Taming Front-Running Attacks in Blockchains

***Abstract*—Blockchains add transactions to a distributed shared ledger by arriving at consensus on sets of transactions contained in blocks. This provides a total ordering on a set of global transactions. However, total ordering is not enough to satisfy application semantics under the Byzantine fault model. This is due to the fact that malicious miners and clients can collaborate to add their own transactions ahead of correct clients' transactions in order to gain application level and financial advantages. These attacks fall under the umbrella of front-running attacks. Therefore, total ordering is not strong enough to preserve application semantics. In this paper, we propose causality preserving total order as a solution to this problem. The resulting Blockchains will be stronger than traditional consensus based blockchains and will provide enhanced security ensuring correct application semantics in a Byzantine setting.**

***Index Terms*—Blockchain, Causal Order, Front-Running Attack, Security, Broadcast, Byzantine failure, Application Semantics, Consensus**

I. Introduction

Blockchain is a shared distributed ledger that provides a tamper-proof ordered sequence of records. Bitcoin [1] was the first blockchain that provided a solution to the *double-spending problem* and revolutionized electronic money transfer. Bitcoin solves Byzantine-tolerant consensus [2] via proof-of-work. This led to the development of further blockchains that solved consensus such as Ethereum [3] that go a step further and provide smart contracts [4] that allow the blockchain to act as a *universal computer*. Smart contracts are code hosted on the blockchain that provide operations to change the state of the blockchain. Since the code is tamper-proof, a set of parties can conduct business in a transparent manner on the blockchain. Further, smart contracts provide the capability to run classic centralized applications on the blockchain in a decentralized manner, such as auctions [5] and elections [6]. As more applications are designed for blockchain, an important question that arises is — does blockchain provide the required semantics for these applications? In the case of peer-to-peer money transfer, the answer is yes, because total order prevents double-spending. However, total order is not enough for a decentralized auction, because a Byzantine miner can collude with a Byzantine client by informing the client of its opponents' bids prior to them being added to the blockchain. This lack of enforcement of semantics provides an opportunity to Byzantine nodes to launch a variety of attacks known as front-running attacks [7], [8]. In this paper, we propose utilizing causal ordering protocols enforcing strong safety [9]–[11] to provide an enhanced level of security in the blockchain ecosystem. Our contributions are as follows:

1) We formalize front-running attacks and prove that they are a violation of causal ordering.
2) We prove that utilizing a causal ordering protocol will enforce application semantics and make the blockchain more secure and suitable for classic centralized applications.
3) We introduce a protocol to provide security against front-running attacks by providing a causality preserving total ordering across transactions recorded in the blockchain. We term the resulting blockchain as a strong blockchain since it provides stronger security guarantees and semantics compared to traditional blockchains.
4) We prove the correctness of our protocol and analyze its intrinsic fairness properties.

II. System Model

This paper models the set of miners as a distributed system having Byzantine processes which are processes that can misbehave [2], [12]. A correct process (miner) behaves exactly as specified by the blockchain protocol whereas a Byzantine process (miner) may exhibit arbitrary behaviour including crashing at any point during the execution. A Byzantine process cannot impersonate another process or spawn new processes. The distributed system is modelled as an undirected graph $G = (\mathcal{M}, H)$. Here $\mathcal{M}$ is the set of miners adding blocks to the blockchain. Let n be $|\mathcal{M}|$. H is the set of FIFO logical communication links over which miners communicate by message passing. G is a complete graph. This model is equivalent to the permissioned blockchain model [13]. Nonetheless, as can be seen by the proofs in this paper, our results hold for permissionless blockchains [1] as well. However, the solution we provide is geared towards permissioned blockchains.

The system is assumed to be synchronous, i.e., there is a known fixed upper bound δ on the message latency, and a known fixed upper bound ψ on the relative speeds of processors [14]. This is opposed to an asynchronous system, i.e., there is no upper bound δ on the message latency, nor any upper bound ψ on the relative speeds of processors [14]. Clients send their transactions to the system of miners by broadcasting protocol message containing the transaction to the system. This message contains all the required metadata for the transaction such as gas fees and the client's identity. Next, transactions sit at each miner's mempool [15], waiting to get added to the blockchain. Clients can also be Byzantine and collude with Byzantine miners and miners can also act as clients in the system. Our protocol assumes an upper bound on the number of Byzantine miners, t with $n \geq 3t + 2$. The number of Byzantine clients is assumed to be unbounded.

Definition 1. *The happens before relation $\rightarrow$ on messages consists of the following rules:*

1) *The set of messages delivered from any process p_i by a process is totally ordered by $\rightarrow$.*
2) *If p_i sent or delivered message m before sending message m', then $m \rightarrow m'$.*
3) *If $m \rightarrow m'$ and $m' \rightarrow m''$, then $m \rightarrow m''$.*

Let R denote the set of messages in the execution.

Definition 2. *The causal past of message m is denoted as $CP(m)$ and defined as the set of messages in R that causally precede message m under $\rightarrow$.*

The correctness of Byzantine causal order unicast/multicast/broadcast is specified on $(R, \rightarrow)$ for strong safety.

Definition 3. *A causal ordering algorithm for unicast/multicast/broadcast messages must ensure the following:*

1) ***Strong Safety:*** *$\forall m' \in CP(m)$ such that m' and m are sent to the same (correct) process(es), no correct process delivers m before m'.*
2) ***Liveness:*** *Each message sent by a correct process to another correct process will be eventually delivered.*

Definition 4. *A **transaction** is a string contained in messages broadcasted to the system of miners with the intention of being recorded in the blockchain via consensus. Transactions change the state of the blockchain by executing business logic between two or more clients.*

Definition 5. *The happens before relation $\rightarrow$ on transactions is defined as follows: Given messages m_1 and m_2 containing transactions t_1 and t_2 respectively, $t_1 \rightarrow t_2$ if and only if $m_1 \rightarrow m_2$.*

Definition 6. *The causal past of transaction t is denoted as $CP(t)$ and defined as the set of transactions that causally precede t under $\rightarrow$.*

Definition 7. *A causal ordering algorithm for blockchain BT must ensure the following:*

1) ***Strong Safety:*** *Given transaction t, $\forall t' \in CP(t)$, t' gets recorded in BT before t.*
2) ***Liveness:*** *Each transaction sent by a correct client eventually arrives in every correct miner's memory pool.*

Definition 8. *A **block** contains a sequence of totally ordered transactions and a hash of its parent block. A block is only added to the blockchain after the system of miners arrive at consensus on the contents of the block.*

Definition 9. ***Blockchain*** *is a distributed data structure consisting of a tree of blocks. Each block has only one parent (except the genesis block) and may have multiple children blocks.*

Definition 10. *Given a tree BT containing a blockchain, the **consensus chain** is a ordered sequence of blocks $B_0, B_1, ..., B_l$ such that B_k is the parent of B_{k+1} and $tree\_depth(BT) = n$.*

Definition 11. *BT is a **valid blockchain** if BT contains one and only one consensus chain.*

III. Front-Running Attacks

In this section we first present a broad family of attacks called front-running attacks. We formalize the attack and prove that it is essentially an attack on causal ordering. An important point to note is that front-running attacks are executed prior to execution of the blockchain consensus protocol. Miners can view unconfirmed transactions in their memory pools and broadcast their own transactions with higher transaction fees with the intention of executing front-running attacks on unsuspecting clients. Byzantine miners can also collude with Byzantine clients to execute front-running attacks. However, without loss of generality in our proofs and solutions, we assume that miners act as clients in executing attacks. The following are illustrative examples of front-running attacks on real-world applications:

1) An honest client process p_i sends transaction t_i to the network as part of a decentralized auction. A malicious miner M reads t_i, figures out the bid value p_i wants to place for an asset being auctioned and sends its own transaction t_M with the purpose of getting into the blockchain first. t_M would contain a higher bid price than p_i's bid price. This results in an unfair advantage for M in winning the auction.

2) An honest client process p_i sends a request to buy cryptocurrency (transaction t_1) at price x, where the market price y is less than x. A malicious miner M can aim to profit off this by adding two transactions to the block where it includes t_1:
    a) It adds t_0 buying cryptocurrency at price y from the market. t_0 is placed before t_1.
    b) It sells cryptocurrency to p_i in transaction t_2 to p_i (placed after t_1) at price x.

    This results in M making a profit by arbitraging off an honest client with a profit of $(y - x)$ per coin.

An honest miner should not look into the content of transactions in the network. Blockchains incentivize miners to go after transactions with higher mining fees to maximize profits. The blockchain protocol requires miners to be concerned with only transaction fees and not the contents of transactions. The Byzantine fault model encapsulates behaviour that does not follow specified protocol. Therefore, such malicious miners can be modeled as Byzantine processes and Byzantine fault-tolerant protocols can be utilized to prevent such behaviour.

Observation 1. *Miners executing front-running attacks are Byzantine.*

Front-running attacks are broadly categorized as follows [16]:

1) **Displacement Attack**: A Byzantine miner reads transaction t from its memory pool and broadcasts its own

transaction (copying contents of t) t' with higher transaction fees in order to record t' in the blockchain before t.

2) **Sandwich Attack**: A Byzantine miner reads transaction t_1 from its memory pool and broadcasts two transactions t_0 and t_2 with the intention of recording t_0 before t_1 and t_2 after t_1 in the same block. In this way, the Byzantine miner creates an arbitrage opportunity to make a profit.
3) **Suppression Attack**: A Byzantine miner reads transaction t from its memory pool and broadcasts a set of transactions T containing transactions with high transaction fees. This attack essentially forces t to not get recorded in the next block in the blockchain.

We now formally define front-running attacks and prove that they require causal ordering violation in order to occur. Since front-running attacks are executed before consensus and are harder to execute when no forks exist, for the sake of proofs we assume without loss of generality that they are executed on valid blockchains.

Definition 12. *A Byzantine miner executes a* ***front-running attack*** *by reading an unconfirmed transaction t_x and broadcasting/mining its own transaction t_y with the intention of recording t_y before t_x in the consensus chain of a valid blockchain BT.*

Theorem 1. *Front-running attacks are a violation of causal ordering.*

Proof. BT is a valid blockchain and $C = consensus\_chain(BT)$. Let p_i broadcast m_1 (containing transaction t_1) to the system of miners. Miner M delivers m_1 and adds t_1 to its memory pool. Miner M then broadcasts m_2 (containing t_2) with significantly higher transaction fees than m_1 (t_1), with the intention of adding t_2 to C before t_1 is added to it. If this attack succeeds, one of the following scenarios may play out:

1) Miner M' (M may be M') succeeds in adding the next block B containing t_2 to C. The new consensus chain of BT is $C' = C + B$. Eventually, t_1 gets added to $consensus\_chain(BT)$ as part of block B'. Since C' is a prefix of some future consensus chain, $consensus\_chain(BT)$, t_2 is ordered before t_1.
2) Miner M' (M may be M') succeeds in adding the next block B to C. B contains both t_1 and t_2 with t_2 ordered before t_1.

By Definition 12, this is a front-running attack on t_1 by M via t_2. In order to execute this attack, M delivered m_1 and broadcasted m_2. By the message order rule in Definition 1, $m_1 \rightarrow m_2$. Since t_2 is recorded in C before t_1, the contents of m_2 are consumed by the system before the contents of m_1 resulting in a strong safety violation as per Definition 3. Therefore, it is clear that a front-running attack across transactions requires a causality violation across their respective protocol messages. □

IV. Background

A. Some Cryptographic Basics

We utilize non-interactive threshold cryptography as a means to guarantee strong safety of multicasts [17]. Threshold cryptography consists of an initialization function to generate keys, message encryption, sharing decrypted shares of the message and finally combining the decrypted shares to obtain the original message from ciphertext. The following functions are used in a threshold cryptographic scheme:

Definition 13. *The dealer executes the generate() function to obtain the public key PK, verification key VK and the private keys SK_0, SK_1, ... , SK_n.*

The dealer shares private key SK_i with each process p_i while PK and VK are publicly available.

Definition 14. *When process p_i wants to send a message m to p_j, it executes $E(PK, m, L)$ to obtain C_m. Here C_m is the ciphertext corresponding to m, E is the encryption algorithm and L is a label to identify m. p_i then broadcasts C_m to the system of processes.*

Definition 15. *When process p_l receives ciphertext C_m, it executes $D(SK_l, C_m)$ to obtain σ_l^m where D is the decryption share generation algorithm and σ_l^m is p_l's decryption share for message m.*

When process p_j receives a cipher message C_m intended for it, it has to wait for k decryption shares to arrive from the system to obtain m. The value of k depends on the security properties of the system. It derives the message from the ciphertext as follows:

Definition 16. *When process p_j wants to generate the original message m from ciphertext C_m, it executes $C(VK, C_m, S)$ where S is a set of k decryption shares for m and C is the combining algorithm for the k decryption shares.*

The following function is used to verify the authenticity of a decryption share:

Definition 17. *When a decryption share σ is received for message m, the Share Verification Algorithm is used to ascertain whether σ is authentic : $V(VK, C_m, \sigma) = 1$ if σ is authentic, $V(VK, C_m, \sigma) = 0$ if σ is not authentic.*

B. Byzantine Causal Broadcast via Byzantine Reliable Broadcast

We propose a causal order broadcast algorithm for clients to send transactions to miners. Byzantine-tolerant causal broadcast is invoked as BC_broadcast(m) and delivers a message through BC_deliver(m).

Definition 18. *Byzantine Causal Multicast satisfies the following properties:*

1) *(BCB-Validity:) If a correct process p_i* BC_deliver*s message m from $sender(m)$ then $sender(m)$ must have* BC_broadcast *m.*

2) *(BCB-Termination-1:) If a correct process* BC_broadcast*s a message m then some correct process eventually* BC_deliver*s m.*
3) *(BCB-Agreement or BCM-Termination-2:) If a correct process* BC_deliver*s a message m from a possibly faulty process, then all correct processes eventually deliver m.*
4) *(BCB-Integrity:) For any message m, every correct process p_i* BC_deliver*s m at most once.*
5) *(BCB-Causal-Order:) If $m \rightarrow m'$, then no correct process* BC_deliver*s m' before m.*

BCB-Causal-Order is the Strong Safety property of Definition 3. BCB-Termination-1 and BCB-Agreement imply the liveness property of Definitions 3.

The Byzantine-tolerant Reliable Broadcast (BRB) [18], [19] is invoked by BR_broadcast and its message is delivered by BR_deliver, and satisfies the properties given below.

Definition 19. *Byzantine-tolerant Reliable Broadcast (BRB) provides the following guarantees [18], [19]:*

1) *(BRB-Validity:) If a correct process* BR_deliver*s a message m from $sender(m)$, then $sender(m)$ must have* BR_broadcast *m.*
2) *(BRB-Termination-1:) If a correct process* BR_broadcast*s a message m, then some correct process eventually* BR_deliver*s m.*
3) *(BRB-Agreement or BRB-Termination-2:) If a correct process* BR_deliver*s a message m from a possibly faulty process, then all correct processes eventually* BR_deliver *m.*
4) *(BRB-Integrity:) For any message m, every correct process* BR_deliver*s m at most once.*

V. Causal Ordering Protocol to Prevent Front-Running Attacks

In light of the result of Theorem 1, we present a causality preserving blockchain protocol to strengthen the security of blockchain to withstand front-running attacks under the synchronous system setting. A synchronous system can assume lock-step execution in rounds. Within a round, a process can send messages, then receive messages, and lastly have internal events; further a message sent in a round is received in the same round at all its destinations. Algorithm 1 serves as a reference point for synchronous round-based communication. Without loss of generality, we assume that all processes send their messages at the beginning of each round, all messages arrive in the same round that they are sent out and messages are delivered at the end of each round. Threshold cryptography in conjunction with the execution in rounds and Byzantine Reliable Broadcast are used to ensure strong safety + liveness. Clients broadcast transactions to the system of miners encapsulated in protocol messages via BRB. Using BRB protects against liveness attacks by Byzantine clients via *BRB-Termination-1* and *BRB-Termination-2*.

We present our solution, called *causality preserving blockchain protocol* in Algorithm 2. The blockchain consensus protocol generates a total ordering of transactions. Our protocol guarantees a stronger property, *causally consistent total ordering* of transactions. Algorithm 2 not only provides a total ordering of blockchain transactions, but also ensures that this total ordering does not violate causality across transactions, hence ensuring application semantics in a Byzantine setting. This is why we term any blockchain following our protocol as a stronger blockchain than classic blockchains. For simplicity, we term this as a *strong blockchain* and is defined below.

Algorithm 1: Synchronous round-based message passing protocol

```
Data: Each process locally maintains two FIFO queues $Q_s$
      and $Q_d$ for storing outgoing/incoming messages
      respectively
1 when round $r$ starts:
2     broadcast all messages in FIFO order after dequeuing
        from $Q_s$
3 when round $r$ ends:
4     deliver all messages in FIFO order after dequeuing from
        $Q_d$
5 when the application is ready to broadcast message $m$:
6     $Q_s.enqueue(m)$
7 when message $m$ arrives:
8     $Q_d.enqueue(m)$
```

Definition 20. *A **strong blockchain** BT must satisfy the following properties:*

1) *BT is a valid blockchain*
2) *$\forall t_1, t_2$ such that $t_1 \rightarrow t_2$, t_1 is recorded before t_2 in BT's consensus chain.*

Algorithm 2 does not change the consensus mechanism and is therefore independent of the consensus protocol employed by the blockchain. Instead, Algorithm 2 ensures that only transactions whose causal past is already in the consensus chain are allowed to be mined. In other words, we have added a clause to determine whether a transaction is valid.

In addition to blockchain-specific properties that need to be satisfied (e.g., sufficient balance, identity of client), a transaction is not considered valid if it causally depends on one or more transactions that have not been finalized in the blockchain. We call valid transactions as safe transactions, formalized below:

Definition 21. *Transaction t is an **safe transaction** with regards to a valid blockchain BT if and only if $\forall t' \in CP(t), \exists B \in consensus\_chain(BT)$ such that $t' \in B$.*

Algorithm 2 only considers *safe blocks* for consensus, preventing front-running attacks by maintaining causal relations across transactions. Definition 22 formalizes a safe block.

Definition 22. *A **safe block** B only contains valid transactions, or in case any invalid transaction $t \in B$, then $\forall t' \in CP(t)$, t' must precede t in B.*

Algorithm 2 provides the BC_broadcast primitive to clients to protect against front-running attacks and BC_deliver

Algorithm 2: Causality Preserving Blockchain Protocol

Data: Each client process p_{c_i} has access to PK (global public key), each miner process p_m has access to VK (global verification key). Each miner p_{M_i} has access to a local secret key SK_i. Each client uses a FIFO queue Q_s for outgoing protocol messages. Each miner p_{M_i}'s memory pool is denoted by a set MP. The causal past of transaction t is denoted as $CP(t)$. The set of all miners is $\mathcal{M}$. BT is the shared blockchain. B_r^i is the block proposed by miner p_{M_i} in round $r+1$.

```
when round $r$ starts at client $p_{c_i}$:
while $Q_s.head() \neq \phi$ do
    $C_m = Q_s.pop()$
    BR_broadcast($C_m, M$)

when client $p_{c_i}$ sends $m$ to $M$ via BC_broadcast($m, M$) in round $r$:
$C_m = E(PK, m, id_m)$
$Q_s.push(C_m)$

when round $r$ starts at miner $p_{M_i}$:
$B = consensus(candidate\_set)$ ;    ▷ consensus on the set of blocks delivered in the previous round
$candidate\_set = \phi$
Add $B$ at the end of $consensus\_chain(BT)$
for all $t \in B$ do
    delete $t$ from $MP$
    for all $t'$ such that $t \in CP(t')$ do
        $CP(t') = CP(t') \setminus t$
$B_r^i = \phi$
for all $t$ in $MP$ such that $t$ is semantically invalid do
    delete $t$
for all $t$ in $MP'$ where $MP' \subseteq MP \wedge CP(t) = \phi$ do
    $B_r^i = B_r^i \cup \{t\}$ ;    ▷ Block construction with safe transactions
for all $p_{m_j} \in \mathcal{M}$ do
    send $B_r^i$ to $p_{m_j}$

when $B_r^j$ arrives at miner $p_{M_i}$ during round $r$: ;    ▷ Block created by miner $p_{m_j}$ in round $r$ and proposed for consensus in round $(r+1)$
for all $t \in B_r^j$ do
    if $t$ is semantically invalid $\vee$ $CP(t) \neq \phi$ then
        discard $B_r^j$
if $B_r^j$ has not been discarded then
    $candidate\_set = candidate\_set \cup B_r^j$ ;    ▷ all safe blocks arriving in round r are added to candidate_set

when $C_m$ is BR_delivered at miner $p_{M_i}$ in round $r$:
$\sigma_i^m = D(SK_i, C_m)$
for all $p_{m_j} \in M$ do
    send $\sigma_i^m$ to $p_j$ in round $(r+1)$

when miner $p_{M_i}$ receives $(2t+1)$th valid $\langle \sigma_x^m \rangle$ message by round $r$:
Store $(t+1)$ decryption shares in set $S$
$m = C(VK, C_m, S)$
extract $t_m$ from $m$ ;    ▷ bc_delivery(m)
$CP(t_m) = MP$
$MP = MP \cup \{t_m\}$
```

to miners for extracting transactions from messages. BR_broadcast and BR_delivery are the underlying primitives implementing Byzantine reliable broadcast (BRB) [18], [19]. Let β and γ denote the maximum and minimum number of rounds (sequential steps) respectively in a BRB protocol. For example, Bracha's BRB has $\beta = 4$,$\gamma = 3$ and requires $n > 3t$ [18], [19] whereas Imbs-Raynal [20] has $\beta = 3$,$\gamma = 2$ and requires $n > 5t$. A BR_broadcast sent in round r is delivered as BR_deliver by round $r+\beta-1$. Although a message m sent in a round is delivered after all messages sent in previous rounds, a Byzantine miner can peek into m before its transaction is committed to the blockchain and send a causally dependent message m' in the same round to initiate a broadcast send via its own BR_broadcast. m' may be BR_delivered in the same round as m at some miners, thus leading to a potential front-running attack across the transactions contained in m and m'. To prevent a Byzantine process from peeking into the transaction of a message during the β rounds, the message is encrypted using threshold encryption. In round $r+\beta$, each process that has BR_delivered the encrypted message sends its decryption share to the destinations of the multicast. A message gets revealed only at the end of round $r+\beta$. Any message sent before that cannot be causally dependent on this revealed message m, and the only messages that the process sends that are causally dependent on the above message m can get sent (and hence delivered) only in later rounds. This straightforwardly guarantees strong safety and liveness. The $\beta+1$ rounds $a\times(\beta+1)+1$, $a\times(\beta+1)+2$, $\ldots$ $a\times(\beta+1)+k$, $a\times(\beta+1)+\beta+1$ constitute a meta-round a for $a \geq 0$. Thus, rounds $r, r+1, \ldots r+\beta$, such that $r\,div\,(\beta+1) = a$ and $r \bmod (\beta+1) = 0$ constitute meta-round a. The first β rounds of a meta-round are for BRB and the $\beta+1$th round is for sending the decryption shares to the system of miners.

Algorithm 2 consists of both miner side code and client side code divided into six *when blocks*, each in reaction to an event in the protocol. The *when block* from lines 1-4 is executed in the beginning of a round, with each client broadcasting messages using BRB it created in the previous round in a FIFO manner from a local queue containing those messages. FIFO ordering at the client in conjunction with FIFO channels ensures source order at the miners' end. The *when block* in lines 5-7 describes how clients utilize the BC_broadcast primitive provided by Algorithm 2. Clients encrypt messages using threshold cryptography and enqueue them in a local FIFO queue, ready to be sent out in the beginning of the next round. The *when block* between lines 8-22 is executed by each miner in the beginning of a round. In line 9, miners arrive at consensus on the set of blocks proposed by each miner in the previous round. These blocks are stored in a set $candidate\_set$. Miners then clear $candidate\_set$, ready to store blocks in the current round and the consensus block B is added to the blockchain. Lines 12-15 clear transactions contained in B from the miners' memory pool, MP (a set data structure containing transactions waiting to be added to the blockchain) and the causal past ($CP(t)$ keeps track of transactions in MP that need to be added to the blockchain before t) of the remaining transactions in MP. Next, the miner constructs its own block (to be sent out for consensus in the next round) with semantically valid and safe transactions (lines 16-20). In lines 21-22, miners send their blocks for consensus in the next round. The *when block* in lines 23-28 deals with incoming blocks from other miners for which consensus will be arrived at in the next round. When a miner receives a block, it checks if the block is semantically valid and makes sure all transactions in the block do not have causal dependencies on existing transactions in the memory pool. If that is the case, the block is added to $candidate\_set$. The *when block* in lines 29-32 deals with miners receiving a message (containing a transaction) from a client via BR_delivery, computing their decryption shares for the message and broadcasting the decryption share in the next round. Finally, the *when block* in lines 33-38 deals with miners receiving the required number of decryption shares ($t+1$) for decrypting a protocol message. The miners decrypt the message m in line 36 and extract the transaction t_m in line 37 (this line is BC_delivery) and store the causal past of t_m in $CP(t_m)$. Finally, t_m is added to the memory pool in line 39. For the purposes of this protocol, $CP(t)$ is treated as a dynamic set data structure, which starts off containing the entire set of transactions in the causal past of t. As each of these transactions are added to blockchain BT, they are removed from $CP(t)$. Once $CP(t) = \phi$, t is a safe transaction and is ready to be added to the blockchain.

Lemma 1. *Memory pool, MP, will be the same at all correct miners following Algorithm 2 at the end of every round.*

Proof. All $(2t+1)$ correct miners' decryption shares are required to decrypt any C_m (ciphertext of message m) that has been BR_delivered as seen in line 33. Therefore, no miner can see transaction t_m (contained in m) until the last miner to BR_deliver m sends its decryption share in lines 29-32. If the last miner to broadcast its decryption share for C_m does so in round r, all correct miners will BC_deliver m and add t_m to their mining pools in round r (lines 32-38). Therefore, at the end of round r, MP will be the same at all correct miners. □

Theorem 2. *For all transactions t_1 and t_2 in a valid blockchain BT, such that $t_1 \rightarrow t_2$, Algorithm 2 guarantees that t_1 is ordered before t_2 in BT's consensus chain.*

Proof. Consider messages m_1 and m_2 containing transactions t_1 and t_2 respectively, with $m_1 \rightarrow m_2$. From Definition 5, $t_1 \rightarrow t_2$. Let p_{m_j} (possibly Byzantine) be the sender of m_2. p_{m_j} BC_delivers m_1 and views t_1 in line 37 of Algorithm 2. The earliest that m_2 can be broadcasted to the system is in round r itself (this is Byzantine behaviour, a correct miner would broadcast m_2 in round $(r+1)$). The fastest delivery time of m_2 at any miner would be the minimum latency of BRB (γ) + decryption share latency (1 round) + sending round (r). Therefore, the earliest m_2 can be delivered at any miner is at round $r_{m_2} = (r+\gamma+1)$. The latest delivery of m_1 at any miner would be round $r_{m_1} = (r+\beta-\gamma)$. For any BRB protocol, $\beta < 2\gamma$, therefore we can derive the following by replacing β with 2γ in the equation for r_{m_1}:

$$r_{m_1} < (r + \gamma)$$
$$r_{m_1} < (r + \gamma + 1)$$
$$r_{m_1} < r_{m_2}$$

Since $r_{m_1} < r_{m_2}$, m_1 will be BC_delivered before m_2 and t_1 will be in the memory pool (MP) prior to the extraction of t_2 (lines 33-38). Therefore, when t_2 will be included in MP at all correct miners, $CP(t_2)$ will include t_1 (lines 36-38). From Lemma 1, $t_1 \in CP(t_2)$ at all correct miners (line 37). Any block B containing t_2 will be rejected by correct miners if t_1 is not recorded in blockchain BT (lines 23-26). Consequently, no such block B can be added to the blockchain until t_1 is recorded in BT. Therefore, given $t_1 \rightarrow t_2$, t_1 will be recorded in BT prior to t_2. □

Theorem 3. *All transactions broadcasted to blockchain BT via Algorithm 2 will be added to each correct miner's memory pool MP within bounded time.*

Proof. Let client p_{c_i} send message m (containing transaction t_m) to BT via Algorithm 2 in round r. p_{c_i} sends m's ciphertext C_m via BRB in lines 1-4 to the system of miners. By BRB-Termination-1 and BRB-Termination-2 from Definition 19, it can be seen that all correct miners will BR_deliver C_m within β rounds at line 29 and broadcast their respective decryption shares in the next round in lines 30-32. In the next round all correct processes will receive the required number of decryption shares to decrypt C_m in line 33. All correct miners will proceed to decrypt message m and store its transaction t_m in MP in the same round (lines 34-38). Therefore, a transaction t_m sent in round r via Algorithm 2 will arrive at every correct miner's memory pool during or before round $(r + \beta + 1)$. □

Corollary 1. *Algorithm 2 guarantees causal ordering as defined in Definition 7.*

Theorem 4. *Any blockchain constructed by Algorithm 2 is resilient to front-running attacks.*

Proof. Follows from Theorem 1 and Corollary 1. □

A critical observation about Algorithm 2 is that any transaction t BC_delivered in round r will be added to the causal past of every transaction BC_delivered in rounds $(r+1), (r+2), ...(r+k)$, where $(r+k)$ is the round where t is recorded to the blockchain. Consequently, any transaction t' added to the memory pool after t cannot be added to the blockchain until t is added to it. This forces miners to mine and add existing transactions in the memory pool to the blockchain in order to ensure that future transactions do not end up waiting in the memory pool, thereby preventing wastage of both resources and time. This leads us to Observation 2.

Observation 2. *Any blockchain constructed by Algorithm 2 guarantees intrinsic fairness to clients.*

VI. Discussion

Front-running attacks. In this paper we studied front-running attacks and proved that all front-running attacks are causal ordering violations accross transactions. The reason that front-running attacks are feasible against existing blockchains is because blockchains provide a *total ordering* of transactions by solving Byzantine-tolerant consensus but do not preserve causality when building this total-ordering. We conclude that solving consensus is not enough from an application semantics perspective in a Byzantine environment.

Stronger Blockchains. In light of our findings, we defined the notion of a *strong blockchain*, which is a blockchain that provides a causality-preserving total-order across transactions. This eliminates the feasibility of front-running attacks by Byzantine processes and guarantees application semantics. We proposed a causal ordering protocol to be used in conjunction with the consensus protocol to build a strong blockchain. This approach is modular because it does not interfere with the consensus protocol of the blockchain. Instead, the causal ordering protocol on transactions runs prior to the consensus protocol on blocks of transactions. That is, the causal ordering protocol ensures that transactions added to blocks do not have causal dependencies in the memory pool. This makes it straightforward to incorporate causal ordering as a pre-consensus protocol to existing blockchains. Our blockchain protocol keeps track of causal dependencies of every transaction added to the memory pool of every miner. BRB ensures that all correct miners have correct knowledge of the causal dependencies. This allows our protocol to stop any transactions from being mined whose causal dependencies have not been added to the blockchain. It only allows blocks containing *safe transactions* to be candidates for consensus.

Related Work. Recently, a technique to make sandwich attacks unprofitable to *rational* Byzantine processes in the permissionless setting was proposed [21]. This technique involves changing the blockchain protocol itself by making random reorderings of transactions within proposed blocks. Fair ordering of transactions at the consensus level has been formalized in [22], [23]. However, this approach does not completely rule out front-running attacks. Our protocol uses threshold cryptography, which has previously been used in a probabilistic algorithm based on atomic (total order) broadcast for secure causal atomic broadcast (liveness and strong safety) in an asynchronous system [24]. This algorithm used acknowledgements and effectively processed the atomic broadcasts serially. This protocol would force miners to see transactions in a total order inhibiting parallel mining of transactions sent concurrently. Additionally, this protocol in conjunction with blockchain would solve consensus twice, wasting time and resources. More recently, threshold cryptography has been used to develop a non-deterministic multicast algorithm for causal ordering in asynchronous systems [11].

Causality Preserving Blockchain Protocol. We proposed a strong blockchain protocol and proved its correctness in this paper. Our protocol provides *deterministic* causal ordering in a synchronous communication model. Since our protocol operates in a synchronous setting, the consensus protocol will also be deterministic. Our protocol assumes that there are $(3t+1)$ miners out of which at most $(t-1)$ can be Byzantine [1]. This means that this protocol is suited for a permissioned blockchain, with a static number of miners. Our protocol has a message complexity of $O(n^2)$ and has an upper bound on latency (time for a transaction to arrive in all correct miners' memory pools) of $(\gamma+1)$ rounds.

Conclusion. Our result in Theorem 1, stating that front-running attacks are causal violations is independent of the system model of our protocol. Therefore, front-running attacks are not feasible against our notion of a strong blockchain regardless of the system model assumptions (permissioned vs. non-permissioned, synchrony vs. asynchrony). Future work comprises developing protocols for strong blockchain in different system settings such as non-permissioned blockchains and blockchains with asynchronous communication. Although our solution is deterministic, it is not possible to develop a deterministic strong blockchain in an asynchronous system [10]. This paper established that causal ordering is critical for blockchain security and maintaining application semantics and provided a causal ordering solution for synchronous permissioned blockchains. To the best of our knowledge, this is the first work that addressed the *root cause* that makes front-running attacks possible, and proved that front-running attacks are causal violations. Additionally, we provided a solution that can be adopted by existing blockchains *without interfering* with the blockchain protocol.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[2] L. Lamport, R. E. Shostak, and M. C. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[3] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[4] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 2017, pp. 303–312.

[5] A. Hahn, R. Singh, C.-C. Liu, and S. Chen, "Smart contract-based campus demonstration of decentralized transactive energy auctions," in *2017 IEEE Power & energy society innovative smart grid technologies conference (ISGT)*. IEEE, 2017, pp. 1–5.

[6] R. Hanifatunnisa and B. Rahardjo, "Blockchain based e-voting recording system design," in *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*. IEEE, 2017, pp. 1–6.

[7] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," in *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer, 2020, pp. 170–189.

[8] P. Züst, T. Nadahalli, and Y. W. R. Wattenhofer, "Analyzing and preventing sandwich attacks in ethereum," *ETH Zürich*, 2021.

[9] A. Misra and A. D. Kshemkalyani, "Solvability of byzantine fault-tolerant causal ordering problems," in *International Conference on Networked Systems*. Springer, 2022, pp. 87–103.

[10] ——, "Byzantine fault-tolerant causal ordering," in *Proceedings of the 24th International Conference on Distributed Computing and Networking*, 2023, pp. 100–109.

[11] ——, "Byzantine fault-tolerant causal order satisfying strong safety," in *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*. Springer, 2023, pp. 111–125.

[12] M. C. Pease, R. E. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[13] J. Polge, J. Robert, and Y. Le Traon, "Permissioned blockchain frameworks in the industry: A comparison," *Ict Express*, vol. 7, no. 2, pp. 229–233, 2021.

[14] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[15] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, "Exploring the attack surface of blockchain: A systematic overview," *arXiv preprint arXiv:1904.03487*, 2019.

[16] C. F. Torres, R. Camino *et al.*, "Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1343–1359.

[17] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *Journal of Cryptology*, vol. 15, no. 2, pp. 75–96, 2002.

[18] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[19] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, 1985.

[20] D. Imbs and M. Raynal, "Trading off t-resilience for efficiency in asynchronous byzantine reliable broadcast," *Parallel Processing Letters*, vol. 26, no. 04, p. 1650017, 2016.

[21] O. Alpos, I. Amores-Sesar, C. Cachin, and M. Yeo, "Eating sandwiches: Modular and lightweight elimination of transaction reordering attacks," *arXiv preprint arXiv:2307.02954*, 2023.

[22] M. Kelkar, S. Deb, and S. Kannan, "Order-fair consensus in the permissionless setting," in *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, 2022, pp. 3–14.

[23] C. Cachin, J. Mićić, N. Steinhauer, and L. Zanolini, "Quick order fairness," in *International Conference on Financial Cryptography and Data Security*. Springer, 2022, pp. 316–333.

[24] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[1]BRB requires an upper bound of t Byzantine processes out of $(3t+1)$ processes. In our case, the client becomes the $(3t+2)^{th}$ process in the system when broadcasting to the system of miners via BRB. In case the broadcasting client is Byzantine, correctness of the protocol can only be guaranteed when at most $(t-1)$ miners are Byzantine.