of blockchain models. A formal model of the Bitcoin protocol using automata was developed in [9], in which the probability for double-spending attacks was also studied. The decentralized smart contract protocol (DSCP) was analyzed using game theory and the Markov decision process in [17], and the PRISM model checker was used to verify a family of DSCP properties. In [18], smart contracts were formally specified using Promela and verified in SPIN. The work in [19] adopted interface automata as the semantic model for smart contracts and used the NuSMV model checker to detect violations of the agreements. In [6], the Behavior Interaction Priorities framework (BIP) was used to specify the behavior

of smart contract implementation, and the blockchain behavior was verified using the statistical model-checking tool SMC.

Timed automata were adopted in [12] to develop a modeling framework for the Bitcoin contracts, and some security properties were verified based on this model. The runtime verification approach was investigated in [20,21], in which the formal model of the smart contract was provided using some form of automata. In [22], the behavior of EVM was formally defined in Why3, and a framework combining proofs and testing for the analysis of EVM and smart contracts was developed.

Meanwhile, there also exist some works on blockchain consensus. In [23], a detailed study of some network consensus algorithms was proposed. It is significant to compare different consensus algorithms as they are the key components in blockchain protocols. Based on model checking techniques, Reference [24] presented an interesting semi-automatic approach for asynchronous consensus algorithms.

To guarantee the trustworthiness of the CKB blockchain, we need to formally verify the CKB protocols. In previous works [14,15,25], we discussed this topic, and this paper extends our previous results by further investigating the robustness of the CKB protocols against malicious attacks. This work is helpful for the trustworthiness of the CKB blockchain.

7. Conclusions and Future Work

In this paper, we proposed a formal model of the CKB consensus protocol using timed automata and verified a family of properties related to the correctness and consistency of the CKB blockchain for different cases with or without malicious attacks in the UPPAAL model checker. We simulated potential malicious attacks in the experiments and investigated the impacts of such attacks. The properties that were formally verified provided a reference for possible scenarios of CKB applications. We hope that users of the CKB may understand the behavior of the CKB consensus protocol more precisely with the help of the formal model. According to the verification results, we can reasonably conclude that the CKB protocols are able to counter malicious attacks.

The CKB framework is still under development, and some possible optimizations might be adopted for the protocol to make better use of the bandwidth and computation resource. In the future, we hope to further develop the formal model to incorporate the optimization and provide better enhanced assurance for the trustworthiness of the consensus protocol. We will also investigate the formal model of the consensus protocol further to check the result under other kinds of attacks, such as the Sybil attack, etc. Additional investigation of different concrete application scenarios and the impact of the transport layer protocol on the CKB are in our scope as well.

Author Contributions: Conceptualization, M.S. and S.L.; methodology, Y.L., Q.Z. and Y.F.; software, Y.F. and Q.Z.; validation, M.S., S.L. and Y.L.; formal analysis, Y.F. and Q.Z.; investigation, Y.L.; resources, M.S. and S.L.; data curation, Y.F. and Q.Z.; writing—original draft preparation, Y.L., Y.F. and Q.Z.; writing—review and editing, M.S. and S.L.; visualization, Y.F.; supervision, M.S. and Y.L.; project administration, M.S.; funding acquisition, S.L. and M.S. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the Guangdong Science and Technology Department (Grant No. 2018B010107004), the National Natural Science Foundation of China under Grant No. 62172019, 61772038, and 61532019, ROIS NII Open Collaborative Research 2021-(21FS02), and Hiroshima University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors are grateful to the members of Cryptape, and the Nervos team for their helpful discussions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. Blockchain challenges and opportunities: A survey. *Int. J. Web Grid Serv.* **2018**, *14*, 352–375. [CrossRef]
2. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: <https://bitcoin.org/bitcoin.pdf> (accessed on 2 October 2021).
3. Ethereum. Available online: <https://github.com/ethereum> (accessed on 2 May 2020).
4. Xie, J. Nervos CKB: A Common Knowledge Base for Crypto-Economy. 2018. Available online: <https://github.com/nervosnetwork/rfcs/blob/master/rfcs/0002-ckb/0002-ckb.md> (accessed on 1 October 2021).
5. Bhargavan, K.; Blanchet, B.; Kobeissi, N. Verified models and reference implementations for the TLS 1.3 standard candidate. In Proceedings of the 2017 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 22–26 May 2017; pp. 483–502.
6. Abdellatif, T.; Brousmiche, K.-L. Formal verification of smart contracts based on users and blockchain behaviors models. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018; pp. 1–5.
7. Behrmann, G.; David, A.; Larsen, K.G. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*; Ser. LNCS; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3185, pp. 200–236.
8. David, A.; Larsen, K.G.; Legay, A.; Mikušonis, M.; Poulsen, D.B. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* **2015**, *17*, 397–415. [CrossRef]
9. Chaudhary, K.; Fehnker, A.; van Pol, J.; Stoelinga, M. Modeling and verification of the bitcoin protocol. In Proceedings of the Workshop on Models for Formal Analysis of Real Systems, MARS 2015, Ser. EPTCS. Suva, Fiji, 23 November 2015; pp. 46–60. Available online: <https://arxiv.org/abs/1511.04173> (accessed on 13 September 2021).
10. Zhang, R. CKB Consensus Protocol. 2019. Available online: <https://github.com/nervosnetwork/rfcs/blob/master/rfcs/0020-ckb-consensus-protocol/0020-ckb-consensus-protocol.md> (accessed on 1 October 2021).
11. Lu, Y.; Sun, M. Modeling and verification of IEEE 802.11i security protocol in UPPAAL for internet of things. *Int. J. Softw. Eng. Knowl. Eng.* **2018**, *28*, 1619–1636. [CrossRef]
12. Andrychowicz, M.; Dziembowski, S.; Malinowski, D.; Mazurek, L. Modeling bitcoin contracts by timed automata. In *Formal Modeling and Analysis of Timed Systems, Proceedings of the 12th International Conference, FORMATS 2014, Florence, Italy, 8–10 September 2014*; Ser. LNCS; Springer: Cham, Switzerland, 2014; Volume 8711.
13. Clarke, E.M.; Henzinger, T.A.; Veith, H.; Bloem, R. (Eds.) *Handbook of Model Checking*; Springer: Cham, Switzerland, 2018.
14. Zhang, Q.; Lu, Y.; Sun, M. Modeling and Verification of the Nervos CKB Block Synchronization Protocol in UPPAAL. In *Blockchain and Trustworthy Systems, Proceedings of the Second International Conference, BlockSys 2020, Dali, China, 6–7 August 2020*; Springer: Singapore, 2020; pp. 3–17.
15. Feng, Y.-C.; Lu, Y.; Sun, M. Modeling and Verification of CKB Consensus Protocol in UPPAAL. In Proceedings of the 33rd International Conference on Software Engineering & Knowledge Engineering (SEKE 2021), Pittsburgh, PA, USA, 1–10 July 2021; KSI Research Inc. and Knowledge Systems Institute: Skokie, IL, USA, 2021; pp. 150–153.
16. Nervos Network Homepage. 2020. Available online: <http://www.nervos.org> (accessed on 1 October 2021).
17. Bigi, G.; Bracciali, A.; Meacci, G.; Tuosto, E. Validation of decentralised smart contracts through game theory and formal methods. In *Programming Languages with Applications to Biology and Security—Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*; Ser. LNCS; Springer: Cham, Switzerland, 2015; Volume 9465, pp. 142–161.
18. Bai, X.; Cheng, Z.; Duan, Z.; Hu, K. Formal modeling and verification of smart contracts. In Proceedings of the 2018 7th International Conference on Software and Computer Applications (ICSCA 2018), Kuantan, Malaysia, 8–10 February 2018; ACM: New York, NY, USA, 2018; pp. 322–326.
19. Madl, G.; Bathen, L.A.D.; Flores, G.H.; Jadav, D. Formal verification of smart contracts using interface automata. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 556–563.
20. Ellul, J.; Pace, G.J. Runtime verification of ethereum smart contracts. In Proceedings of the 2018 14th European Dependable Computing Conference (EDCC), Iasi, Romania, 10–14 September 2018; IEEE Computer Society: Washington, DC, USA, 2018; pp. 158–163.
21. Azzopardi, S.; Colombo, C.; Pace, G.J. Model-Based Static and Runtime Verification for Ethereum Smart Contracts. In *Model-Driven Engineering and Software Development, Proceedings of the 8th International Conference, MODELSWARD 2020, Valletta, Malta, 25–27 February 2020*; Revised Selected Papers, Ser. CCIS; Springer: Cham, Switzerland, 2021; Volume 1361, pp. 323–348.
22. Zhang, X.; Li, Y.; Sun, M. Towards a Formally Verified EVM in Production Environment. In *Coordination Models and Languages, Proceedings of the 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Valletta, Malta, 15–19 June 2020*; Ser. LNCS; Springer: Cham, Switzerland, 2020; Volume 12134, pp. 341–349.
23. Duan, Z.; Mao, H.; Chen, Z.; Bai, X.; Hu, K.; Talpin, J.-P. Formal modeling and verification of blockchain system. In Proceedings of the 10th International Conference on Computer Modeling and Simulation (ICCMS 2018), Sydney, Australia, 8–10 January 2018; pp. 231–235.
24. Tsuchiya, T.; Schiper, A. Verification of consensus algorithms using satisfiability solving. *Distrib. Comput.* **2011**, *23*, 341–358. [CrossRef]
25. Bu, H.; Sun, M. Towards Modeling and Verification of the CKB Block Synchronization Protocol in Coq. In *Formal Methods and Software Engineering, Proceedings of the 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, 1–3 March 2021*; Springer: Cham, Switzerland, 2020; pp. 287–296.