Revisiting Transaction Ledger Robustness in the Miner Extractable Value Era

Fredrik Kamphuis[1(✉)], Bernardo Magri[2], Ricky Lamberty[1], and Sebastian Faust[3]

[1] Corporate Research, Robert Bosch GmbH, Renningen, Germany
{fredrik.kamphuis,ricky.lamberty}@de.bosch.com

[2] The University of Manchester, Manchester, UK
bernardo.magri@manchester.ac.uk

[3] Technical University of Darmstadt, Darmstadt, Germany
sebastian.faust@cs.tu-darmstadt.de

Abstract. In public transaction ledgers such as Bitcoin and Ethereum, it is generally assumed that miners do not have any preference on the contents of the transactions they include, such that miners eventually include all transactions they receive. However, Daian *et al.* S&P'20 showed that in practice this is not the case, and the so called *miner extractable value* can dramatically increase miners' profit by re-ordering, delaying or even suppressing transactions. Consequently an "unpopular" transaction might never be included in the ledger if miners decide to *suppress* it, making, e.g., the standard liveness property of transaction ledgers (Garay *et al.* Eurocrypt'15) impossible to be guaranteed in this setting.

In this work, we formally define the setting where miners of a transaction ledger are dictatorial, i.e., their transaction selection and ordering process is driven by their individual preferences on the transaction's contents. To this end, we integrate dictatorial miners into the transaction ledger model of Garay *et al.* by replacing honest miners with dictatorial ones. Next, we introduce a new property for a transaction ledger protocol that we call *content preference robustness* (CPR). This property ensures *rational liveness*, which guarantees inclusion of transactions even when miners are dictatorial, and it provides *rational transaction order preservation* which ensures that no dictatorial miner can improve its utility by altering the order of received candidate transactions. We show that a transaction ledger protocol can achieve CPR if miners cannot obtain a-priori knowledge of the content of the transactions. Finally, we provide a generic compiler based on time-lock puzzles that transforms any robust transaction ledger protocol into a CPR ledger.

Keywords: blockchain · liveness · rational security

1 Introduction

In distributed transaction ledgers such as Bitcoin [26] and Ethereum [8], transactions proposed by users are verified in a decentralized way and appended into a public ledger in an unalterable order. To this end, a set of network participants called *miners* are responsible for the process of including and finalizing transactions by running the consensus protocol. The *liveness* property of a consensus protocol guarantees that a correctly generated transaction that is provided as input to all the miners will eventually appear on the ledger. It has been formally shown that this guarantee is achieved under the assumption of honest majority (or supermajority) of miners [3,16,25]. Therefore, it is commonly assumed that honest miners input all the transactions to the consensus protocol in the exact same order they were received. In practice, transaction ledger protocols usually establish incentive mechanisms (e.g., in the form of transaction fees) to justify this fundamental assumption. A transaction ledger protocol therefore aims to achieve self-enforced *honest behavior* by incentivizing parties to behave honestly and penalizing deviation of the desired protocol behavior [4,21]. These mechanisms yield profit for miners, e.g. for honestly including and appending transactions into blocks. Nevertheless, when analyzing the incentive compatibleness of *honest behavior*, the approach usually taken [2] is based on the assumption that miners do not have any rational interest in the actual content of the transactions they include. As it turns out, this assumption cannot be guaranteed in practice. In fact, there are many works that show that rational miners indeed profit from altering the order or even ignoring certain transactions entirely [10,15,30,31]. While forking on the underlying ledger to revert transactions could theoretically lead to the same results, it requires at least 1/3rd or the majority of the resources of the network (i.e., computation or stake) to be executed [7,22,24]. Additionally, such an attack could lead to unforeseeable dynamics that might be undesirable for rational adversaries [2,7]. Moreover, reordering, delaying or suppressing single transactions during the inclusion in the block is a much more subtle deviation from the honest behavior compared to forking, and more importantly can be accomplished by any individual miner. Therefore, this type of behavior might be considered as a viable and practical strategy by rational miners [10,27].

Daian *et al.* [10] generalizes this concept of content-depending utilities as *miner extractable value* (MEV). MEV is a metric representing all kinds of opportunities a rational miner can generate utility permissionlessly from e.g., re-ordering, delaying or censoring of transactions depending on the transaction's content. [10] shows that MEV is not just a theoretical concept, but rather a real-world phenomenum that is already occurring at scale in today's DeFi[1] space, e.g. in the form of transaction front-running, reaching its current spike with roughly 25000 ETH (4.1 million USD) available for arbitrage daily[2]. Overall, the actual content preference of rational miners depends on on-chain dynamics within the transaction ledger protocol, such as arbitrage opportunities, front running and censorship, but also on utility sources outside the ledger. This might include cross-chain dynamics where miners can generate profit on other chains by tak-

[1] The term *decentralized finance* (DeFi) refers to an alternative financial infrastructure, that is built on top of open and permissionless protocols, such as the Ethereum blockchain.

[2] Flashbots. http://explore.flashbots.net/, as of July 09, 2022.

ing rational actions on their chain [24], but also off-chain dynamics where the profit for taking rational actions is generated entirely off-chain [7]. Therefore, the actual individual utility a rational miner can generate for taking actions against candidate transactions is unknown to the public and the protocol designer.

Further, as illustrated by Daian *et al.* unclaimed MEV opportunities in a branch of the blockchain can incentivize miners to support that branch and abandon the longest chain, thus "rolling back" blocks and creating potential forks. This harms the network and even poses a fundamental threat to the security of the underlying consensus protocol.

To summarize, the inherent issue is that miners can use their a-priori knowledge of the transaction content to alter the set of transactions they are supposed to include. This information asymmetry provides miners with an advantage by taking rational actions (e.g., altering order, delaying, suppressing) which may be rewarded disproportionately in comparison to honest behavior [10,15]. Moreover, these rational actions are not a violation of the underlying transaction ledger protocol, and thus not captured by existing security definitions. The miners that are willing to take rational actions based on the transactions' contents are referred to as *dictatorial* miners.

Due to space limitations the related work section is presented in Appendix A.

1.1 Contributions

In this work we analyze the liveness and transaction order guarantee that can be achieved against dictatorial miners. In this vein, we follow the Bitcoin backbone model of Garay, Kiayias and Leonardos [16] with the twist that instead of honest miners we assume a set of dictatorial miners with hidden[3] preferences over the contents of transactions. The dictatorial miners participate honestly in the consensus protocol but may choose to take rational actions that do not violate the properties of the ledger protocol, e.g. re-order, delay or suppress a particular set of transactions. In addition, dictatorial miners are allowed to collude with each other if it is (individually) more profitable. Our contributions can be summarized as follows:

- We introduce a new property for transaction ledger protocols that we call *content preference robustness* (CPR), which yields robustness against dictatorial miners. A CPR-ledger provides the following guarantees:
  - *Rational liveness:* It provides essentially the same guarantees as the original liveness definition [16], but against dictatorial miners. To achieve this we show that no dictatorial miner can increase its profit by selectively suppressing transactions if (1) dictatorial miners gain no a-priori knowledge of the transactions contents, (2) withholding transactions is punishable by the ledger and (3) dictatorial miners expect to not lose utility (on average) by honestly participating in the transaction ledger protocol.

[3] We call *hidden* the preferences that are individual to the miners, and not known to the protocol designer.

  - *Rational transaction order preservation:* It ensures that no dictatorial miner can improve its expected utility by altering the order of received transactions. We show that rational transaction order preservation can be achieved under essentially the same conditions as rational liveness.
- We present a generic compiler based on time-lock puzzles that compiles any robust transaction ledger protocol (according to [16]) into a transaction ledger protocol that is CPR. On a technical level, the compiled protocol maintains two (logically) separate chains; the "control chain" that contains only time-lock puzzles of the transactions, and the "sanitized chain" that contains the actual contents of the transactions from (the common prefix of) the control chain. The intuition is that the control chain provides a global ordering of the transactions for all miners, and once that ordering is fixed, the sanitized chain can be built with the actual contents of all valid transactions from the control chain. Finally, we show that the compiled protocol is CPR.

2 Transaction Ledger Model

In this work we extend the transaction ledger model of Garay *et al.* [16] to the setting where dictatorial miners may use their a-priori knowledge of the transaction content in order to generate MEV. To this end, we first state the fundamentals of the transaction ledger model by Garay *et al.* [16] and then continue to explain how we adapt their model.

2.1 Ledger Backbone Model

According to Garay *et al.* [16] a transaction ledger is represented as a vector of blocks $l = (\mathcal{B}_1, ..., \mathcal{B}_d)$, where each block $\mathcal{B}_i = (tx_1, ..., tx_n)$ is a vector of transactions $tx \in \mathcal{T}$. $\mathcal{T}$ denotes the set of valid transactions. Appending a transaction tx to a vector l is denoted by $l||tx$. Also, appending a vector of transactions $\mathcal{B}$ to another vector l is denoted as $l||\mathcal{B}$. $tx_{i,j}$ denotes transaction tx_j in block $\mathcal{B}_i$. As a ledger is a vector of transactions, we simply denote it as $l = (tx_1, ..., tx_m)$ omitting the block numbers when clear from the context.

The transaction ledger protocol is executed by a set of miners $\mathcal{M}$ in the presence of a PPT adversary $\mathcal{S}$, and driven by a PPT environment $\mathcal{Z}$. Each honest miner M_i maintains its own local copy of the chain l_i. Further, an honest miner M_i process a local buffer $\mathsf{X}_i := (tx_1, \ldots, tx_e)$, that are candidate transactions to be incorporated into the ledger l_i provided by the environment $\mathcal{Z}$.[4] In [16], a transaction ledger protocol is defined by the transaction generation oracle TxGen that issues transactions on behalf of the users and the set of valid ledgers $\mathcal{L}$. Upon receiving a message $(\mathsf{IssueTx}, \gamma, P)$, TxGen generates a unique transaction $tx[\gamma] \in \mathcal{T}$ on behalf of P, where $tx[\gamma]$ denotes a transaction tx that contains an encoding of content γ. Further, a ledger is defined by the three interface functions $\mathsf{V}(\cdot), \mathsf{I}(\cdot), \mathsf{R}(\cdot)$. Where $\mathsf{V}(\cdot)$ is the chain validity function, $I(\cdot)$ is the

[4] Note that honest miners operate X_i as provided by the environment and do not change any ordering to maximize fees.

input contribution function that is executed to provide new blocks, and $\mathsf{R}(\cdot)$ is the chain reading function that returns a semantic interpretation of the ledger.

Moreover, a transaction ledger protocol is called *robust* if it provides *persistence* and *liveness*. Persistence means that a transaction that appears'deep enough' into the chain will appear at the same position for all honest miners. On the other hand, liveness means that if a transaction is input to the honest miners for at least v rounds it will appear k blocks deep into the chain of those miners.

In our work we assume the existence of a robust transactions ledger protocol $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$ for some liveness parameter v. For more details on the ledger backbone protocol protocol we refer the reader to the Appendix B.

2.2 Dictatorial Miners

In our model, the transaction ledger protocol is executed by miners in the presence of a PPT adversary $\mathcal{S}$, and driven by a PPT environment $\mathcal{Z}$. The adversary $\mathcal{S}$ can fully corrupt a minority of the miners (as in [16]). In contrast to the model of [16], every miner that is not fully corrupt will automatically be dictatorial, leaving no honest miner in the protocol. The difference between an honest miner and a dictatorial miner is that a dictatorial miner has preferences over the content of transactions and might therefore re-order, delay, or suppress transactions it receives from the environment.

Formally, a dictatorial miner M_i receives, at the beginning of each round, in its transaction buffer X_i candidate transactions provided by the environment $\mathcal{Z}$, just as honest miners would. However, M_i may execute the input contribution function $\mathsf{I}(\cdot)$ on a modified X_i' instead. At the beginning of each round M_i may choose X_i' depending on the received transaction buffer X_i and the current ledger l. For every transaction $tx \in \mathsf{X}_i$, M_i decides whether to *include* tx in X_i', to *suppress* it, or to *delay* it to a later round.

When deciding on how to treat a candidate transaction $tx[\gamma]$ a dictatorial miner is assumed to maximize its expected utility with respect to its private utility function $u_i : l \times \Gamma \mapsto \mathbb{R}$. The utility function $u_i(l, \gamma)$ computes the utility of M_i for including a transaction $tx[\gamma]$ into the ledger l[5]. If a miner M_i includes a transaction tx without knowing its content the expected utility for including this transaction is denoted as $u_i(l, tx)$. The miner's expectation is taken over the distribution on the transaction's contents supplied by the environment $\mathcal{Z}$ which is assumed to be common prior for all miners[6].

[5] The utility function covers all kinds of revenues a miner might expect including fees, extractable values, bribes. Since, estimating this utility is rather complex we assume the function only to be known to the respective miner itself.

[6] For the simplicity of our model we assume that all dictatorial miners share a common believe on the contents of transactions the environment $\mathcal{Z}$ will provide. In practice, this belief might be different for the miners considering the information available to the miners. However, aligning the believes might be part of the collaboration between the miners.

We say that a miner prefers to include a transaction $tx_0[\gamma_0]$ over transaction $tx_1[\gamma_1]$ in some ledger l if $u_i(l, \gamma_0) > u_i(l, \gamma_1)$. The preference is denoted as $tx_0[\gamma_0] \succ tx_1[\gamma_1]$ if the ledger l is evident from the content. A miner is indifferent between $tx_0[\gamma_0]$ and $tx_1[\gamma_1]$, denoted as $tx_0[\gamma_0] \sim tx_1[\gamma_1]$, if $u_i(l, \gamma_0) = u_i(l, \gamma_1)$. Appending a new transaction results in a new ledger, therefore the utility for appending $tx_0[\gamma_0]$ and then appending $tx_1[\gamma_1]$ is $u_i(l, \gamma_0) + u_i(l||\gamma_0, \gamma_1)$. Finally, we denote an empty content of a transaction as $\perp$. The expected utility for including an empty or neutral content $\perp$ is assumed to be $u_i(l, \perp) = 0$, since including an empty transaction results semantically in the same ledger.[7]

Miner's Coalition. Dictatorial miners may collude with each other if forming a coalition yields a higher individual utility. In fact, [10] shows that even if miners have concurrent preferences, e.g., they are concurring for the same MEV opportunity, collaborative behavior is not just stable but in fact yields a higher individual utility. Therefore, it is assumed (and observed in practice) that dictatorial miners will eventually collude if it is individually more profitable [10]. Our model allows dictatorial miners to coordinate their actions against single transactions, thus whenever there exists an individually profitable coalition of dictatorial miners, one could see it as a single entity.[8] Note that this may include the grand coalition of all miners.

Forking. We stress, that dictatorial miners execute the same input contribution function as honest miners (with potentially different inputs), thus only creating valid blocks and extending the current chain. While a sufficiently large coalition of malicious miners could fork the the longest chain, the simple reorder, delay or suppression of transactions that can be performed by dictatorial miners is a much more subtle deviation from the honest behavior compared, thus allowing for a positive result even against a coalition of all dictatorial miners.

3 Content Preference Robustness

In this section we formalize the concept of content preference robustness, that yields liveness and transaction order guarantee against dictatorial miners. To this end, we define the properties *rational liveness* and *rational transaction ordering*. Further, we explain the rational of restricting dictatorial miners beliefs in order to exclude trivial cases from our model.

3.1 Rational Liveness

Rational liveness says that a transaction that is input to all dictatorial miners will eventually appear in the ledger. Consider a transaction ledger protocol $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$. Rational liveness is defined as:

[7] Appending an empty transaction does not change, e.g., any balances nor account states.

[8] Practically, this means that dictatorial miners might even exchange secret information, e.g. if a transaction content is secret shared amongst the miners.

Definition 1 (Rational liveness). *If a transaction $tx[\gamma] \in \mathcal{T}$ issued by* TxGen *is input for all dictatorial miners for at least v consecutive rounds, then all dictatorial miners will report this transaction at least k blocks deep into the ledger, for some $k, v \in \mathbb{N}$.*

Intuitively, rational liveness provides the same guarantees as the liveness definition of [16], but extended to dictatorial miners. In order to show that a *robust* transaction ledger protocol Π achieves this property one has to show that it is in the dictatorial miners best interest to behave as an honest miner, such that the transactions provided by the environment are forwarded unchanged to the input contribution function.

3.2 Rational Transaction Ordering

The original model of [16] does not formalize the concept of transaction ordering. However, as practically demonstrated by [10] dictatorial miners can extract significant utility by rearranging transactions in different ways. Therefore, transaction ordering is of major relevance when considering dictatorial behavior of the miners. To this end we define rational transaction ordering preservation as follows:

Definition 2 (Rational transaction ordering). *A transaction ledger protocol $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$ preserves rational transaction ordering if for all pairs of transactions (tx_0, tx_1) issued by* TxGen*, for all ledgers $l \in \mathcal{L}$ and for all miners $M_i \in \mathcal{M}$, we have that $u_i(l||tx_0, tx_1) = u_i(l||tx_1, tx_0)$.*

Intuitively, rational transaction order preservation means that a dictatorial miner receives the same expected utility for including transaction tx_0 before transaction tx_1 into a ledger l for all pairs (tx_0, tx_1) and all ledgers $l \in \mathcal{L}$. In particular this means that a dictatorial miner does not improve its expected utility by altering the order of transactions received by the environment, and hence has no incentive to do so.

3.3 Restrictions on the Environment

Since we allow the miner's utility function to depend on off-chain dynamics, it is possible that the utility of a miner is always negative. Thus, this miner would do strictly better by always suppressing all transactions, independently of their content (or the current ledger). This might be induced, e.g., by some utility source that offers a large compensation for mining empty blocks. Since this kind of utility source would make the rational decision independently of the transaction's contents or the ledger state, we explicitly exclude them from our model.

For this, we limit the expectations of the miners about the environment $\mathcal{Z}$ such that all transactions sent to the miners are expected to yield at least a positive utility, as otherwise the miners are not incentivized to include transactions.

Definition 3 (Expected incentive compatible environment). *An environment $\mathcal{Z}$ providing transactions to a set of miners $\mathcal{M}$ executing the ledger protocol Π is expected incentive compatible if for all miners $M_i \in \mathcal{M}$, for all ledgers $l \in \mathcal{L}$, and for all $tx \in \mathcal{T}$ we have that $u_i(l, tx) > 0$.*

Note, that this does not restrict the environment itself but rather the believe of the dictatorial miners about the environment. Intuitively, this means that the miners expect the environment to provide transactions that yield a positive utility on average, where the expectation is taken over the distribution of transaction contents the environment provides. This assumption is essential to ensure that dictatorial miners will eventually improve their expected utility by including transactions. Note that miners expecting to lose utility for including transactions provided by the environment would trivially not include it. In practice, this is usually accomplished by transaction fee mechanisms.[9] Note however that this assumption *does not* trivialise the problem, as even with an expected incentive compatible environment, dictatorial miners could still increase their profits by, e.g., suppressing or reordering selected transactions from the ledger.

Content Preference Robustness. Putting all together, we now define the notion of *content preference robustness* (CPR) for a transaction ledger protocol executed in the presence of dictatorial miners.

Definition 4 (content preference robustness). *A transaction ledger protocol $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$ executed by a set of dictatorial miners $\mathcal{M}$ driven by some* expected incentive compatible *environment $\mathcal{Z}$ in presence of a PPT adversary $\mathcal{S}$ is called CPR if Π achieves* rational liveness *and* rational transaction ordering.

4 Compiling a Robust Ledger into a CPR Ledger

In this section we show how to get *content preference robustness* for a transaction ledger protocol. In particular, we show how to generically compile a *robust* transaction ledger protocol into a CPR transaction ledger protocol.

4.1 CPR Compiler

A CPR compiler for a robust transaction ledger protocol Π is defined as follows.

Definition 5 (CPR Compiler). *Let $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$ be a robust transaction ledger protocol. A CPR compiler Φ is a PPT algorithm $\Pi' \leftarrow \Phi(\Pi)$ such that $\Pi' = (\mathsf{I}', \mathsf{V}', \mathsf{R}', \mathsf{TxGen}', \mathcal{L}')$ achieves CPR.*

In the following we provide an overview of our CPR compiler, followed by a detailed description of how our CPR-compiler Φ transforms each component of a robust transaction ledger protocol Π to build the CPR ledger protocol Π'.

[9] Note, that in practice it would be sufficient for the dictatorial miners to believe that the environment is incentive compatible for themselves. However, for the sake of simplicity of our model we assume that miners believe that the environment is expected incentive compatible for every miner.

4.2 Compiler Overview

The CPR compiler modifies the transaction generation oracle TxGen into a "time-lock transaction generation oracle", that issues transactions inside time-lock puzzles [5,28] that can be opened after some specified time has passed. The idea is to let the miners commit to a set and order of time-locked transactions before their content gets revealed. This forces the miners to make decisions about incoming transactions before knowing the actual content of the transactions. However, the miners might try to delay transactions until their content gets revealed before making their final decision. Therefore, the protocol Π' has to ensure that delaying transactions is not expected to br profitable for the miners.

The compiled transaction ledger protocol Π' maintains two separate chains; the "control" chain l'_c that contains a ledger consisting of time-locked transactions, and the "sanitized" chain l'_m that contains the actual contents of the transactions that are final in the control chain. The mining of new blocks follows the rules of the underlying robust ledger protocol Π (e.g., Proof-of-Work), and it occurs in the control chain first. To extend the ledger, miners first gather all the new (time-locked) transactions from their input buffer and build a block. The miner that wins the right to append a block to the control chain is also responsible for extending the sanitized chain by providing solutions to the time-lock puzzles that are in the common-prefix of the control chain.

To illustrate with a concrete example, consider a robust ledger protocol with liveness parameter v of 2 rounds, i.e., after 2 rounds a transaction is considered final in the ledger, e.g. being $k = 2$ blocks deep into the chain. Assume a time-lock puzzle instantiation that can only be opened 2 rounds after its creation. Hence, for the first 2 rounds of the protocol only the control chain will be extended, as no solution to puzzles is yet available. At round 3 the miner that creates the new block on the control chain must also create a new block on the sanitized chain by providing solutions to all the puzzles that are included in the block created at round 1 in the control chain; the solution to the puzzles are in fact the contents of the transactions. From round 3 onwards, every new round will extend both chains, and the sanitized chain at round r will contain the contents of the transactions included at round $r - 2$ in the control chain. As all transactions that are at least 2 blocks deep in the ledger are final, then the blocks in the sanitized chain only includes contents that are already final.The structure of the build ledger is illustrated in Fig. 1.

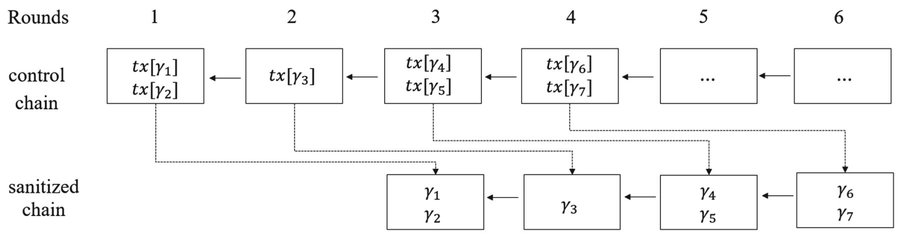

Fig. 1. High-level architecture of a CPR ledger with liveness parameter $v = 2$.

Note that the control chain is the only chain that needs consensus, as the state of the sanitized chain can be deterministically retrieved from the control chain. However, miners can still freely choose the time-locked transactions to be included in the control chain, and therefore try to delay a time-locked transaction until learning its content before deciding to either include it or suppress it. To this end, our construction implements a mechanism that invalidates transactions that are "too old". Thus, dictatorial miners face the dilemma of either including a time-locked transaction without knowing its content, or delaying it, which leads to the invalidation of the transaction and loss of utility. To solve the dilemma, rational miners must rely on their expectation on the transactions' content.

4.3 Time-Lock Transaction Generation Oracle

Here we describe how our CPR compiler Φ builds the time-lock transaction generation oracle TxGen' for the compiled protocol Π'. A natural way to model a time-lock transaction generation oracle TxGen' is by describing it as a functionality. The functionality we need from such an oracle is that transactions can be created on behalf of users and the miners are merely notified that a transaction has been created. Moreover, published transactions should be associated with a fresh ledger account. The content corresponding to the transaction can only be retrieved after some predetermined number of rounds.

More formally, the $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ functionality encapsulates the content γ of a transaction into a randomly generated transaction tag $\tilde{tx}$ that is delivered to every miner. A miner can learn the content associated with a transaction tag *at least* δ rounds after the transaction has been issued by sending at least δ solve requests for the same transaction tag to $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$. Once, the correct amount of requests is issued the functionality returns the associated solution tag sid. Further, the functionality allows to verify if a content belongs to a transaction tag by returning the corresponding content. To this end, on receiving a solution tag sid at any time the functionality $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ returns the associated content γ. By separating, solving the puzzle from revealing the message it is ensured that a message can be revealed at any time if sid is known, even without solving the puzzle. Note that in our functionality, the issuing round of a transaction as well as a fresh ledger account are also associated with the transaction tag.

Practically, this means that users sign the time-locked transaction using a freshly generated ledger account and include a time stamp of the creation time, which relates to the current round.[10] The first can be used for practical reasons to deter malicious users from flooding the ledger with time-locked transactions, as we will discuss later on in Sect. 5. The latter is used by the protocol to determine whether a transaction is "too old". The $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ functionality is described next.

[10] According to Garay *et al.* this would practically relate to the block number. Therefore this "timestamping" of the creation round is practically achieved by including the latest known block number.

Functionality $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$

The $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ functionality is parameterized by a delay parameter δ that represents how many rounds the transaction content stays hidden, a set of transaction tags $\tilde{\mathcal{T}}$, a set of ledger accounts $\mathcal{A}$, and space of solution tags Sid. It is executed by a set of users $\mathcal{P}$, a set of miners $\mathcal{M}$, and an adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ maintains initially empty lists $\mathcal{O}$ and $\mathcal{Q}$.

- On message $(\mathtt{IssueTx}, \mathsf{txid}, \gamma)$ from $P_i \in \mathcal{P}$ in some round r the functionality samples $\tilde{tx} \leftarrow_\$ \tilde{\mathcal{T}}$, $\mathsf{sid} \leftarrow_\$ \mathsf{Sid}$, and the account $A \leftarrow_\$ \mathcal{A}$. Then, it records the tuple $(\mathsf{txid}, \tilde{tx}, \mathsf{sid}, \gamma, r, A, P_i)$ in $\mathcal{O}$ and sends it to P_i. Further, send the message $(\mathtt{Issued}, \mathsf{txid}, \tilde{tx}, r, A)$ to every $M \in \mathcal{M}$ and $\mathcal{S}$.
- On message $(\mathtt{Solve}, \mathsf{txid}, \tilde{tx})$ from $M_i \in \mathcal{M}$ in some round r the functionality does the following: If no record $(\mathsf{txid}, \tilde{tx}, r, M_i)$ exists in $\mathcal{Q}$ append $(\mathsf{txid}, \tilde{tx}, r, M_i)$ to $\mathcal{Q}$. Let L be the set of records of the form $(\mathsf{txid}, \tilde{tx}, \cdot, M_i)$ in $\mathcal{Q}$ and $(\mathsf{txid}, \tilde{tx}, \mathsf{sid}, \gamma, r', A, P_j)$ a record in $\mathcal{O}$. If $|L| \geq \delta$ send the message $(\mathsf{txid}, \tilde{tx}, \mathsf{sid}, r, M_i)$ to M_i and $\mathcal{S}$. Otherwise send message $(\mathsf{txid}, \tilde{tx}, \perp, r, M_i)$.
- On message $(\mathtt{RevealMsg}, \mathsf{txid}, \tilde{tx}, \mathsf{sid})$ from $M_i \in \mathcal{M}$ in some round r the functionality does the following: If there is a record $(\mathsf{txid}, \tilde{tx}, \mathsf{sid}, \gamma, r', A, P_j)$ in $\mathcal{O}$ send the message $(\mathsf{txid}, \tilde{tx}, \mathsf{sid}, \gamma, r', A, P_j, r, M_i)$ to M_i and $\mathcal{S}$. Otherwise send message $(\mathsf{txid}, \tilde{tx}, \mathsf{sid}, \perp, \perp, \perp, \perp, r, M_i)$.

Note that we intentionally separate the set of users (that only post transactions) and miners (that process the transactions) in the $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ functionality. This is to better illustrate that our results only concern the case where miners do not send transactions. It is easy to see that whenever a miner itself generates transactions it is not possible to prevent this miner from learning the content of its own transaction. Therefore, we restrict the functionality to only accept `IssueTx` commands from users.

In order to instantiate the transaction generation oracle TxGen' of the compiled protocol Π' the functionality $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ should be parameterized with a delay parameter $\delta = v$, where v is the liveness parameter of Π. Further, the content space Γ' for the protocol Π' corresponds to the transaction space $\mathcal{T}$ in the support of the transaction generation oracle TxGen of the underlying transaction ledger protocol Π. Intuitively, this means that the environment $\mathcal{Z}$ samples transactions in the support of TxGen that are provided as content to $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ when issuing a transaction.[11]

Realizing the $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ Functionality. Our $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ functionality is a simplified version of the time-lock puzzle functionality of [5, Fig. 3], where we simply cast it as a transaction generation oracle. In particular, the original functionality sam-

[11] Nevertheless, the functionality $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ is not restricted to a specific content space.

ples a consecutive sequence of puzzle states, while sending a solve request returns the next state in the sequence. For the sake of simplicity of our functionality we omit the intermediary states. Instead in our functionality a puzzle tag is queried multiple times to solve the puzzle until the functionality returns a solution tag. This solution tag corresponds to the last puzzle state from the original functionality and can be used to reveal the massage at any time as in [5]. Note, that this simplification does not interfere with the security since no additional information is leaked. Additionally, our functionality samples a new account that is associated with the transaction tag. Since this tag is sampled uniformly at random it does not leak any information about the content or solution tag associated with the transaction tag and therefore does not have any impact on the security.

In [5], it was shown that (a version of) the well known time-lock puzzle construction by Rivest, Shamir and Wagner [28] realizes the time-lock puzzle functionality of [5] in the Universal Composition (UC) model [9]. The time-lock puzzle construction of [28] is based on the assumption that it is hard to compute repeated squarings of an element of $(\mathbb{Z}/N\mathbb{Z})^\times$ with large N in less time than it would take to compute each of the squarings sequentially, unless the factorization of N is known. Therefore, in order to solve a time lock puzzle a miner has to perform a predefined amount sequential squarings.

Hence, by the composition property of the UC framework, we can simply use the time-lock puzzle protocol of [5] as a plug-in replacement for our functionality.

4.4 Chain Validity Function

A ledger l' of the compiled transaction ledger protocol Π' consists of the control chain l'_c and the sanitized chain l'_m. The control chain is a ledger of tuples $tx' = (\mathsf{txid}, \tilde{tx}, A)$ providing time-locked transaction tags $\tilde{tx}$ from the tag space $\tilde{\mathcal{T}}$, an associated ledger account A from the the ledger account space $\mathcal{A}$, and unique transaction identifiers txid. The sanitized chain l'_m is a ledger of transaction contents containing tuples of the form $\gamma' = (\mathsf{sid}, \gamma, r, P)$. A transaction ledger l' consists of blocks $(\mathcal{B}_1^{r_1}, ..., \mathcal{B}_n^{r_n})$ where for each block $\mathcal{B}_i^{r_i}$, r_i denotes the round the block is created in. Each block extends the control chain l'_c and the sanitized chain l'_m. Therefore, each block is a tuple $\mathcal{B}_i^{r_i} = (\mathcal{B}_{c,i}^{r_i}, \mathcal{B}_{c,i}^{r_i})$, with $\mathcal{B}_{c,i}^{r_i} = (tx'_{i,1}, ..., tx'_{i,y})$ and $\mathcal{B}_{c,i}^{r_i} = (\gamma'_{i,1}, ..., \gamma'_{i,q})$. For simplicity we denote $l' = (l'_c, l'_m) = ((tx'_1, ..., tx'_p), (\gamma'_1, ..., \gamma'_q))$ when referring to the concatenation of all blocks. A ledger l' is in the set of valid ledgers $\mathcal{L}'$ if the chain validation function V' returns *true* on l'. Intuitively, $\mathsf{V}'(\cdot)$ checks for every content in the sanitized chain if it corresponds to the transaction tag at the same position in the control chain. The formal algorithm for checking if a ledger l' is a valid ledger for the transaction ledger protocol Π' is provided in Algorithm 1.

4.5 Input Contribution Function

The input contribution function $\mathsf{I}'(\cdot)$ is executed in order to generate an updated ledger. It receives as input a current transaction ledger $l' = (l'_c, l'_m) = ((tx'_1, ..., tx'_p), (\gamma'_1, ..., \gamma'_q))$, a buffer of not included transactions X. Further, $\mathsf{I}'(\cdot)$

Algorithm 1. $\mathsf{V}'(l')$

```
1: parse $(l'_c, l'_m) \leftarrow l'$
2: parse $(tx'_1, ..., tx'_p) \leftarrow l'_c$
3: parse $(\gamma'_1), ..., (\gamma'_q)) \leftarrow l'_m$
4: for $i = 1, ..., q$ do
5:     parse $(\mathsf{txid}_i, \tilde{tx}_i, A) \leftarrow tx'_i$
6:     parse $(\mathsf{sid}_i, \gamma_i, r_i, P_i) \leftarrow \gamma'_i$
7:     send $(\texttt{RevealMsg}, \mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i)$ to $\mathcal{F}^{\delta}_{\mathsf{tl\text{-}TxGen}}$ to receive $(\mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i, \tilde{\gamma}_i, \tilde{r}_i, \tilde{P}_i)$
8:     if $\tilde{\gamma}_i \neq \gamma_i \vee \tilde{r}_i \neq r_i \vee \tilde{P}_i \neq P_i$ then:
9:         return false
10:    end if
11: end for
12: return true
```

is stateful by maintaining a buffer of time-lock puzzle solutions C. Intuitively, $\mathsf{I}(\cdot)$ starts with solving all transaction tags from the control chain that are not solved yet. All found solutions are stored in the solution buffer. Then, in order to extend the ledger l' the function $\mathsf{I}'(\cdot)$ extends the sanitized chain with the contents of transaction tags from the block that is k blocks deep into the control chain. Further, $\mathsf{I}'(\cdot)$ extends the control chain by selecting a set of new time-lock transaction tags from X. Finally, $\mathsf{I}'(\cdot)$ includes for all selected time-locked transactions the current round number.[12] The formal algorithm is shown in Algorithm 2.

Algorithm 2. $\mathsf{I}'(l', \mathsf{X}, r)$

```
1: parse $(l'_c, l'_m) \leftarrow l'$
2: parse $(tx'_1, ..., tx'_p) \leftarrow l'_c$
3: parse $(\gamma'_1), ..., (\gamma'_q)) \leftarrow l'_m$
4: for $i = q+1, ..., p$ do
5:     parse $(\mathsf{txid}_i, \tilde{tx}_i, A) \leftarrow tx'_i$
6:     send $(\texttt{Solve}, \mathsf{txid}_i, \tilde{tx}_i)$ to $\mathcal{F}^{\delta}_{\mathsf{tl\text{-}TxGen}}$ to receive message $(\mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i)$
7:     if $\mathsf{sid}_i \neq \bot$ then:
8:         send $(\texttt{RevealMsg}, \mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i)$ to $\mathcal{F}^{\delta}_{\mathsf{tl\text{-}TxGen}}$ to receive $(\mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i, \gamma_i, r_i, A, P_i)$
9:         append $(\mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i, \gamma_i, r_i, A, P_i)$ to $\mathsf{C}$
10:    end if
11: end for
12: parse $(\mathcal{B}^{r_1}_{c,1}, ..., \mathcal{B}^{r_n}_{c,n}) \leftarrow l'_c$
13: $j \leftarrow n + 1 - k$
14: parse $(tx'_{q+1}, ..., tx'_{q+y}) \leftarrow \mathcal{B}^{r_j}_{c,j}$
15: $\mathcal{B}^{r}_{m,n+1} = \bot$
16: for $i = q+1, ..., q+y$ do
17:     get $(\mathsf{txid}_i, \tilde{tx}_i, \mathsf{sid}_i, \gamma_i, r_i, A, P_i)$ from $\mathsf{C}$
18:     $\gamma' \leftarrow (\mathsf{sid}_i, \gamma_i, r_i, P_i)$
19:     $\mathcal{B}^{r}_{m,n+1} = \mathcal{B}^{r}_{m,n+1} || \gamma'$
20: end for
21: get $(tx'_p, ..., tx'_{p+j})$ from $\mathsf{X}$
22: $\mathcal{B}^{r}_{c,n+1} \leftarrow (tx'_p, ..., tx'_{p+j})$
23: $\mathcal{B}^{r}_{n+1} = (\mathcal{B}^{r}_{c,n+1}, \mathcal{B}^{r}_{m,n+1})$
24: return $l' = l' || \mathcal{B}^{r}_{n+1}$
```

Remark. One could also separate the tasks of solving time-lock puzzles and extending the chain into different functions. However, since extending the ledger depends on solving the puzzles we simply incorporate solving the puzzles into $\mathsf{I}(\cdot)$. By doing so, we additionally stick closer to the model of [16].

[12] According to Garay *et al.* this "timestamping" of the inclusion round is practically achieved by giving a block number to the selected transactions.

4.6 Chain Reading Function

The chain reading function $\mathsf{R}'(\cdot)$ returns a semantic interpretation of a ledger $l' \in \mathcal{L}'$. It receives as input a transaction ledger l' and internally calls the chain validation function $\mathsf{V}(\cdot)$ and chain reading function $\mathsf{R}(\cdot)$ of the underlying transaction ledger protocol Π. The intuition is that the revealed contents in the sanitized chain contain transactions in the support of TxGen from the protocol Π. Therefore, $\mathsf{R}'(\cdot)$ can determine the longest chain that is a valid ledger with respect to $\mathsf{V}(\cdot)$. This longest chain can then be interpreted by $\mathsf{R}(\cdot)$. Additionally, $\mathsf{R}'(\cdot)$ checks for every transaction content in the sanitized chain if the corresponding time-locked transaction tag was included "too late" in the control chain by comparing the revealed round number of the transaction tag generation with the round number of the block that included the transaction tag. If the difference between block creation round and transaction creation round is more than the liveness parameter v the content gets ignored. The function $\mathsf{R}'(\cdot)$ ensures that transaction contents that are considered as "too old" are not interpreted by $\mathsf{R}(\cdot)$ and are therefore not considered for the semantic interpretation of the ledger l'. The algorithm for R' is shown in Algorithm 3.

Algorithm 3. $\mathsf{R}'(l')$

```
1: if $\mathsf{V}'(l') = false$ then:
2:     return $\perp$
3: end if
4: $(l'_c, l'_m) \leftarrow l'$
5: parse $(\mathcal{B}^{r_1}_{c,1}, ..., \mathcal{B}^{r_n}_{c,n}) \leftarrow l'_c$
6: parse $(\mathcal{B}^{r_1}_{m,1}, ..., \mathcal{B}^{r_n}_{m,n}) \leftarrow l'_m$
7: $\tilde{l} \leftarrow \perp$
8: for $i = 1, ..., n$ do
9:     $(\gamma'_{i,1}, ..., \gamma'_{i,y}) \leftarrow \mathcal{B}^{r_i}_{m,i}$
10:     for $j = 1, ..., y$ do
11:         parse $(\mathsf{sid}_j, \gamma_j, r_j, P_j) \leftarrow \gamma'_j$
12:         parse $tx[\tilde{\gamma_j}] \leftarrow \gamma_j$
13:         if $\mathsf{V}(\tilde{l}||tx[\tilde{\gamma_j}]) = true \wedge r_i \leq r_j + v$ then:
14:             $\tilde{l} \leftarrow \tilde{l}||tx[\tilde{\gamma_j}]$
15:         end if
16:     end for
17: end for
18: return $\mathsf{R}(\tilde{l})$
```

4.7 Security Analysis

Now we state our main theorem and show that the compiled transaction ledger protocol Π' compiled by our CPR-compiler Φ is indeed CPR if the underlying protocol Π is robust.

Theorem 1. *Let Π be a robust transaction ledger protocol. Then, for all expected incentive compatible environments $\mathcal{Z}$, the compiled transaction ledger protocol $\Pi' \leftarrow \Phi(\Pi)$ achieves CPR.*

The intuition of the proof is to show that dictatorial miners maximize their utility by behaving honestly in all expected incentive compatible environments. If all dictatorial miners behave as honest miners we can conclude that Π' is CPR. The intention is that no single miner nor coalition of miners is able to gain any a-priori knowledge of the transaction content by the time they can decide about including the transactions. Additionally, the construction deincentivizes *delaying* transactions until the miners can learn the content, due to the invalidation of 'too old' transactions. Therefore, dictatorial miners fear of missing out on the transactions and the associated expected profit if they try to learn the content first. The full analysis is deferred to Appendix C.

5 Discussions

In this section we discuss several aspects of our results.

Coercion Resistance in CPR Transaction Ledger Protocols. As shown in our analysis, rational liveness can be achieved if the dictatorial miners gain no a-priori information about the content of candidate transactions. However, miners may try to coerce users in order to make them reveal the content of their transactions a-priori. By doing so, the miners would again be able to enforce their preferences on the transactions. While coercion resistance is to the best of our knowledge a new issue in transaction ledger protocols, it is quite well known in voting protocols [11,18]. However, the techniques from the voting literature does not seem to apply in our setting. The inability of a user to prove the content of its transaction to a miner would not solve the problem of coercion. Recall that all transactions are eventually opened in the sanitized chain. Thus, a miner could simply establish a contract where a user commits to the content that it discloses before hand, and gets punished if the time-locked transaction opens to something else later on. This coercion attack is possible in this context since in transaction ledger protocol it is very unlikely that another user sends a transaction with the same content. In voting on the other hand it is very much possible that there is another ballot with the same vote. We believe that coercion-resistance in the setting of transaction ledger protocols is an interesting open problem left for future work.

Performance Considerations. In our compiler, every transaction is included as a time-lock puzzle in the control chain, while the solution must be provided later in the sanitized chain, once the puzzle is deep enough in the control chain. This requires miners to solve time-lock puzzles for every transaction. While this additional computational effort burdens the miners, we note that the computational cost *per transaction* is constant (unlike, e.g. PoW). Nevertheless, it is possible to improve the efficiency significantly in practice. For example, one could allow the issuer of the transaction to reveal the puzzle solution once the puzzle is deep enough in the control chain, such that the miners only have to solve the puzzles for non responsive users. Alternatively, recent advancements in time-lock puzzles yield promising results. In particular, Abadi and Kiayias [1] recently proposed a construction for "chained time-lock puzzles" that allows to compose multiple puzzles into a single one, hence dramatically reducing the computational effort of solving multiple the puzzles.

Invalid and Conflicting Transactions. Naturally, it is hard to decide if a time-lock puzzle contains a transaction that does not contradict any other transaction already in the chain. However, we stress that this is not an issue for the safety and correctness of the ledger since it is still possible to deterministically consider only valid transactions in the sanitized chain. Nevertheless, the inclusion of inconsistent transactions in the control chain might be undesirable in practice. Therefore, flooding the ledger with time-lock puzzles of invalid transactions should be deincentivized. Flooding attacks are usually deincentivized by a fee paid by the sender of a transaction [8,26]. To this end, in our construction, a time-locked transaction is associated with a ledger account. This associated ledger account can be charged fees for including the time-locked transaction in the control chain, independently of the content hidden inside the time-lock puzzle. Note, that this fee does not necessarily replace any fees for executing the transactions' contents hidden inside the time-lock puzzle. This assumes that users have access to unlinkable ledger accounts, such that the associate ledger account does not reveal any information about the content hidden inside the time-lock puzzle. This can usually be achieved using anonymization and mixing techniques [6,29]. However, as in Garay et al. [16], the design of a concrete fee mechanism is outside the scope of this work.

Targeted Censorship. We stress that in this work we are not concerned with censorship targeted at individuals, but we only consider preferences over the contents of the transactions. We note that the full anonymization of transactions requires not only protocol level anonymization but also network level anonymization, what is known to be a hard problem in practice. Dictatorial miners, might be able to gain some information about the transaction content form the network layer of the protocol, for example learning the sender node of a transaction in the underlying network. However, to protect against some level of censorship against individuals, one can still run the CPR protocol over TOR [13].

Alternative Approaches. A different approach to achieve CPR could be by leveraging threshold cryptography or multi-party computation [25]. However, even if threshold cryptography is clearly capable of preventing an unqualified set of miners from taking rational actions, it also inherently defines a coalition of miners that can. As pointed out by Daian *et al.* [10], any coalition that yields higher utility should be expected to be formed eventually. Hence, we rely on time-lock puzzles which are inherently coalition resistant by design. Choosing time-lock parameters in a way that no miner can learn the content a-priori while on the same hand guaranteeing that every miner can be expected to provide the solutions is practically challenging. In particular, due to hardware differences, some miners may perform the required sequential computation faster than others. Also, aligning the delay parameter of the time-lock puzzle with the desired block creation time of the underlying transaction ledger protocol might be challenging, especially for probabilistic block creation times [8,26]. Therefore, a practical instantiation should reflect this by considering a gap in which miners are expected to provide the solutions for time-lock puzzles. Another approach to the computational time-lock puzzle is proposed by Liu *et al.* [23]. They propose a construction of a "time-release encryption" relative to a reference clock using witness encryption. With their construction it is possible to encrypt a transaction such that its plaintext is released, e.g., at a predefined blockdepth of the ledger. While this solves the challenge of choosing time-lock puzzle parameters, the implementation of such scheme would be rather impractical, since the size of the witness used for decryption is approximately the size of the entire blockchain.

6 Conclusion

In this work we investigate the setting where "dictatorial miners" can use their a-priori knowledge of transactions' content to alter the set and order of candidate transactions in their most favorable way to improve their utility. To this end, we introduce the model of dictatorial miners that may deviate from honest behavior by suppressing or reordering transactions selectively depending on their content. We incorporate dictatorial miners in the transaction ledger protocol modeled by Garay *et al.* [16] by replacing honest miners with dictatorial ones. In that vein, we show that a transaction ledger protocol can achieve content preference robustness by guaranteeing rational liveness and rational transaction order preservation. We show that this can be achieved if dictatorial miners cannot learn the contents of transactions before they are in the common-prefix of the chain. In particular, we provide a construction for a CPR compiler that can transform any generic robust transaction ledger protocol into a CPR ledger protocol by leveraging time-lock puzzles.

Acknowledgements. We thank the anonymous reviewers for their effort and valuable comments that helped to improve this work. This work was partly supported by the German Federal Ministry of Education and Research (BMBF) iBlockchain project (grant nr. 16KIS0902) and by the German Research Foundation (DFG) via the DFG CRC 1119 CROSSING (project S7).

A Related Work

In this section we discuss some related works and compare them with our results.

Bitcoin Incentive Compatibility. Badertscher *et al.* [2] showed that Bitcoin satisfies the properties of persistence and liveness (as defined in [16]) in presence of a rational majority. However, in their work the utilities of the rational participants are restricted to a natural class of incentives for the miners, such as fees and block rewards, explicitly excluding preferences over transaction contents. In this work we extend their results and explicitly focus on utilities based on transaction contents.

Order-Fairness for Byzantine Consensus. Kelkar *et al.* [19] deal with the issue of order fairness in transaction ledger protocols. They point out that consistency and liveness do not protect against malicious manipulation of the received order of transactions. This implies that the resulting ordering of transactions does not necessarily reflect the received ordering. Their proposed solution provides block order fairness, which says that if sufficiently many miners receive a transaction tx before tx' then no honest miner can report tx' in a block before tx. Further, they show that it is not possible to guarantee received order fairness, consistency and liveness at the same time.

Moreover, [19] gives a positive result for a slightly weaker definition of order fairness under the strict assumption that a fraction of the parties behave honestly, i.e., the honest parties will not alter the order of transactions under any circumstances. In comparison, our model does allow every miner to alter the order of candidate transactions, or even suppress them, for the sake of individual profit. Our construction ensures that miners are indifferent between transactions they are supposed to include into the ledger, and do therefore not expect a higher utility for altering the transaction's order. Note however that this does not contradict the impossibility result of Kelkar *et al.* [19].

Censorship Resistant Consensus. Miner's suppresion of transactions was already addressed by Miller *et al.* [25]. In order to achieve censorship resilience they propose a BFT consensus protocol combined with threshold encryption. In the proposed construction miners select transactions from their local buffer and encrypt them under the common public key of the threshold encryption. Before decryption the miners exchange and agree on the encrypted transactions. As in [19] the security of the construction is also based on the assumption of an honest fraction of miners. We note that in our model, any qualified set of miners can decrypt the threshold encrypted messages at any time and therefore can learn the plaintext collaboratively for the sake of common and individual profit.

Time-lock Puzzle in Blockchains. Khalil *et al.* [20] provide an implementation of a trustless centralized exchange based on an underlying blockchain that prevents front-running attacks of the centralized operator and the miners of the blockchain by using time-lock puzzle. The idea of their construction is that the set and the ordering of bids and offers is determined before the plaintexts are revealed.

Deuber *et al.* [12] use time-lock puzzles to ensure opening of commitments. In [12] they propose a minting mechanism based on waiting time auctions. In order to ensure that the block creators actually include all the openings for the commitments of bids in a block, it is required that all openings to the commitments are encapsulated in a time-lock puzzle and sent together with the bid transaction. In both works time-lock puzzles are used to ensure that commitments can be opened even if the opening messages get "lost" in the way, e.g. by a corrupted miner. While both of the previous works utilize time-lock puzzles to ensure openings in their respective ledger application we leverage time-lock puzzles to strengthen the ledger itself by improving its liveness guarantees.

Further, Doweck and Eyal [14] propose a construction for a multi-party timed commitment. In their construction a set of N users engage in an interactive commitment protocol with a single coordinator to commit to a list of messages by the users that can be revealed by the coordinator at a later time. Their construction is based on El-gamal encryption with a randomly sampled public key of a small group size, where the private key is revealed by the coordinator by brute force. Additionally, they provide a construction for a transaction ledger protocol that leverages their multi-party timed commitment. However, their construction requires the users to engage in interactive commitment protocols with one or even several miners leading to a significant communication overhead especially for higher numbers of users. Moreover, the searching for an El-gamal private key can be parallelized, offering no lower bound of operations under miners' coalition. Our construction on the other hand let users publish and propagate their transactions as commonly done in transaction ledger protocols.

Bribery and MEV Attacks. There are various incentive based attacks utilizing rational miners that intend to either revise, reorder or to exclude certain transactions from the ledger. On a high level, all these attacks are dynamics that might influence a rational miner's preference over transaction content. In [22], Liao and Katz show an attacker that incentivizes forking the main chain using high transaction fees. Moreover, McCorry *et al.* [24] present a different bribery contract that makes the miners change their mining strategy. In their work, miners' utility depends on the attackers bribe rather than on hidden content preferences. For example, the goldfinger attack of [24] incetivizes miners to mine empty blocks independent of the content of any available transaction.[13]

Bribery attacks that incentivize miners to suppress transactions based on their content are proposed by Winzer *et al.* [31] and Tsabary *et al.* [30]. These types of attacks can be prevented if the dictatorial miners are *not able* to choose transactions based on their content.

Daian *et al.* [10] introduced *Time-bandit attacks* where adversarial miners can fork the blockchain by utilizing MEV opportunities. The attack works by leaving MEV opportunities in the main chain for other miners to claim, thus incentivizing other rational miners to fork the chain to claim the MEV opportunity. Similarly to [22], this subsidizes a 51% attack. Miners may be incentivized to break consensus if the block rewards are not enough in comparison to the MEV [10] opportunities. While our construction does not entirely prevent this type of attack, it mitigates it by making the attack more costly for the miner to pull off; the required fork would to claim the MEV would be considerably deeper, making it less profitable.

[13] Clearly, this attack can not be prevented by our construction and outlines the limits we will elaborate in this work; dictatorial miners might suppress transactions independent of the content if they expect to improve their utility by doing so.

Sandwich attacks are a common predatory trading strategy in which a miner or a trader "wraps" a victim's transaction between two adversarial transactions [10]. If the market price of an asset is expected to rise/fall after the execution of a large pending transaction, the adversary may extract value by inserting its own transaction right before/after the spotted pending transaction. Our construction prevents sandwich attacks since the attacker is not able to spot a target transaction and sandwich it at the same time.

Finally, Judmayer et al. [17] showed that it is almost impossible to determine the exact globally available MEV opportunities at a certain point in time, and that a narrow definition of MEV fails to capture all extractable value occasions of other actors, the emerging network dynamics, or the probabilistic nature of permissionless cryptocurrencies. In that vein, we consider the complexity of MEV by assuming the exact utility of miners for including transaction to be unknown.

B Transaction Ledger Protocol

According to Garay *et al.* [16] a transaction ledger aims at keeping a record of monetary accounts and its associated balance; a transaction record in the ledger is typically (but not limited to) an instruction to move balances between accounts. A transaction ledger is represented as a vector of blocks $l = (\mathcal{B}_1, ... , \mathcal{B}_d)$, where each block $\mathcal{B}_i = (tx_1, ..., tx_n)$ is a vector of transactions $tx \in \mathcal{T}$. $\mathcal{T}$ denotes the set of valid transactions. Appending a transaction tx to a vector l is denoted by $l||tx$. Also, appending a vector of transactions $\mathcal{B}$ to another vector l is denoted as $l||\mathcal{B}$. $tx_{i,j}$ denotes transaction tx_j in block $\mathcal{B}_i$. As a ledger is a vector of transactions, we simply denote it as $l = (tx_1, ..., tx_m)$ omiting the block numbers when clear from the context.

The transaction ledger protocol is executed by a set of miners $\mathcal{M}$ in the presence of a PPT adversary $\mathcal{S}$, and driven by a PPT environment $\mathcal{Z}$. The protocol execution takes place in rounds. The environment provides inputs to all parties and receives outputs, while the attacker might fully corrupt some of the miners. Each honest miner M_i maintains its own local copy of the chain l_i. Further, an honest miner M_i process a local buffer $\mathsf{X}_i := (tx_1, \ldots, tx_e)$, that are candidate transactions to be incorporated into the ledger l_i provided by the environment $\mathcal{Z}$. In [16], a transaction ledger protocol is defined by the transaction generation oracle TxGen, the set of valid ledgers $\mathcal{L}$ and by the three interface functions $\mathsf{V}(\cdot), \mathsf{I}(\cdot), \mathsf{R}(\cdot)$.

The transaction generation oracle TxGen generates transactions on behalf of the users $\mathcal{P}$ which are abstracted by the environment $\mathcal{Z}$. It is defined with respect to the set of valid transactions $\mathcal{T}$, the set of valid contents Γ, (which denotes the set of content information with semantic value for the ledger, e.g., "account A increases its balance by 10 monetary units") and the set of ledger accounts $\mathcal{A}$. Note that a user P_i might be associated with multiple ledger accounts. During the execution of the transaction ledger protocol, TxGen can be accessed by the environment $\mathcal{Z}$ and it generates transactions that are provided to the miners and the adversary $\mathcal{S}$. Upon receiving a message $(\mathsf{IssueTx}, \gamma, P)$ from the environment $\mathcal{Z}$, TxGen generates a unique transaction $tx[\gamma] \in \mathcal{T}$, where $tx[\gamma]$ denotes a transaction tx that contains an encoding of content γ. After that, TxGen sends $(\texttt{Issued}, tx[\gamma], A)$ for some ledger account $A \in \mathcal{A}$ to every miner and $\mathcal{S}$.

On the other hand, the three interface functions $\mathsf{V}(\cdot), \mathsf{I}(\cdot), \mathsf{R}(\cdot)$ are defined as follows:

- $\mathsf{V}(l)$: The content validation predicate, upon input a sequences of transactions $(tx_1[\gamma_i], ..., tx_m[\gamma_m])$ checks whether all the transactions constitute a semantically valid ledger. Formally, $\mathsf{V}(\cdot)$ defines the set of valid ledgers $\mathcal{L}$ and checks if $l \in \mathcal{L}$, e.g. $\mathsf{V}(l)$ checks if there are no conflicting transactions in l.
- $\mathsf{R}(l)$: The chain reading function returns a semantic interpretation of the contents $(\gamma_1, ..., \gamma_n)$, e.g. a list of account addresses and balances Upon receiving a ledger $l = (tx_1[\gamma_1], \ldots, tx_n[\gamma_n])$, and if $\mathsf{V}(l) = 1$.
- $\mathsf{I}(l, \mathsf{X}, r)$: Upon receiving a ledger and a buffer of local transactions in some round r the input contribution function creates some new block $\mathcal{B} = (tx_1[\gamma_1], \ldots, tx_e[\gamma_e])$, where $tx_i \in \mathsf{X}$ and returns $l' := l||\mathcal{B}$.

Moreover, a transaction ledger protocol is called *robust* if the following properties are satisfied:

- *Persistence:* If at any round r an honest miner M_i maintains a ledger that contains a transaction $tx \in \mathcal{T}$ in a block more than $k \in \mathbb{N}$ blocks deep in the chain, then tx occurs at the same position in the chain of all the other honest miners.
- *Liveness:* If a transaction $tx \in \mathcal{T}$ issued by TxGen is input for all honest miners in $\mathcal{M}$ for at least v consecutive rounds, then all honest miners will report this transaction at least k blocks deep into the ledger, for some $k, v \in \mathbb{N}$.

According to [16] a robust transaction ledger protocol can be build on top of a blockchain backbone protocol that satisfies the properties *common prefix*, *chain quality* and *chain growth* by defining the interfaces $\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}$, and $\mathcal{L}$. In our work we assume the existence of a robust transactions ledger protocol $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$ for some liveness parameter v. Therefore, we can waive details of the underlying backbone protocol that is used to implement Π. For more details on the ledger backbone protocol protocol we refer the reader to the paper of Garay *et al.* [16].

C Analysis of Theorem 1

Let $\Pi = (\mathsf{I}, \mathsf{V}, \mathsf{R}, \mathsf{TxGen}, \mathcal{L})$ be a robust transaction ledger protocol executed by a set of miners $\mathcal{M}$ in the presence of a PPT adversary $\mathcal{S}$ driven by some environment $\mathcal{Z}$, and let $\Pi' = (\mathsf{I}', \mathsf{V}', \mathsf{R}', \mathcal{F}_{\mathsf{tl\text{-}TxGen}}, \mathcal{L}')$ be the compiled transaction ledger protocol $\Pi' \leftarrow \Phi(\Pi)$ executed by a set of dictatorial miners $\mathcal{M}'$ in the presence of a PPT adversary $\mathcal{S}$ driven by some environment $\mathcal{Z}'$. Let $\delta = v$ be the delay parameter of $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$.

At any round r a dictatorial miner M_i receives a transaction buffer X_i from the environment $\mathcal{Z}'$ and provides an altered transaction buffer X_i' to the input contribution function $\mathsf{I}'(\cdot)$.

In order to show that Π' achieves CPR it is necessary to show that Π' achieves *rational liveness* and *rational transaction ordering.* To this end, we show that a dictatorial miner M_i can not improve its expected utility u_i by deviating from honest behavior. A dictatorial miner M_i behaves honest if $\mathsf{X}_i' = \mathsf{X}_i$ at any round r. If it is in the best interest of dictatorial miners to behave honest it can be concluded that Π' achieves rational liveness if Π achieves liveness.

Let therefore $\mathsf{X}_i = (tx_1', ..., tx_n')$ with $tx_j' = (\mathsf{txid}_j, \tilde{tx}_j, A_j)$ for all $j \in [n]$ be a transaction buffer that is provided to some dictatorial miner M_i in some round r_y for some current ledger $l_{r_y}' \in \mathcal{L}'$ by the environment $\mathcal{Z}$. Since, $\mathcal{Z}$ is expected incentive compatible it holds that $u_i(l', tx_j') > 0$ for every $tx_j' \in \mathsf{X}_i$. Therefore, it follows that M_i prefers to *include* tx_j' over suppressing it, if $\gamma_j' = (\mathsf{sid}_j, \gamma_j, r_j, P_j)$ associated with tx_j' is sampled by $\mathcal{Z}$ from some common prior distribution over Γ and M_i did not gain any additional information about γ_j. Therefore, a dictatorial miner that is able to reduce its uncertainty about γ_j over the course of some rounds might actually be able to improve its expected utility by *suppressing* tx_j'. Consequently, it would be in a dictatorial miners best interest to learn the content of transactions instead of relying on the common prior expectation.

Since $\mathcal{Z}$ provides transactions tx_j' issued using the functionality $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ any dictatorial miner is able to learn the content γ_j' associated with tx_j' after at least v rounds after it was issued.[14]. However, $\mathcal{F}_{\mathsf{tl\text{-}TxGen}}$ does not allow any single miner M_i nor an adversary $\mathcal{S}$ that corrupts any subset of miners to learn γ_j' before v rounds. In particular this means that no single miner nor any coalition of miners is able to reduce its uncertainty about γ_j' before v rounds. However, a dictatorial miner M_i could still improve its expected utility by *delaying* every transaction tx_j' it receives in some round r_y for v rounds so it can learn its contents. To this end, the chain reading function $\mathsf{R}'(\cdot)$ checks for every transaction $tx_j' = (\mathsf{txid}_j, \tilde{tx}_j, A_j)$ with associated content $\gamma_j' = (\mathsf{sid}_j, \gamma_j, r_j, P_j)$ included in some block $\mathcal{B}_y^{r_y}$ created in round r_y if $r_y \leq r_j + v$. If not, γ_j' is ignored by $\mathsf{R}'(\cdot)$. Consequently, whenever a dictatorial miner M_i receives a transaction tx_j' in some round r_y for the first time and decides to *delay* this transaction for at least 1 round, it knows that γ_j' associated tx_j' will be ignored by $\mathsf{R}'(\cdot)$. Since any content γ_j' that is ignored by $\mathsf{R}'(\cdot)$ is treated as if the corresponding transaction tx_j' was not included at al *delaying* a transaction yields the same expected utility as *suppressing* it for every dictatorial miner M_i, every transaction tx_j' and every ledger $l_{r_y}' \in \mathcal{L}'$. Since, $\mathcal{Z}$ is expected incentive compatible it can be concluded that every dictatorial miner M_i prefers to *include* any transaction tx_j' in the round it received it first. Therefore, any miner M_i will include any transaction $tx_j' \in \mathsf{X}_i$ into X_i' in any round. Therefore, Π' executed by a set of dictatorial miners $\mathcal{M}$ in presence of an adversary $\mathcal{S}$ driven by an expected incentive compatible environment $\mathcal{Z}$ achieves *rational liveness.* Moreover, let $\mathsf{X}_i^{r_y}$ be the set of transactions in X_i that miner M_i received for the first time in that round r_y. Since for every transactions $tx_j' \in \mathsf{X}_i^{r_y}$ the transaction tag $\tilde{tx}_i$ and the associated account A are chosen uniformly at random and do not reveal any information

[14] Note that, since $\mathcal{Z}$ is expected to provide the inputs to the dictatorial miners the miner M_i is expected to receive any transaction tx_j' in the same round it is issued.

about the associated content γ'_i any dictatorial miner M_i must be indifferent between either including some transaction tx'_1 in some ledger $l'_{round_y}||tx'_0$ or including some transaction tx'_0 in some $l'_{round_y}||tx'_1$ for every pair of transactions $(tx'_0, tx'_1) \in \mathsf{X}_i^{r_y}$ and every ledger l'_{round_y}. Therefore, it can be concluded that Π' also achieves *rational transaction preservation*.

References

1. Abadi, A., Kiayias, A.: Multi-instance publicly verifiable time-lock puzzle and its applications (2021)
2. Badertscher, C., Garay, J., Maurer, U., Tschudi, D., Zikas, V.: But why does it work? A rational protocol design treatment of bitcoin. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 34–65. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_2
3. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: a composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 324–356. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_11
4. Bano, S., et al.: SoK: consensus in the age of blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 183–198 (2019)
5. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: a foundation of time-lock puzzles in UC. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 429–459. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_15
6. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE Computer Society Press (2014). https://doi.org/10.1109/SP.2014.36
7. Bonneau, J.: Why buy when you can rent? In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 19–26. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_2
8. Buterin, V.: Ethereum: a next-generation smart contract and decentralized application platform (2014). https://github.com/ethereum/wiki/wiki/White-Paper
9. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001). https://doi.org/10.1109/SFCS.2001.959888
10. Daian, P., et al.: Flash boys 2.0: frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 910–927. IEEE (2020)
11. Delaune, S., Kremer, S., Ryan, M.: Coercion-resistance and receipt-freeness in electronic voting. In: 19th IEEE Computer Security Foundations Workshop (CSFW2006), p. 12. IEEE (2006)
12. Deuber, D., Döttling, N., Magri, B., Malavolta, G., Thyagarajan, S.A.K.: Minting mechanism for proof of stake blockchains. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020. LNCS, vol. 12146, pp. 315–334. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57808-4_16
13. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-generation Onion Router. Tech. rep, Naval Research Lab Washington DC (2004)

14. Doweck, Y., Eyal, I.: Multi-party timed commitments. arXiv preprint arXiv:2005.04883 (2020)
15. Eskandari, S., Moosavi, S., Clark, J.: SoK: transparent dishonesty: front-running attacks on blockchain. In: Bracciali, A., Clark, J., Pintore, F., Rønne, P.B., Sala, M. (eds.) FC 2019. LNCS, vol. 11599, pp. 170–189. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_13
16. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
17. Judmayer, A., Stifter, N., Schindler, P., Weippl, E.: Estimating (miner) extractable value is hard, let's go shopping! Cryptology ePrint Archive, Report 2021/1231 (2021). https://ia.cr/2021/1231
18. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Chaum, D., et al. (eds.) Towards Trustworthy Elections. LNCS, vol. 6000, pp. 37–63. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12980-3_2
19. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for Byzantine consensus. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 451–480. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_16
20. Khalil, R., Gervais, A., Felley, G.: TEX - A securely scalable trustless exchange. Cryptology ePrint Archive, Report 2019/265 (2019). https://eprint.iacr.org/2019/265
21. Kroll, J.A., Davey, I.C., Felten, E.W.: The economics of bitcoin mining, or bitcoin in the presence of adversaries. In: Proceedings of WEIS, vol. 2013, p. 11 (2013)
22. Liao, K., Katz, J.: Incentivizing blockchain forks via whale transactions. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 264–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_17
23. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Des. Codes Crypt. **86**(11), 2549–2586 (2018). https://doi.org/10.1007/s10623-018-0461-x
24. McCorry, P., Hicks, A., Meiklejohn, S.: Smart contracts for bribing miners. In: Zohar, A., et al. (eds.) FC 2018. LNCS, vol. 10958, pp. 3–18. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-58820-8_1
25. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 31–42. ACM Press (2016). https://doi.org/10.1145/2976749.2978399
26. Nakamoto, S.: Bitcoin: A Peer-to-peer Electronic Cash System. Tech. rep, Manubot (2019)
27. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: how dark is the forest? arXiv preprint arXiv:2101.05511 (2021)
28. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
29. Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P mixing and unlinkable bitcoin transactions. In: NDSS 2017. The Internet Society (Feb/Mar 2017)
30. Tsabary, I., Yechieli, M., Eyal, I.: MAD-HTLC: Because HTLC is crazy-cheap to attack. arXiv preprint arXiv:2006.12031 (2020)
31. Winzer, F., Herd, B., Faust, S.: Temporary censorship attacks in the presence of rational miners. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 357–366. IEEE (2019)