Memory Pool Publisher Algorithm for Preventing Malicious Fork in the Bitcoin Environment

Ahmed Madkour [*a], Hatem Abdelkader [a], Asmaa Elsaid [a]
[a] *Information Systems* dept. , *Faculty of Computer and Information,* Menoufia University, Egypt
hamdymadkour@ci.menofia.edu.eg, hatem.abdelkader@ci.menofia.edu.eg, asmaa.elsayed@ci.menofia.edu.eg

Abstract—Recently, blockchain is considered the most reliable data storage technology that stores data in blocks. Many cryptocurrency frameworks such as Bitcoin, therefore, utilize blockchain technologies to preserve all past details on blocks. These blocks are related as a chronological link list and can be changed to a fork structure. Which there are two types of fork: accident fork which appears when two blocks are published in the blockchain at the same time. On the other hand, there are selfish miner attackers who have supercomputer properties to generate illegal blocks and maximizing their illicit reward (intentional fork). A set of blockchain transactions will be rollbacked when the malicious fork appears in the Bitcoin environment, therefore, user waiting times and illicit miner rewards will increase. In this paper, The proposed algorithm is a lightweight algorithm (Memory pool publisher) that has used the Bitcoin memory pool as a publisher of blocks to prevent the Bitcoin environment's malicious fork and rollbacked transaction. The findings show that the major cause of the malicious fork is the multiplicity of blockchain publishers of blocks in the bitcoin environment. The proposed algorithm success to solve the limitation of the previous fork. Which has a strong potential to prevent the intentional and accident fork problem, and the rollback problem. Therefore minimize user waiting times from 60 minutes to 10 minutes by makes one block publisher to the blockchain.

Keywords—Cryptocurrency; Bitcoin; Fork; Rollback; Selfish Mining Attacker

I. INTRODUCTION

Online currency exchange, such as Bitcoin[1], is a safe idea to move currency between users without any fees[1]. Bitcoin uses P2P technology and works with no confident third party jurisdiction, which may function as a bank, a CA, a notary, or other centralized services[1]. Bitcoin uses P2P (peer-to-peer technology) technology. The owner has absolute ownership of his bitcoins and will invest them anytime and anywhere without requiring any centralized authority. Bitcoin architecture is open-source, and no one owns or manages it. Besides, Bitcoin is a cryptographically protected electronic payment mechanism that facilitates transactions involving virtual currencies in the form of digital tokens called Bitcoin coins (BTC or simply bitcoins).

There are a variety of implementations in blockchain technologies, such as cryptocurrency [2], Internet [3], and the supply chain. The government [4] continuously use blockchain software capable of creating consensus in a shared environment. A distributed ledger is developing as the name means, a blockchain is a series of data blocks that have a single way hash function to cryptographically chain to each other. Blockchains use these logical frameworks. Blockchain is like the liked list structure that stores chronological blocks of data. On other hand, many attackers may be changed this chain to a malicious fork. A malicious fork happens when two blocks are published in the blockchain (Accident fork) at the same time, or a miner has supercomputer properties that produce a series of blocks as a branch and does not publish this branch to the blockchain after reaches the length of the main branch (Intentional fork) [5], which is referred to as the "Selfish Mining Attack" to maximizing the miner's illicit reward [6].

Our motivation from this study: our proposed algorithm aims to keep the blockchain as a linked list, prevent the malicious fork, reduce user waiting time, and reduce illicit miner rewards.

Our Contribution to this study: the Memory pool algorithm is presented to prevent the malicious fork and ensure that the blockchain is maintained as a linked list. Memory pool publisher is recommended and therefore the rollback problem is avoided, as well as reducing user waiting times and illicit miner rewards. Memory pool publisher makes one block publisher in the Bitcoin environment and divides block construction into two phases. The miner creates the block and transfers this block to the memory pool in the first phase. The memory pool publishes all received blocks to the blockchain in the second phase.

The paper is organized as follows: Section II presents a review of the previous work that highlighted the malicious fork. In section III, the proposed algorithm is introduced to try to prevent the malicious fork. Section IV presents the result of a comparison between the previous work algorithms that tried to prevent the malicious fork and the Memory pool publisher algorithm. Finally, Section V introduces the conclusion of the Memory pool publisher algorithm.

II. BACKGROUND

The framework of Bitcoin includes a set of nodes: user, miner, and memory pool [7]. Users would be able to send and receive currency [7,8]. Each transformation is stored in a transaction and each set of transactions are stored in blocks. Each transaction includes lock time, sender key, sender signature, the final amount after this transformation of the sender (change), receiver key, and transformation amount and fee that only use fees to accelerate the transaction processing. The block is produced to validate a series

of transactions in an average of ten minutes. [9], which contains block id, header, and body. The block header contains the version of the Bitcoin system, previous block id, timestamp of the block generation, nonce, shared target, and Merkle root address. To create blocks, the Bitcoin framework requires at least one miner who enters the scheme as a user and builds new blocks to win any Bitcoins using his computing tools [7]. The miner obtains from the memory pool a group of transactions and verifies the transaction signature and the sender's possession of the amount and fee of the transaction. To prevent the user from being able to send the same currency to more than one user at the same time[10,11], which is called "double-spend attacks" each sender may make only one transaction at the same block. Then, as the first transaction in the block, the miner produces a transaction reward, builds a block, and stores this block in the local blockchain, and transfers this block to all other miners. When the other miners receive a block, check the previous block id and approve this block if the previous id is located in the local blockchain and the block id is lower than the shared target. Otherwise, the miners would reject this block. All transactions are processed in the memory pool and sorted by the highest fee, where the transaction with a high fee is first confirmed. Then the memory pool gives all miners a series of ordered transactions based on the block size [9,12]. However, owing to certain attacks on the bitcoin infrastructure, or where two blocks are stored at the same time in the blockchain, those transactions are undone and returned to the memory pool [7].

Blockchain is a framework connected to the list but can be modified to a structure fork. There are two types of forks: a useful fork or a malicious fork. Using the useful fork to upgrade the soft or hard Bitcoin system or blockchain laws [13]. For instance, If the block size is 8 megabytes, up to 16 megabytes must be modified. The hard fork is used when the two types of block sizes are 8 and 16 MB. The soft fork is used when only the current block size (16 M) is used. There are two types of malicious fork: accident and intentional fork. The intentional fork is shown in Fig. 1 consists of two branches the honest branch and the private branch (attacker's branch). The miner must choose the highest branch as the main chain or if the two branches are equal in length, pick a random branch as the main chain [7, 14]. Other branch transactions are rollbacked to the memory pool [15]. Consequently, the miner is unable to get all the blocks in the blockchain and cannot check the sender's amount. The rollback issue is an incomplete process for a transaction and this incomplete transaction is returned to the memory pool. Furthermore, rollback is a user time-consuming issue. At the end of a transaction, the user waits for transaction confirmation. A transaction will be confirmed after the release of six blocks in the blockchain [16]. So, the user waiting time is six blocks after the block that includes his transaction. If the six blocks or his block were rollbacked, the waiting period will increase. For illustration, If the attacker has 10 percent of mining resources, the probability of publishing the next block after two blocks is 10 percent, 1 percent after four blocks, and 0.1 percent after six blocks, therefore, all block transactions are confirmed after generating six blocks in the same branch [16].

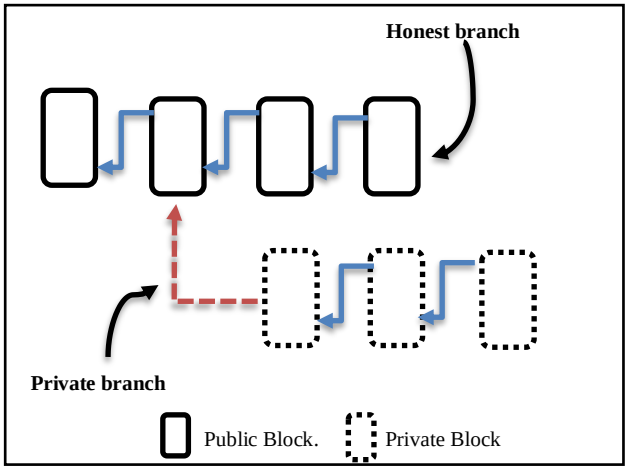

Fig. 1. Intentional fork.
Note: this figure is a graph which show the Intentional fork issue

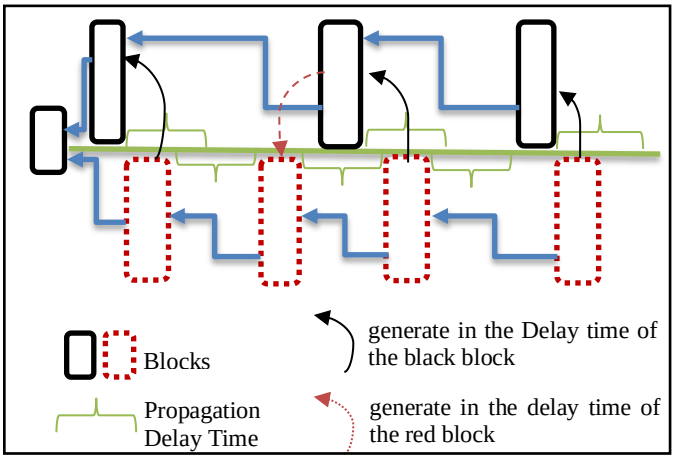

Fig. 2. Publish or perish example.
Note: this figure is a graph which show the Publish or perish algorithm and the uncle blocks.

III. Related Work

There are many ways proposed in the literature that has dealt with the malicious fork issue. This section reflects research efforts to address the malicious fork issue and comparison between them by three metrics: accident fork, intentional fork, and rollback transactions. There are two viewpoints on all research efforts that address the issue of the malicious fork.

A. First Viewpoint.

Some authors select one branch as the main branch and rollback the other branches to maintain the blockchain as a linked list structure. In [17] the block timestamp is used to select the main chain. The selection of the main branch depends on the timestamp of each branch's last block. This algorithm selects a branch as the main chain with the most recent timestamp and if all branches have the same timestamp, picks the first obtained branch timestamp as the main chain. Otherwise, the largest block length branch is selected as the main chain. In [18] The weight of a branch is used instead of the length of the branch to select the main chain. Each block requires a waiting time for the other miners to publishing. For all blocks, this algorithm assumes the propagation delay time is constant. The branch weight is determined by accumulating the branch length and the complete blocks are published in the other

branch during the delay time of each block of the branch (uncle block). The main branch with the largest weight is picked by the algorithm. For instance, as shown in Fig. 2, blockchain contains two branches: black and red which each block has a propagation delay time, and calculate the branch weight by accumulating the branch length and uncle blocks. The weight of the black branch is 3+3=6 and the weight of the red branch is 4+1=5. Then the miners accept the black branch as the main chain and rollback the red branch though the length of the red branch greater than the length of the black branch. In [20] the expected block number is used to select the main chain. This algorithm saves the expected block number in all un-confirmed transactions of the memory pool depend on the fee of the transaction, block size, and blockchain length. When the malicious fork has occurred, this algorithm will calculate the truth state of all branches depend on the expected block number of all transactions. It determines the main branch with the highest truth state. For instance, as shown in Fig. 3, every Memory pool transaction contains the expected block number and when appear a fork in the blockchain accepts the branch with the less wrong expected number. The number of the wrong expected number of the green branch is 4 and 1 for the red branch. Therefore then accept the red branch and rollback all transactions of the green branch.

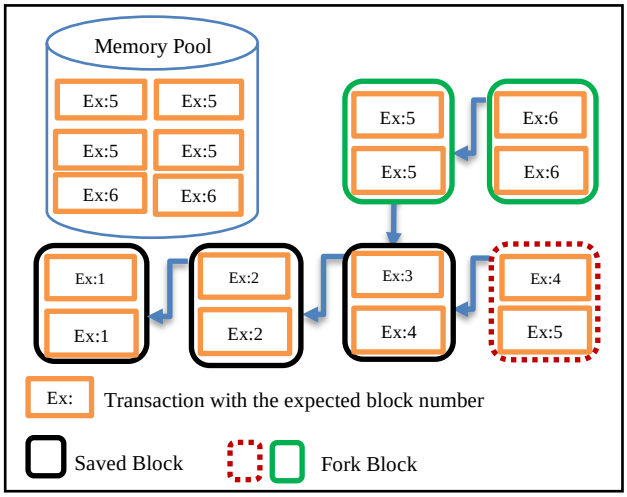

Fig. 3. Countering selfish mining in blockchains example.
Note: this figure is a graph which show that each transaction contains the expected block number and the blockchain contains two branches: red and green.

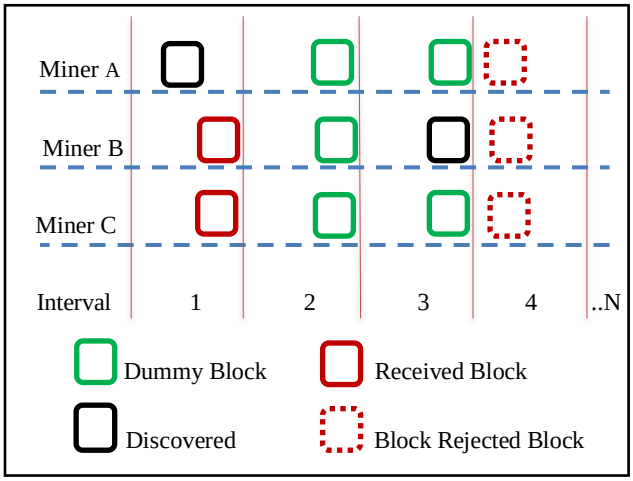

Fig. 4. Zeroblock: Timestamp-free prevention of the block-withholding attack in Bitcoin example.
Note: this figure is a graph and shows that Miner A and C are Honest Miners and Miner B is an attacker.

Table 1. A comparison between the advantages and disadvantages of related work algorithms.

Algorithm	Accident fork	Intentional fork	Rollback transaction	Limitations
One weird trick [17]	Remove the accident fork depending on the timestamp.	Failed to remove the intentional fork.	Rollback too many transactions.	• Pick the private branch and back off from the public branch. • Fail to solve the accident fork problem, because the miner will select the recent branch and rollbacks the other honest branches.
Publish or perish [18]	Failed to remove,	Remove the intentional fork, pick the highest weight branch as the main branch, and rollback the other branches to the memory pool.	Rollback too many transactions.	• The propagation delay time is assumed to be constant, however, is the opposite of reality. • The algorithm fails if it does not have time to create another branch. • Lacks to solve the accident fork problem, because all branches have the same weight. • Increase the load of the miner to the weight of each branch.
Countering [19]	Failed to remove.	Remove the intentional fork, pick the most correct expected block branch as the main branch.	Rollback too may transaction.	• Fail to calculate the expected block number if the transaction was waiting for more time in the memory pool. • Fail to solve the accident fork problem.
Zeroblock [21]	Failed to remove.	Prevent the intentional fork.	Rollback may transactions.	• Increase the load of the miner to generate the dummy block every interval. • Fail to solve the accident fork problem.

B. Second Viewpoint.

Some authors try to prevent the fork to keep the blockchain as a linked list structure. In [20] the dummy block is used to prevent the intentional fork. The time is divided into intervals. Miner will receive or generate a block for each interval, otherwise, generate a dummy block. The dummy block does not contain transactions however contains only a nonce to the next block "Zeroblock". Each miner will generate a block depend on the nonce of the previous dummy block and send this block to all miners, or receives a block from the other miner and compare between the nonce of the received block and the nonce of the previous dummy block. The miner accepts the block that contains the nonce of the previous dummy block. For instance, as shown in Fig. 4, there are three miners (A, B, and C) and four intervals. In the first interval, miner A generated a block and sent it to miners B and C. In the second interval, all miners don't discover a new block therefore a dummy block is generated and saved in all miners (A, B, and C). In the third interval, the dummy block is generated and saved in miners A and C, however, miner B discovered a new block and sent this block to miners A and C. In the fourth interval, miner A and miner C rejected this block, because the nonce of the previous block not the same as the nonce of the received block.

Although these algorithms tried to solve the malicious fork issue, however, there are some shortcomings as shown in table 1. One weird trick [17] failed to solve the intentional fork and rollback too many transactions because it accepts the recent timestamp branch and rollbacked the other branches. Publish or perish [18] failed to solve the accident fork and too many transactions because it depends on the weight of each branch and all branches contain the same weight and it picks one branch as the main chain and rollback other branches to the memory pool. Countering [19] failed to the accident fork because all branch transactions contain the same expected block number and rollback many transactions pick one branch as the main chain and rollback all transactions of all other branches. Zeroblock [20] failed to remove the accident fork because all branches contain the same block nonce and rollback many transactions if more than one miner generates a block at the same period.

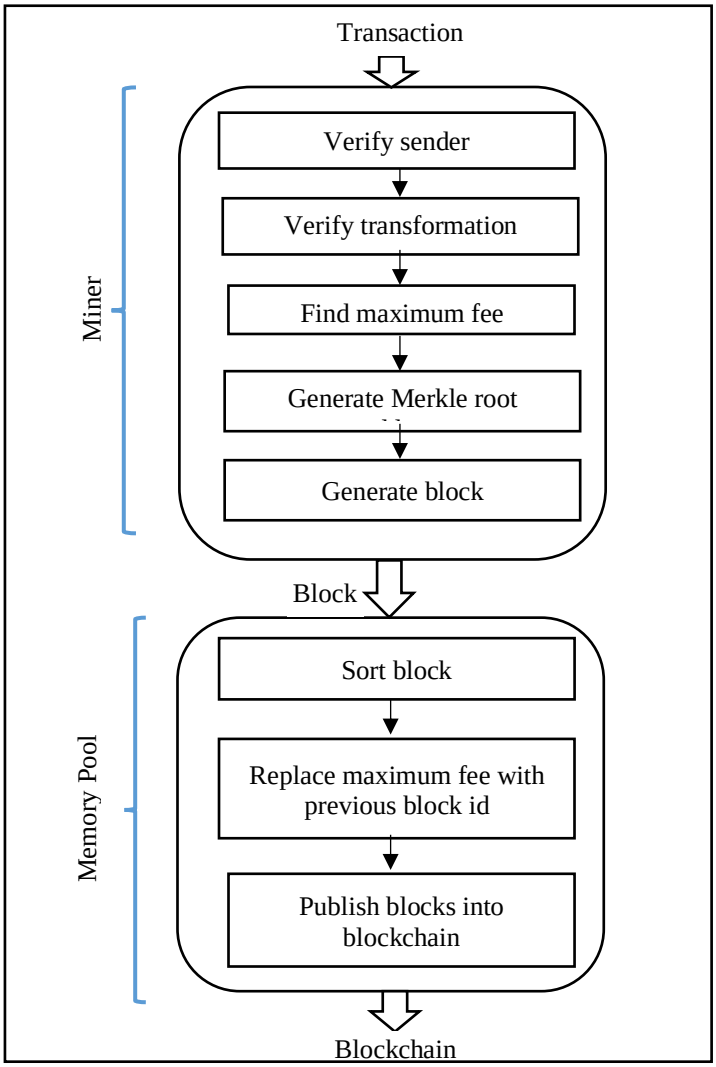

Fig. 5. Memory pool publisher algorithm phases.
Note: this figure is a block diagram which show that Memory pool publisher divides into two phases: miner build a block in the first phase and memory pool publish blocks to the blockchain in the second phase

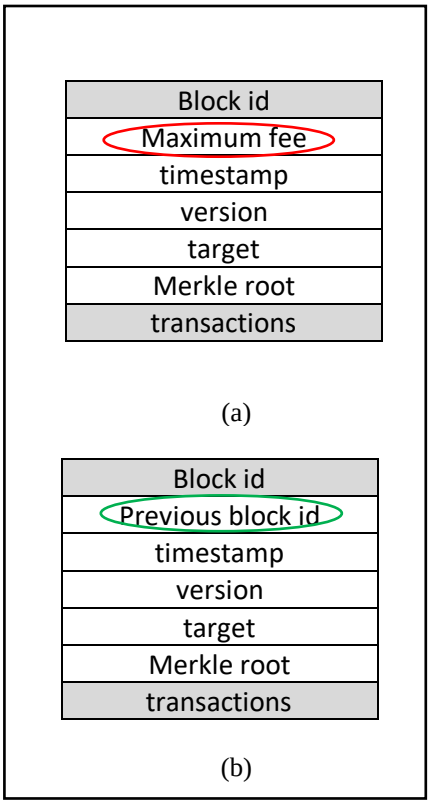

Fig. 6. the Memory pool publisher algorithm phases.
Note: this figure is a graph which show the block form in the first phase save the maximum fee and second phase of the memory pool algorithm replace the maximum fee by the previous block ID.

IV. PROPOSED ALGORITHM

In this section, two subsections will be illustrated. The first briefly explains how the proposed Memory pool publisher algorithm tries to prevent the malicious fork and its deference from the original bitcoin scenario. The second subsection explains the proposed Memory pool publisher algorithm notations, input, output, and steps taken to avoid malicious fork.

A. Memory pool publisher description

After reviewing the previous work and analyzing the limitations of related work algorithms, all related work algorithms try to solve the malicious fork itself, not solve the reasons for occurring these forks. The multiplicity of blockchain publishers in the bitcoin environment are the cause for the incidence of the malicious fork problem. In the original bitcoin scenario, all miners can publish any number of blocks at any time which leads to the multiplicity of blockchain publishers, therefore, the malicious fork has appeared. The proposed algorithm is a new lightweight algorithm and aimed to avoid the existence of malicious fork problems by preventing the multiplicity of blockchain publishers in the blockchain. At the same time, the miner's efforts are used only for the process of generating blocks, not in additional works like the Publish or perish [18] and Zeroblock algorithm [20]. As well as, additional space don't use in the transactions like the Counting algorithm [19].

The Memory pool publisher algorithm tried to prevent the miner from publishing blocks and make it concentrate solely on generating the blocks. The memory pool is assumed to be trusted (not malicious) such as the original scenario of the bitcoin environment, the memory pool is assumed to be trusted and is used to stores the unconfirmed transactions [1]. The Memory Pool makes only the publisher of blocks in the blockchain framework. The block construction is divided into two phases: generation and publishing. The miners responsible for the generation phase, and the memory pool to the publishing phase. As shown in Fig. 5, the miner receives a set of transactions (input), and the miner applies some sequential steps: firstly, check the user authentication. Secondly, the miner checks the user amount compared to the transformation amount to ensure the correctness of the transformation. Thirdly, miners find the maximum fee for all received transactions. Fourthly, the miner generates the Merkle root address by merging each adjacent transaction ids by applying the SHA256 hash function [21,22]. SHA256 hash function generates one address as a parent of these adjacent ids, and this process is repeated until generating one address from all input transactions (Merkle root address) [23]. Fifthly, the miner builds an initial form for the block as shown in Fig. 6a which saves the maximum fee instead of the previous block ID. Finally, the miner sends this block to the memory pool (output). Then, the memory pool receives a set of blocks (input), and some sequential steps are applied to these blocks: firstly, the Memory pool sort the received blocks in a shared stack depends on the timestamp and maximum fee of the initial form of the block. Which the memory pool sorts the received blocks depend on the block timestamp to chronological order. If the memory pool received two or more blocks and these blocks contain the same timestamp, sorting this block depends on the maximum fee because users pay fees to accelerate the confirmation of their transaction. Secondly, the Memory pool replaces the maximum fee by the previous block id as shown in Fig. 6b to connect this block by the last block of the blockchain. Thirdly, the memory pool publishes these blocks into the blockchain. Finally, the memory pool publishes the blockchain (output).

After the Memory Pool algorithm has been implemented, the advantage of the unification of publishers has come which the memory pool just is the publisher of the Bitcoin environment. Therefore, the Memory pool cannot publish two blocks at the same time (accident fork) and all private blocks of the selfish mining attacker (intentional fork) will receive by the Bitcoin publisher (Memory pool) and publishes serially. As a result, the blockchain all time remains a straight line. Therefore, the rollbacked issue is prevented, minimize illicit miner rewards, and user waiting time are and are reduced from 60 to 10 minutes. Which the user waits in the original scenario after confirming six blocks and each block takes 10 minutes to generate. On the contrary, all related work algorithms allows publishing the block/blocks to all miners any time therefore the malicious fork has appeared.

B. The Memory pool publisher algorithm

A brief overview of the Memory pool publisher algorithm generation and publishing phases is provided in this section. Their notations, inputs, outputs, and implementation steps are discussed below:

1. Generation Phase algorithm

For the legibility purpose of the proposed algorithm, some notations are presented as follows:

1. **NTs**: {tx1, tx2,..., txn}: the new transactions that need to be confirmed.
2. **BC**: {b1, b2,..., bn} : the shared blockchain.
3. **VTs**: {tx1, tx2,..., txn}: all transactions of the trusted user with enough currencies to transformation (VerifiedTransactions).
4. **NB**: the generated block.
5. **MF**: the maximum fee of the new transactions.
6. **balanced**: calculated current amount of the sender.
7. **MRA**: Merkle root address.

8. **ID**: identifier of the new block.
9. **merkle**: a function to generate the Merkle root address.
10. **build**: a function to build the block.
11. **TX: a transaction that includes a set of attributes**: sender public key (sender), Receiver public key (receiver), the final amount of the sender after the transformation (change), Transformation amount (transformation), a fee to accelerate the transformation process [24], and Transaction signature (sig).

As shown in algorithm 1, the input data is the new transactions and shared blockchain. The output is the new block. This algorithm is divided into four steps:

1. In the first step at line 5: as shown in Eq. (1) and Eq. (2) The sender is checked by using a transaction signature composed of the public key of the sender and all transaction attributes used to produce the signature [25]. The public key is validated and approve the transaction if it is legitimate, the transaction signature, and all attributes of the transaction are used.

$$Sig = SHA256(private\ key, attributes) \quad (1)$$

$$Public\ key = SHA256(sig, attributes) \quad (2)$$

Algorithm 1: Generation phase Algorithm

Input: *NTs: new transaction and BC: shared Blockchain*
Output: *NB*: Block of the new verified transactions

```
1  begin
2    VTs ← φ
3    MF ← 0
4    foreach TX ∈ TXs do
5      if verify(TX[sig], TX[attributes], TX[sender]) = true then
6        balanced ← 0
7        foreach block ∈ BC do
8          foreach transaction ∈ block[transaction] do
9            if transaction[sender] = TX[sender] then
10             balanced ←
                 balanced + transaction[change]
11       break
12           else if
               transaction[receiver] = TX[sender] then
                 balanced ← balanced+ transaction[transformation]
13           end if
14         end foreach
15       end foreach
16       if balanced >= TX[transformation] then
17         VTs ← VTs ∪ {TX}
18       if TX[fee] > MF then
19         MF ← TX[fee]
20       end if
21       end if
22     end if
23   end foreach
24   MRA ← merkle(VTs)
25   ID ← generate(MRA)
26   NB ← build(ID, MRA, VTs)
27   return NB
28 end
```

Algorithm 2: Publishing phase Algorithm

Input: *NBs*: new blocks and *BC*: shared Blockchain
Output: *BC*: new Blockchain

```
1  begin
2      SBs ←sort(NBs, timestamp, fee)
3      PBID ←BC[lastBlock][ID]
4      foreach B ∈ SBs do
5       B ←replace(B, fee, PBID)
6       PBID ←B[ID]
7       BC ←BC ∪ {B}
8   end foreach
9   return BC
10 end
```

2. In the second step from lines 7 to 15: verifies the Transformation amount (transformation) by scanning all blockchain transactions to calculate the balanced (calculated current amount of the sender). As shown in Eq. (3). For all transactions (TXs), If the sender of the new transaction is the sender of the current transaction, the balance is added by the transaction change and stop the scanning. However, if it is the receiver of the current transaction, the balance is added by the transformation. Then, accept a transaction from the sender if the balanced is greater than the summation of the transformation amount and fee.

$$balanced = balanced + \begin{cases} charge, owner\ is\ sender \\ transformation, owner\ is\ reciever \end{cases} \quad (3)$$

3. Third step from line 16 to 22: add transaction (TX) to the verified transactions (VTs) and find the maximum fee of VTs.
4. Fourth step from lines 24 to 27: the Merkle root address (MRA), transaction id (ID), and block are generated.

2. Publishing phase algorithm

For the legibility purpose of the proposed algorithm, some notations are presented as shown in Table 3:

1. **NBs**: {b1, b2,..., bn} : new blocks that need to be confirmed.
2. **BC**: {b1, b2,..., bn} : the shared blockchain.
3. **SBs**: {b1, b2,..., bn}:sorted blocks by timestamp and fee.
4. **PBID**: the generated block.
5. **sort**: the maximum fee of the new transactions.
6. **Replace**: calculated the current amount of the sender.
7. **B**: a block which includes a set of attributes: Block id (ID), Previous bock id (PBID), Timestamp, System Bitcoin version, Merkle root address (MRA), Transactions.

As shown in algorithm 2, the input data is the new blocks and shared blockchain generated from algorthim1. The output of algorithm 2 is the blockchain and this algorithm is divided into three steps:

1. At line 2: sort the blocks by the smallest timestamp and Maximum fee.
2. At line 5: replace the Maximum fee with the previous block id.
3. At line 7: publish the new block to the blockchain.

After these two algorithms (phases of the memory pool publisher algorithm) are implemented, miners focus solely on the block generation processes as the original scenario of the Bitcoin environment (so-called mining pools) [7] and send it to the memory pool instead of the blockchain. Which the memory pool is the one responsible for block publishing. As a result of these, the Memory pool publisher algorithm succeeds to unify publishers of blocks. Subsequently, the malicious [5,26] fork and rollbacked [27] issues are avoided and the user waiting time is reduced to 10 minutes.

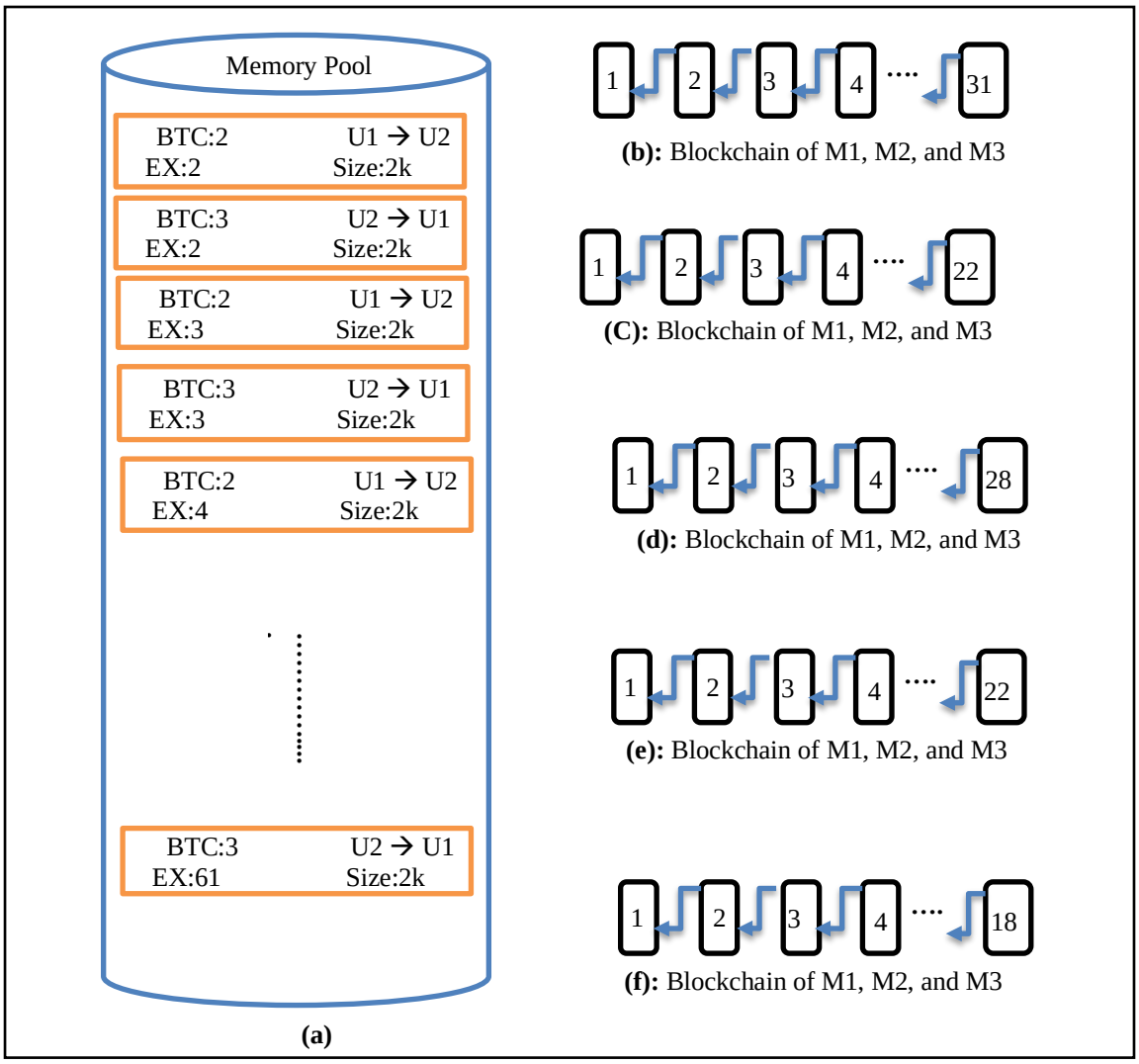

Fig. 7. Blockchain form after applying all algorithms with the intentional fork issue.
Note: this figure is a block diagram which, (a) show the new transactions of the Memory pool (unconfirmed transactions), (b) show the blockchain of the Memory pool publisher, (c) show the blockchain of One weird trick algorithm, (d) show the blockchain of Zeroblock algorithm, (e) show the blockchain of Publish or perish algorithm, and (e) show the blockchain of Countering algorithm.

V. EXPERIMENTAL RESULT AND DISCUSSION

In this section, a real case of the Bitcoin system is applied to compare the suggested algorithm provided in this paper and the algorithms described earlier in Section III. Three nodes are used with this characteristic: processor: Pentium(R) Dual-Core CPU T4200 @ 2.00 GHz, installed memory: 4.00 GB (3.8 GB usable), and system type: 32-bit Operating system. As well as, four nodes with this characteristic: processor: Intel(R) Core(TM) i7-7500U CPU @ 2.7GHz 2.9GHz, installed memory: 8.00 GB (7.6 GB usable), and system type: 64-bit Operating system, x64-based processor. These machines are used to simulate the bitcoin system and are connected via a local peer-to-peer network, in which the seven machines are distributed as follows: One machine is an authentication server, for adding users and miners to the system. One machine is a memory pool node. Two machines as user nodes (U1, U2) each of which has 5 Bitcoins. Three machines as miner nodes (M1, M2, M3).the transaction size is 2k and each block size is 4k (each block contains two transactions). The blockchain starts with one block to confirm five Bitcoins for U1 and five Bitcoins for U2. U1 sends 2 Bitcoins to U2 (Transaction 1), U2 sends 3 Bitcoins to U1 (Transaction 2), and repeat these transactions 30 times automatically (60 transactions total). The memory pool contains 60 transactions which each transaction contains transformation amount, sender and receiver, expected block number, and size as shown in Fig. 7a. The results of this experiment are analyzed based on three metrics: generation time, the intentional fork [5], and the accident fork [26].

A. Generation time

For the proposed algorithm discussed in this paper, the block generation time was constant, and the algorithms were specified on a ten-minute average in section 2 and regulated by the shared target. The shared target is a constraint on the block generation to keep the average block generation in ten minutes to reduce the load on the miner [9].

B. Intentional fork

M1 and M2 machines are honest miners, and M3 is a miner attacker with supercomputer characteristics. M3 is trying to win by illicit bitcoins by making an intentional fork. The memory pool includes 60 transactions, every two transactions need one block to confirm, and each block spends 10 minutes to be generated. Before this experiment starts the blockchain of each miner includes one block (two transactions). After 600 minutes of running the Memory pool publisher algorithm as shown in Fig.7b, the blockchain of M1, M2, and M3 are increased by 30 blocks. Since each block contains two transactions, therefore transactions that are confirmed are 60 transactions. It is inferred from the experiment that the problem of rollback did not arise. But after repeating the same experiment with the One weird trick algorithm [17] as shown in Fig. 7c, each miner's blockchain was increased by 21 blocks or increased by 42 transactions (42 confirmed transactions) then 18 transactions are returned to the memory pool (18 rollbacked transactions). As well as or the Zeroblock algorithm [20] as shown in Fig. 7d, the blockchain will be increased by 13 blocks (26 confirmed transactions and 34 rollbacked transactions). As shown in Fig. 7e, for the Publish or Perish algorithm [18], the blockchain is increased by 21 blocks (42 confirmed transactions and 18 rollbacked transactions). Finally as shown in Fig. 7f, for the Countering algorithm [19], the blockchain is increased by 17 blocks (34 confirmed transactions and 26 rollbacked transactions). As shown in the experimental results of the intentional fork in Table 2, the rollback problem occurred in all algorithms mentioned in Section III, but it did not appear in the Memory pool publisher algorithm.

Table 2. Intentional Fork Result

Algorithm	Number of transactions		
	Before the experiment	After the experiment	Rollbacked
One weird trick [17]	2	44	18
Zeroblock [20]	2	28	34
Publish or perish [18]	2	44	18
Countering [19]	2	36	26
Memory pool publisher	2	62	0

C. Accident fork

The previous experiment is repeated to replace the miner M3 with honest miner M4 and after 600 minutes. As shown in the experimental results of the accident fork in Table 3, the rollback problem occurred in the previous algorithms, but it didn't appear in the Memory pool publisher algorithm. Consequently, the accident issue didn't occur in the Memory pool publisher algorithm.The rollback issue appears in all previous algorithms and doesn't appear in the memory pool algorithm when applying the accident and intentional fork as shown in Fig. 8. consequence is that the Memory pool publisher algorithm is designed to prevent the malicious forks issue from arising by making the memory pool is just the publisher of the Bitcoin system (unifying block publishers) and making the miner just focus on the block construction.

Table 3. Accident fork result

Algorithm	Number of transactions		
	Before the experiment	After the experiment	Rollbacked
One weird trick [17]	2	54	8
Zeroblock [20]	2	57	5
Publish or perish [18]	2	58	4
Countering [19]	2	55	7
Memory pool publisher	2	62	0

VI. CONCLUSION

Malicious forks are the worst challenge of blockchain technologies and thus raise the rollbacked transaction issue, therefore, increased transaction confirmation time, and illicit rewards of the miner. Current algorithms have attempted to overcome the malicious fork by accepting one branch and rollbacking other branches but have not tried to avoid these forks therefore the rollback transaction issue appearing. In this paper, we found that the key source of the malicious fork is the multiplicity of blockchain

publishers on the blockchain. Which all miners can publish any number of blocks at any time, therefore, the malicious fork has appeared. The Memory pool publisher algorithm is proposed to prevent the malicious fork by unifying publishers of blocks, therefore, do not publish two blocks at the same time. The proposed algorithm divides the block construction process into two phases: In the first phase, the miner generates a block and saves the maximum fee of all transactions in the block header, and sends this block to the memory pool. In the second phase, the memory pool sorts the received blocks by the recent timestamp and the maximum block fee if the memory pool receives more than one block with the same timestamp and publishes these blocks to all miners.

Experimental results show that the Memory pool publisher algorithm prevents the malicious fork, therefore prevents transaction rollback problems, and reduces transaction confirmation time to 10 minutes, and reduces illicit miner rewards. The advantages of the Memory pool algorism: it keeps the blockchain as a linked list, prevents the malicious fork, and prevents the rollback issue. The disadvantages of the Memory pool algorism: the Memory pool bottleneck problem, because the bitcoin system only has one publisher. It increases the waiting time of block generations which sending all blocks to the memory pool and publishing them. The miner will publish the block directly to the blockchain in the original scenario. Therefore, future work on this object we will try to prevent the malicious fork problem with multiple memory pools, support the useful fork in the blockchain.

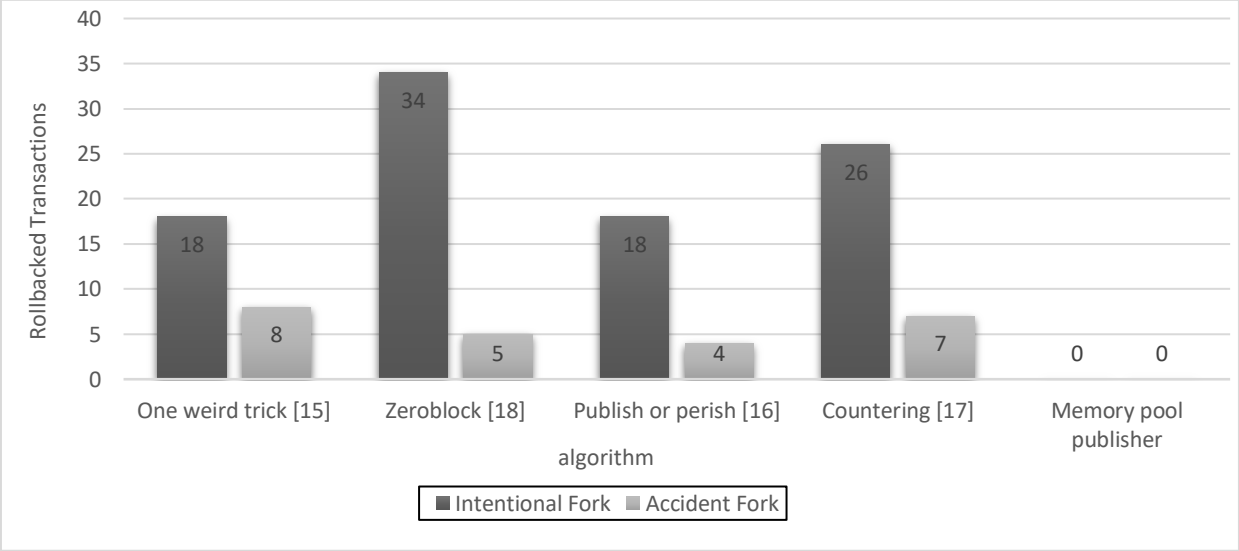

Fig. 8. A comparison between the previous and Memory pool publisher algorithms
Note: this figure is a column chart that shows that the rollback issue does not appear in the Memory pool publisher.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Available:http:// bitcoin.org/ bitcoin.pdf, 2008 (access tim: 7-2-2021).

[2] G. Danezis and S. Meiklejohn, “Centrally banked cryptocurrencies,” in Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2016. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/centrally-banked-cryptocurrencies.pdf (access tim: 7-2-2021).

[3] E. F. Jesus, V. R. L. Chicarino, C. V. N. de Albuquerque, and A. A. de A. Rocha, “A survey of how to use blockchain to secure internet of things and the stalker attack,” Security and Communication Networks, vol. 2018, pp. 9 675 050:1–9 675 050:27, 2018. [Online]. Available: https://doi.org/10.1155/2018/9675050 (access tim: 7-2-2021).

[4] M. M. Eljazzar, M. A. Amr, S. S. Kassem, and M. Ezzat, “Merging supply chain and blockchain technologies,” Computing Research Repository (CoRR), vol. abs/1804.04149, 2018. [Online]. Available: https://goo.gl/5wMVJS (access tim: 7-2-2021).

[5] Chicarino, Vanessa, Célio Albuquerque, Emanuel Jesus, and Antonio Rocha, "On the detection of selfish mining and stalker attacks in blockchain networks." Annals of Telecommunications (2020): pp. 1-10.

[6] Qin, Rui, Yong Yuan, Shuai Wang, and Fei-Yue Wang, "Economic issues in bitcoin mining and blockchain research," in *IEEE Intelligent Vehicles Symposium*, 2018.

[7] Conti, Mauro, E. Sandeep Kumar, Chhagan Lal, and Sushmita Ruj., "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, 2018, pp. 3416-3452.

[8] Liu, Yi, Xingtong Liu, Chaojing Tang, Jian Wang, and Lei Zhang. "Unlinkable coin mixing scheme for transaction privacy enhancement of bitcoin.", 2018, pp. 23261-23270.
[9] Bag, Samiran, Sushmita Ruj, and Kouichi Sakurai, "Bitcoin block withholding attack: Analysis and mitigation," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, 2016, pp. 1967–1978.
[10] Decker, Christian, and Roger Wattenhofer. "Information propagation in the bitcoin network." IEEE P2P 2013 Proceedings, 2013., pp. 1-10.
[11] Zhang, Shijie, and Jong-Hyouk Lee. "Double-spending with a Sybil attack in the Bitcoin decentralized network." IEEE Transactions on Industrial Informatics 15.10 (2019): pp. 5715-5722.
[12] Tschorsch, Florian, and Björn Scheuermann. "Bitcoin and beyond: A technical survey on decentralized digital currencies." IEEE Communications Surveys & Tutorials 18.3 (2016): pp. 2084-2123.
[13] Lin, Iuon-Chang, and Tzu-Chun Liao. "A Survey of Blockchain Security Issues and Challenges." IJ Network Security 19.5 (2017), pp. 653-659.
[14] Nguyen, Cong T., Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities," *IEEE Access*, vol. 7, 2019, pp. 85727-85745.
[15] Mohanta, Bhabendu Kumar, Debasish Jena, Soumyashree S. Panda, and Srichandan Sobhanayak, "Blockchain technology: A survey on applications and security privacy challenges," *Internet of Things* , vol. 8, 2019, pp. 100-107.
[16] Rosenfeld, Meni. "Analysis of hashrate-based double spending." arXiv preprint arXiv:1402.2009 (2014).
[17] Heilman, Ethan, "One weird trick to stop selfish miners: Fresh bitcoins, A solution for the honest miner (poster abstract)," in Financial Cryptography and Data Security - FC Workshops Christ Church, Barbdos, Mar 2014, pp. 161–162. [Online]. Available:https://doi.org/10.1007/978-3-662-44774-1_12 (access tim: 7-2-2021).
[18] R. Zhang and B. Preneel, "Publish or perish: A backward-compatible defense against selfish mining in bitcoin," in Topics in Cryptology – CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, 2017,pp. 277–292.
[19] Saad, Muhammad, Laurent Njilla, Charles Kamhoua, and Aziz Mohaisen, "Countering selfish mining in blockchains," in *nternational Conference on Computing, Networking and Communications (ICNC)*, 2019.
[20] S. Solat and M. Potop-Butucaru, "Brief announcement: Zeroblock: Timestamp-free prevention of block-withholding attack in bitcoin," in Stabilization, Safety, and Security of Distributed Systems: 19th International Symposium. Springer International Publishing, 2017, pp. 356–360.
[21] Liang G, Weller SR, Luo F, Zhao J, Dong ZY. Distributed blockchain-based data protection framework for modern power systems against cyber attacks. IEEE Transactions on Smart Grid. 2018 Mar 27,10(3), pp. 3162-73.
[22] Gilbert, Henri, and Helena Handschuh. "Security analysis of SHA-256 and sisters." International workshop on selected areas in cryptography, 2003.
[23] R. C. Merkle, "A digital signature based on a conventional encryption function," in Advances in Cryptology — CRYPTO '87: Proceedings. Springer Berlin Heidelberg, 1988, pp. 369–378.
[24] Alqassem, Israa, and Davor Svetinovic. "Towards reference architecture for cryptocurrencies: Bitcoin architectural analysis." 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom). IEEE, 2014.
[25] Eastlake, Donald, and Tony Hansen.US secure hash algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, May, 2011.
[26] Tomescu, Alin, and Srinivas Devadas, "Catena: Efficient non-equivocation via bitcoin," in *IEEE Symposium on Security and Privacy (SP)*, 2017.
[27] Gervais, Arthur, Ghassan O. Karame, Vedran Capkun, and Srdjan Capkun, "Is bitcoin a decentralized currency?," IEEE security & privacy, vol. 12, no. 3, pp. 54-60, 2014.