Opportunistic Algorithmic Double-Spending: How I learned to stop worrying and hedge the Fork

Nicholas Stifter[1,2], Aljosha Judmayer[2,1], Philipp Schindler[1,2], and Edgar Weippl[2,1]

[1] SBA Research, Vienna, Austria
(firstletterfirstname)(lastname)@sba-research.org
[2] University of Vienna, Vienna, Austria

Abstract. In this paper we outline a novel form of attack we refer to as Opportunistic Algorithmic Double-Spending (*OpAl*). *OpAl* attacks not only *avoid equivocation*, i.e., do not require conflicting transactions, the attack is also *carried out programmatically*. Algorithmic double-spending is facilitated through transaction semantics that dynamically depend on the context and ledger state at the time of execution. Hence, *OpAl* evades common double-spending detection mechanisms and can *opportunistically* leverage forks, even if the malicious sender itself is not aware of their existence. Furthermore, the cost of modifying a regular transaction to opportunistically perform an *OpAl* attack is low enough to consider it a viable default strategy for most use cases. Our analysis suggests that while Bitcoin's stateless UTXO model is more robust against *OpAl*, designs with expressive transaction semantics, especially stateful smart contract platforms such as Ethereum, are particularly vulnerable.

1 Introduction

Double-spending attacks in cryptocurrencies have primarily been considered in two general categories. In the first category, an adversary is either themselves capable, or is able to coerce others, to participate in an attack that undermines the expected security guarantees of the underlying consensus protocol [73,76,88]. Hereby, attack vectors such as information withholding and information eclipsing [25,35,67,3], as well as exploiting the rational behavior of participants [8,55,41], have received particular attention. The second category of double-spending attacks leverages inadequately chosen security parameters by merchants, i.e, they provide goods or services while the probability of the payment transaction being reverted is non-negligible [44,2,45,76]. In either case, it is generally assumed that the adversary proactively performs a double-spending attack by creating mutually exclusive transactions, i.e., *equivocates* [38].

We hereby challenge this status quo and discuss an alternative attack we refer to as *algorithmic double-spending*, whereby the intent to double-spend is intentionally encoded in the transaction semantics. Algorithmic double-spending does not require equivocating transactions and is facilitated through distributed ledgers that exhibit two properties, namely i) the ability to define *transaction*

semantics that dynamically depend on the ledger state or execution context, which we refer to as *semantic malleability*, and ii) *probabilistic consensus decisions*, i.e., protocols without finality, or where security failures have compromised the *safety*, i.e, consistency [29], of consensus decisions.

1.1 Paper Structure

An introduction and executive summary that outlines the concept of algorithmic-double spending and highlights the contributions of this paper is presented in Section 1.2. In Section 1.3 we discuss related work and background literature. Section 2 provides a definition of what is meant by (algorithmic) double-spending. A proof-of-concept *OpAl* attack in the context of Ethereum is presented in Section 3, to highlight the concept of algorithmic double-spending by example. To gain a better understanding of the principles behind *OpAl*, we first define prerequisites and properties of semantic malleability in Section 4, and use them to investigate three different ledger designs in Section 5. Finally, we consider possible mitigation strategies against algorithmic double-spending (Section 6) and highlight future research directions in Section 7.

1.2 Algorithmic Double-Spending in a Nutshell

In this work, we are the first to describe and analyze the unique category of (opportunistic) algorithmic double-spending, which has been largely overlooked as a potential attack vector. The relevance of this novel attack lies, on the one hand, in its automatic execution upon inclusion in a fork, its conceptual simplicity, its evasion of detection strategies, and its plausible deniability. On the other hand, the opportunistic nature of *OpAl* attacks also presents a more serious threat in case the underlying security assumptions of the ledger do not hold in practice, even if the cause of failure itself is benign. In light of any deep fork that extends beyond the assumed k-block common prefix [30,69], replaying transactions on a different branch risks inadvertently triggering an embedded *OpAl* attack.

The characteristic of *OpAl* attacks is comparable to logic bombs found in malware, which can exhibit time- or state-dependent behavior [14,28]. Future detection and prevention mechanisms against *OpAl* may hence need to perform in-depth transaction analysis and classification techniques, e.g. [84,36], or attempt to identify abnormal behavior during on-the-fly analysis [26].

Regarding the previously outlined two general categories of double-spending, *OpAl* attacks are applicable in both scenarios. We note that *OpAl* attacks can be used as a *drop-in replacement* for classical, non-opportunistic equivocation-based attacks. Thus, in this context, algorithmic double spending appears to present a superior strategy. To highlight the practicability of *OpAl*, we demonstrate that Ethereum transactions can easily be augmented to perform such an attack. *OpAl* itself does not affect the success probability of attacks targeted at *causing* the required blockchain fork for performing a double-spending attack. However, it can be easily used as a default strategy in transactions to leverage upon random

forking events or unknown attacks. We hence refer to this form of attack as *opportunistic* algorithmic double-spending (*OpAl*).

OpAl attacks do not stand in contradiction to the security guarantees and desirable properties [30,69,6] offered by Nakamoto-style distributed ledgers. The existence of state instability through forks is abstracted away in *idealized ledgers* by waiting for the relevant actions, e.g. transactions, to be included in the common prefix [9]. Assuming the security guarantees hold, algorithmic double-spending is primarily of concern in cases where users exhibit an insecure interaction model referred to as *hasty players* [9], whereby actions are taken based on unstable state. We crucially note that this pattern is commonly encountered in real-world ledgers, in particular in regard to applications such as *decentralized finance* (DeFi), where hastiness can be financially advantageous [15,90,91].

Our analysis of the governing mechanism for algorithmic double-spending suggests that semantic malleability lies at the core of the problem. Semantically malleable transactions allow for different state transitions depending on the input state and execution environment at the time of processing. It may appear that the solution to this problem is to enforce a single valid state transition for a transaction, such as the EUTXO model [12] employed by Cardano. However, in this case the possibility of algorithmic double-spending still arises if the *validity* of a transaction can be tied to particular ledger states. Interestingly, in the highly limited UTXO model of Bitcoin [4], access to ledger state appears sufficiently constrained to prevent practicable *OpAl* attacks.[3]

While there appear to be some mitigation strategies against *OpAl*, it is unclear if the underlying issue can be completely avoided in practice. One possible defensive approach against *OpAl* attacks is to prevent players from concurrently interacting with malleable ledger state until it has sufficiently stabilized, however such a pattern may not be desirable for users, as it can lead to long waiting periods. In this regard it appears advantageous to be able to achieve fast and guaranteed *consensus finality*, which remains an active research topic for decentralized ledger designs [11,70,68]. Finally, if the security assumptions of the ledger fail due to attacks or technical errors, which practice has shown can happen [61,56,64], *OpAl* attacks can prove particularly severe and superior to equivocation-based techniques. Hence, a *hedge* against such scenarios might be for oneself to proactively engage in *OpAl* (counter)attacks.

1.3 Related Work

Beside the related work on double-spending that we mention in the introduction, it is important to note that prior art has identified a range of security issues in distributed ledgers that tie-into the discussion of *OpAl*, e.g., *timestamp- and transaction-order dependence* [57], *concurrency* and *event ordering* (EO) vulnerabilities [75,50], *blockchain anomalies* [66], *stake bleeding* [31], *time-bandit* [15] attacks, and *order-fairness* [47,53]. We outline several of these works in detail

[3] Transaction inputs from a coinbase transaction in a fork in excess of 100 Blocks can enable a form of *OpAl* attack that does not require equivocation (see Section 5).

within the body of this paper. To the best of our knowledge, we are the first to present the concept of algorithmic double-spending and demonstrate its practicability. Conceptually, Botta et al. [9] relates most to the topics discussed within this work. They effectively highlight the possible effects of blockchain forks, as well as the practical implications of probabilistic finality with *hasty players*, in the context of MPC protocols executed atop distributed ledgers. However the concept of algorithmic double-spending is not considered.

2 What is Algorithmic Double-Spending?

In this section we define our notion of double-spending and put forth the argument that there exists the overlooked class of algorithmic double-spending, which does not necessitate conflicting actions, i.e., equivocation. We then discuss the implications of this insight, such as the possibility of *unintentional* double-spending, and raise the question whether double-spending requires economic damage.

2.1 Defining (Algorithmic) Double-Spending

We observe that while research on double-spending provides concrete descriptions and formal analyses of particular instantiations of double-spending attacks, e.g., [44,2,45,37], a general definition of the term double-spending appears to be outstanding. A clearer definition of what is meant in this regard may not only aid with classification efforts, but could also help identify new or overlooked attack forms. Motivated by the novel class of algorithmic double-spending we present within this work, we hereby set out to propose such a more general definition:

Definition 1 (Double-Spending Attack). *In a double-spending attack an adversary attempts to fool a victim into performing an economic transaction directed at the adversary on the basis of a presumed valid system state, which is later revealed to be stale or invalid. Hereby, the adversary's goal is to be able to reuse any of the resources that form the basis of the economic transaction for other purposes. We distinguish between the following Double-Spending Attacks:*

- ***Equivocation-Based****, whereby the adversary issues multiple conflicting actions in the system, one of which is aimed at fooling the victim, and where at most one of the issued actions can eventually be performed in the system.*
- ***Algorithmic****, whereby the adversary performs a single action that can have different semantic meanings, depending on the system state in which they are interpreted, and where the interpretation of this action in some stale or invalid system states can be used to fool the victim.*

At the core of this work lies the insight that double-spending may be facilitated through other means than the classical notion of requiring equivocation-based conflicting actions of an adversary. Algorithmic double-spending builds on a simple property, that can be observed in various distributed ledger designs with expressive transaction semantics today:

Observation 1 (semantic malleability) *Given a transaction t, it may have different semantic outcomes, depending on the ledger state and environment upon which t is executed.*

We refer to this property as *semantic malleability* due to the fact that external factors, such as the consensus protocol and its ordering guarantees [47,89,53,46], as well as other actors in the system who may be *rushing* [51], e.g., in the context of frontrunning [22,15,91], are able to transition the system state in a way that is able to affect or *malleate* the semantics of transactions.

From Observation 1 we can rather intuitively derive a basic strategy for an algorithmic double-spending attack: an adversary can encode both, the regular payment to the merchant, as well as an alternative malicious action, e.g., payment to herself, as different execution paths within a single transaction. The control flow of the transaction is designed to conditionally branch, depending on the ledger state σ at the time the transaction is processed by a miner. If the same transaction is included in a different state σ', i.e. a fork, the "hidden" algorithmic double-spend is triggered without active participation from the attacker. Figure 1 illustrates this difference to equivocation-based double-spending.

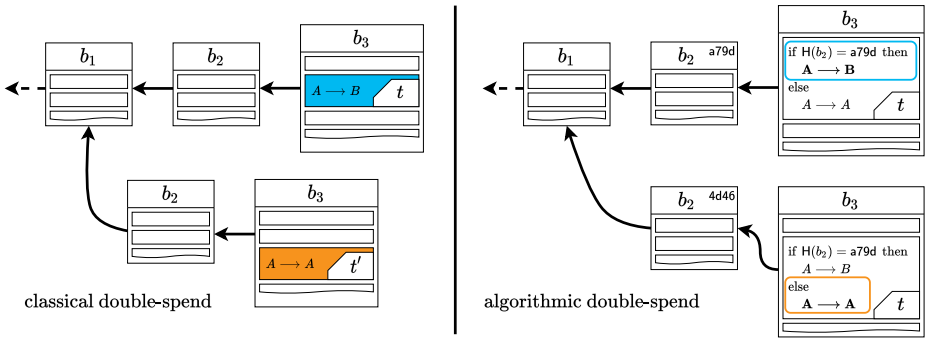

Fig. 1. Conceptual difference between equivocation- and algorithmic double-spending

2.2 Novel Insights inspired by Algorithmic Double-Spending

The concept of algorithmic double-spending raises interesting challenges, two of which we outline in more detail. First, up until now *unintentional* double-spending, for example as a result of technical failures, did not appear of particular concern. Prior art identifies potential vulnerabilities that arise from order

dependence in smart contracts [57,75,50] and violations of transaction causality in forks with unintended side-effects [66]. We expand upon these insights by highlighting that semantic malleability can lead to unintentional algorithmic double-spending as a result of transaction reordering within a blockchain fork. Hereby, it can be difficult to distinguish between misbehavior and misfortune.

Second, in stateful smart contract systems double-spending may not only be performed solely at the economic level by reusing some virtual coin. For example, Botta et al. [9] highlights the need of mitigation strategies against an adversary leveraging forks in MPC protocols with hasty players. On the other hand, Zhou et al. [90] identify network flooding events with equivocating, almost identical, transactions. Similarly, increasing the miner fee of a transaction may require a user to equivocate, raising the question if such behavior should be subsumed under the notion of double-spending. This presents the interesting problem how any divergent system behavior within forks, be it through equivocation- or algorithmic double-spending, should be addressed.

3 A Proof-of-Concept (PoC) *OpAl* Attack in Ethereum

We demonstrate the practicality of algorithmic double-spending by providing a functional PoC *OpAl* attack in Ethereum. Our attack design is inspired, on the one hand, by *hardfork oracles*, which McCorry et al. [61] discusses in the context of atomic-trade protocols during hardforks, and, on the other hand, by the notion of *context sensitive transactions* Gaži et al. [31] describes as a replay protection mechanism in *stake-bleeding attacks*. An informal statement that encapsulates the intended transaction semantics for our PoC *OpAl* attack is the following:

> "IF *this transaction is included in a blockchain that contains a block with hash* $0xa79d$ THEN *pay the merchant,* ELSE *don't pay the merchant."*

Essentially, our attack is based on the insight that a transaction can act as its own *fork oracle* for conditionally branching its execution. In the following, we first outline the construction of such a fork oracle in more detail and then present a PoC attack that allows transactions with the above semantics to be created.

3.1 How to Construct an *OpAl* Fork Oracle in Ethereum

The concept of employing a fork oracle to distinguish between branches of (hard)forks was proposed in cryptocurrency communities [59,23], as well as research [61,62,40]. Hereby, a frequent goal is achieving *replay protection*. McCorry et al. [61] outlines how fork oracles can be leveraged to realize atomic trades across hardforks. Constructing a smart-contract based fork oracle if the underlying forks do not offer replay protection can be challenging [61]. McCorry et al. [62] demonstrate through *history revision bribery* how (equivocation-based) double-spending can be leveraged to realize a fork oracle. Hereby, the fork oracle is not used to facilitate (algorithmic) double-spending. Rather, the mutually exclusive outcomes of the double-spend in different forks are relied upon to actually

implement the oracle. Surprisingly, the idea of using fork oracles to *algorithmically* trigger double-spending was not yet considered.

Block-Hash Based Fork Oracle The fork oracle we propose is inspired by a simple and elegant technique to achieve replay protection that has been considered in the proof-of-stake setting [54,31]. Hereby, the hash of a recent block is included in a transaction, and the transaction is only considered valid for blockchains that contain this block in their prefix. Gaži et al. [31] refer to this mechanism as *context sensitive transactions*. Essentially, context sensitive transactions already implicitly realize the attack semantics described above.[4] In case a fork of sufficient depth occurs, this replay protection mechanism ensures that transactions become invalid at the protocol level, and the double-spending "attack" is realized algorithmically through the underlying protocol rules. Ethereum does not natively support context sensitive transactions, however, this functionality can easily be emulated with smart contract code because of available EVM primitives that expose ledger context, such as the BLOCKHASH opcode. [83] It is hence possible to programmatically act upon the existence of a particular block, or other ledger context, as part of an Ethereum transaction.

Fork Oracle Discussion A downside of the hash-based fork oracle is its reliance on a commitment to *previous* ledger state, thereby requiring a fork of at least depth-2 to trigger the attack. However, it is also possible to construct oracles that enable *OpAl* attacks for forks of depth-1. The key difference between a depth-1 fork oracle and a hash-based fork oracle in the above design is, that the latter is based on ledger state which is *known*, whereas the former is based on some *prediction* of the future state at the time the transaction is processed. Hence, depth-1 fork oracles generally offer much weaker probabilistic guarantees for identifying forks. For example, consider the COINBASE opcode in the EVM that returns the *current block's* beneficiary address [83]. Instead of specifying the highest known block hash as the branching condition, an adversary could use the beneficiary address of a large mining pool[5] in an *OpAl* attack. Hereby the transaction semantics depend on whether the transaction is included in a block from the targeted mining pool, or some other miner. Generally speaking, in Nakamoto-style proof-of-work ledgers the next block producer is not known in advance. However, we note that in future proof-of-stake protocols [7,16] this may be different, thereby allowing for more reliable depth-1 fork oracles.

Another limitation of the hash-based fork oracle specific to the EVM is the restriction that the BLOCKHASH opcode only returns hashes within a 256 block look back window, and 0 otherwise. [83] Hence, if a transaction is processed in a block that exceeds 257 blocks after the height of the blockhash commitment, the oracle will falsely report a fork and trigger the attack branch. We argue that in the case of our intended *OpAl* semantics this limitation is unproblematic, as the transaction would simply transfer the funds back to the attacker.

[4] Thereby introducing the possibility of *unintentional OpAl* attacks (see Section 2.2).
[5] We note that in Ethereum, address reuse in the coinbase by miners is prevalent.

3.2 Proof of Concept *OpAl* Attack Contract

Di Angelo and Salzer present a comprehensive empirical analysis of wallet contracts on Ethereum [18,19]. Of the identified properties, in particular designs that support *flexible transactions*, i.e., forwarding of arbitrary calls, appear suitable for augmentation to support the creation of *OpAl* transactions. Their empirical data shows that at least *tens of thousands* of contracts supporting flexible transactions are currently deployed in Ethereum, suggesting practical use-cases for such contract patterns, even without an *OpAl* augmentation. Our attack requires minimal modifications, and the interaction pattern is almost the same.

In the following, we present a minimal *fully viable PoC OpAl attack smart contract* written in Solidty [82], that relies on the aforementioned hash-based fork oracle. Our contract code (Listing 1.1) is loosely based on the Executor contract from the Gnosis-Safe Wallet [63], which allows the forwarding of arbitrary function calls. Instead of forwarding a call directly, the contract first evaluates if the block hash at a particular height of the current ledger matches the commitment hash that was provided as an additional parameter in the transaction data. This is realized through the *blockhash()* function [24]. If the blockhash matches the commitment, the function call is forwarded, else, no action is performed, i.e., the action is reversed in a fork.

```
pragma solidity 0.8.4;
// This contract acts as an OpAl forwarding proxy for transactions.
contract Opal {
  address public owner;

  modifier onlyOwner() {
    require(isOwner(msg.sender));
    _;
  }
  constructor() {
    owner = msg.sender;
  }

  fallback() external payable {}
  receive() external payable {}

  function isOwner(address addr) public view returns(bool) {
    return addr == owner;
  }

  function cashOut(address payable _to) public onlyOwner {
    _to.transfer(address(this).balance);
  }

  // forwarding function implementing opportunistic double-spending (OpAl)
  function forward(address payable destination, bytes32 commitblockHash,
                   uint commitblockNumber, bytes memory data)
                   onlyOwner public payable returns(bool success) {
    if (blockhash(commitblockNumber) == commitblockHash)
      assembly { success := call(gas(), destination, callvalue(),
                                 add(data, 0x20), mload(data), 0, 0)
      }
  }
}
```

Listing 1.1. Solidity *OpAl* contract that implements a basic fork oracle by only forwarding transactions if the provided commitment to a block hash can be resolved.

Outline of the Attack An adversary wishing to engage in *OpAl* first needs to deploy the attack contract. Once the contract is successfully deployed, whenever they wish to perform a transaction with *OpAl* functionality, instead of calling a function $f()$ in the target contract or sending funds directly, they simply forward this call to the *forward()* function (Line 15 in Listing 1.1) of the deployed attack contract, together with the appropriate parameters. Specifically, the adversary generates transaction t that calls *forward* in the attack contract with the following parameters: i) the target address; ii) the block hash and height h of the current chain tip; iii) the encoded function name to be called at the target $f()$ together with its parameters; iv) any Ether that shall be sent; and broadcasts t to the network. Ideally, the transaction fee is high enough for t to be immediately included in the next block $h+1$. Otherwise, the required fork depth increases in the number of blocks the chain grows between the creation and inclusion of t.

To the recipient of t, the interaction pattern will appear as if the user employed a regular wallet contract. Unless they perform an analysis of the deployed contract bytecode, the malicious behavior only becomes apparent once the attack conditions are triggered, i.e., during a fork. In case the adversary is lucky and a fork at, or before, height h occurs, and their transaction is replayed within this fork, the alternative attack branch of the contract is executed automatically.

3.3 Cost Overhead of PoC Attack in Ethereum

We quantify the additional costs incurred when augmenting a transaction with *OpAl* capabilities, by deploying our attack contract in a private Ethereum testnet and measuring the gas utilization for basic interactions, such as ERC-20 token [80] transfers. Our PoC *OpAl* attack adds a constant overhead of gas that depends on the number of parameters supplied to the target function $f()$. The deployment transaction for the contract in Listing 1.1 required 393175 gas. As this transaction can be done well in advance of any attacks and needn't be timely, we assume a lower gas price of 75 GWei, which translates to deployment costs of ≈ 0.03 Ether or, at an exchange rate of 3400 USD, almost exactly 100 USD. Note that the attacker only needs to deploy this contract once, after which the only overhead derives from using the forwarding function. For ERC-20 token interactions (*approve, transfer, transferFrom*), using *OpAl* adds $\approx$ 3000 gas, which equates to $\approx 8\%$ overhead. At the time of writing, assuming a gas price of 150 GWei for timely inclusion of the transaction, this overhead translates to ≈ 1.5 USD higher fees if a transaction is augmented to support *OpAl* attacks.

4 Prerequisites for Algorithmic Double-Spending

Within this section we identify prerequisites and underlying properties that enable algorithmic double-spending. Our analysis is based on an intentionally simple system model to account for different ledger designs. We define the concept of *semantic malleability* that we introduced in Section 2 and argue that ledgers

with semantically malleable transactions are vulnerable to algorithmic double-spending, and thus *OpAl* attacks. In our analysis we show that any distributed ledger that is robust to semantic malleability must satisfy two necessary properties, namely *eventual replay validity* and *replay equivalence*. Finally, we raise the question whether a characterization of Nakamoto-style ledgers as replicated state machines (RSM) is accurate in light of algorithmic double-spending.

System Model and Assumptions: Following Luu et al. [57], we conceptually view a blockchain as a transaction-based RSM, where its state is updated after every transaction. We denote $\mathcal{S}$ the set of all possible system states and $\sigma \in \mathcal{S}$ a single system state. The initial starting state of a blockchain is defined as σ_0. A valid transition from state σ to σ', via transaction t, is denoted as $\sigma \xrightarrow{t} \sigma'$. $\text{PAST}(\sigma_n)$ is defined as the ordered list of transactions $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, that, when applied to σ_0, lead to state σ_n. If there exists a non-empty sequence of transactions starting from state σ_a to state σ_b, we call σ_a a predecessor of σ_b, in short $\sigma_a \prec \sigma_b$. The predicate $\text{VALID}(t, \sigma)$ represents the transaction validation rules of the underlying protocol, it returns TRUE if and only if the transaction t is considered valid (executable) in state σ. We assume that block producers, e.g., miners, adhere to protocol rules and *transaction liveness* is guaranteed, i.e., any valid transaction will eventually be executed.

Executing a transaction t in state σ alters (part of) the state σ and thus results in a new state σ'. The changes are captured by the function $\text{DIFF}(t, \sigma)$. Consider, for example, a state $\sigma = \{\text{Alice: } 6, \text{Bob: } 5, \text{Carol: } 4\}$ represented as account-value mapping, and a transaction t, where Alice gives 2 coins to Bob. Then $\text{DIFF}(t, \sigma) = \{\text{Alice: } -2, \text{Bob: } +2\}$ captures the balance changes of Alice and Bob while other parts of the state (Carol's balance) remain unaffected. In this example a single account-value mapping is called a *substate*. Note that it is possible that the effects of executing the same transaction t in two different states are equal, i.e., $(\sigma_a \neq \sigma_b) \wedge (\text{DIFF}(t, \sigma_a) = \text{DIFF}(t, \sigma_b))$.

Identification of Prerequisites: We consider a transaction t to be a sequence of operations (computations) that lead to a state transition. A transaction is *semantically malleable*, if the available primitives, which are used to define the semantics of the transaction, allow the control flow of the execution to branch conditionally based on the particular input state σ.

The following two properties we define are necessary for a ledger to be robust against semantic malleability. We refer to these properties as *replay equivalence* and *eventual replay validity*, since replaying the same ordered set of transactions on some initial state σ_0 should always yield the same state transitions and final state, and the validity of transactions should not be affected by the environment.

Definition 2 (replay equivalence). *Assuming that no transaction equivocation happens: A transaction t satisfies replay equivalence, if executing t in all candidates states where t is executable (valid) leads to the same changes in the*

respective (sub)states:

$$\forall \sigma_a, \sigma_b \in \mathcal{S}, \quad (\textsc{valid}(t, \sigma_a) \wedge \textsc{valid}(t, \sigma_b)) \implies (\textsc{diff}(t, \sigma_a) = \textsc{diff}(t, \sigma_b)) \,. \tag{1}$$

Definition 3 (eventual replay validity). *Assuming that no transaction equivocation happens: If a transaction t is found executable (valid) in some state σ_a, then it either remains executable (valid) or has already been executed in successor states of σ_a:*

$$\forall \sigma_a, \sigma_b \in \mathcal{S}, \quad (\textsc{valid}(t, \sigma_a) \wedge \sigma_a \prec \sigma_b) \implies (t \in \textsc{past}(\sigma_b) \vee \textsc{valid}(t, \sigma_b)) \,. \tag{2}$$

Definition 4 (semantic malleability). *A transaction t is semantically malleable if it violates the replay equivalence and/or the eventual replay validity property.*[6]

4.1 Can Blockchains be Characterized as State Machines?

State machine replication is generally based on the notion that the state of the system is solely determined by the sequence of (deterministic) operations it has processed, independent of external factors. Interestingly, we observe (Section 5) that *in practice*, designs appear to deviate from the model we adopt from Luu et al. In his seminal work on the state machine approach F. B. Schneider provides the following semantic characterization of a state machine [74]:

> "Outputs of a state machine are completely determined by the sequence of requests it processes, independent of time and any other activity in a system."

First, consider the herein discussed property of semantic malleability in transactions. The existence of semantic malleability in itself does not violate the above characterization, as a mere reordering of transactions, i.e., requests, may lead to semantic malleability without requiring any access to time or activity within the system. However, in practice, ledger designs often allow transaction semantics to depend on external ledger context that is not solely defined by such requests, i.e., *time* or other *external data from blocks* (See Section 3). In essence, being able to define functions that take as input elements of the ledger context within transaction semantics, such as previous block hashes, the block height, coinbase transaction or block time, can cause a violation of replay equivalence or eventual replay validity, both of which can be directly derived as required properties of a RSM from the above characterization.

Second, blockchain designs generally offer *rewards* as an incentive mechanism for block producers to participate in the consensus protocol. Under the assumption that a block merely represents an ordered set of transactions, i.e., requests,

[6] A equivalent standalone definition of this property is given in Appendix B.

and transactions can not access any external state defined within blocks, this model would appear to realize a RSM. However, if we include the fact that block rewards represent transactions or state transitions that depend on a particular external state, namely the block itself that justifies the reward, the model is no longer independent of the system state.

We note that one possibility to amend this issue is to either include the creation of blocks as requests, or model state updates entirely from the perspective of blocks and not at the transaction level. The latter approach is, for instance, taken by formal models that analyze Nakamoto consensus [30,69,6]. Nevertheless, even if one considers state machine replication only from the perspective of blocks and not individual transactions, there can still exist external dependencies on the environment, in particular on time. Consider that receiving late or early blocks may render them (temporarily) invalid by the protocol rules, leading to different possible interpretations of the same sequence of requests and resulting final state depending on the current system time.

Observation 2 *If operations, i.e. transactions, in a blockchain either depend on- or have access to- the execution context, e.g., time, block hashes, block height, coinbase transactions, etc., then the resulting system does not adhere to the semantic characterization of a replicated state machine.*

5 Do Bitcoin, Ethereum, and Cardano Achieve Eventual Replay Validity and Replay Equivalence?

For the following investigation we set aside the orthogonal topic of *how* to create blockchain forks of sufficient depth to facilitate double-spending attacks. Instead, we are interested in identifying if, in principle, the designs are vulnerable to semantic malleability, by evaluating whether the aforementioned necessary properties are satisfied. We consider Bitcoin, Ethereum and Cardano, as they each represent instantiations of Nakamoto-style blockchains with distinct design differences. Bitcoin [65] is UTXO-based and facilitates a highly limited, non-Turing complete scripting language for transaction semantics. [4] Ethereum [83] adopts an account-based model and offers expressive transaction semantics that can draw upon stateful Turing-complete smart contract functionality. Finally, Cardano [13] is set to adopt the EUTXO model [10], which intends to leverage advantages of a stateless UTXO design with the expressiveness of Turing-complete smart contracts that can carry state.

Bitcoin: In Bitcoin, transactions are based on the so-called *unspent transaction outputs* (UTXO) model [17] and contain simple (deterministic) Boolean functions, called *Scripts*, that determine the transaction semantics.[4] Bitcoin's UTXO model is stateless and non-Turing complete. A key aspect of the UTXO model is that transactions are deterministic and bound to a single execution by committing to the exact input (sub)states, i.e., UTXOs, that a transaction consumes, and a precise set of output UTXOs, that the transaction produces.

We now informally analyze whether Bitcoin's UTXO model appears to satisfy replay equivalence and eventual replay validity. Consider a sequence of transactions $\mathcal{T} = \{t_1, t_2, \ldots t_n\}$ in some valid Bitcoin blockchain $\mathcal{B}$. If $\mathcal{T}$ is replayed in another valid blockchain $\mathcal{B}'$, and both $\mathcal{B}$ and $\mathcal{B}'$ have the same starting state σ_0, then: i) for *replay equivalence* to not hold, there must exist a transaction t_v in $\mathcal{T}$ where the resulting state transition differs between $\mathcal{B}$ and $\mathcal{B}'$. In the UTXO model, there exists precisely one valid input UTXO and one valid output UTXO for every transaction. Hence, for t_v to result in a different state transition, it would have to be invalid, If t_v is invalid, yet included in $\mathcal{B}'$, it would mean $\mathcal{B}'$ is also invalid – a contradiction. Hence, it appears that Bitcoin's UTXO model satisfies *replay equivalence*.

ii) regarding *eventual replay validity*, for a transaction to be included in a blockchain $\mathcal{B}'$, the input UTXO the transaction consumes must be present. Given $\mathcal{B}$ and $\mathcal{B}'$ have the same σ_0 and transactions are executed in the same order, executing t_1 on σ_0 in either $\mathcal{B}$ or $\mathcal{B}'$ yields the same resulting state σ_1. By induction, this holds for every transaction t_i in $\mathcal{T}$, unless t_i also has some *external context dependency* on the environment of $\mathcal{B}$, in addition to the output state generated by t_{i-1}. In Bitcoin, external context (e.g. time) is not accessed explicitly through a UTXO, but implicitly through Scripts or validity rules for transactions. Specifically, it is possible to define some relative or absolute time, in relation to that of the ledger context, from which point onward a transaction may become *valid*. [78] However *it is not possible to permanently invalidate a transaction that depends on ledger context*, i.e., in a live blockchain there is a future point and time where this dependency is satisfied. We therefore conclude that, in principle, the Bitcoin UTXO model could satisfy eventual replay validity.

However, we have excluded the notion of *coinbase transactions* from our analysis, i.e., reward transactions to block producers, whose validity depend on the block in which they were created and therefore implicitly on blockchain $\mathcal{B}$. Hence, eventual replay validity is not satisfied by coinbase transactions.

As a tangible attack example, consider a transaction which includes, as one of its input UTXO, an output from a coinbase transaction that has become spendable, i.e., has matured for 100 Blocks. If a sufficiently deep blockchain fork, of say 144 blocks, occurs, the above transaction can not be replayed within a fork and becomes invalid, thereby facilitating an algorithmic double-spend that does not require equivocation. Karame et al. [45] also outlines a form of double-spending attack in Bitcoin that leverages the different interpretations of validity for a transaction in a softfork [86], however their attack is based on equivocating transactions.

Cardano: Cardano [13] is based on a line of research on provably secure proof-of-stake Nakamoto-style blockchains [49,16,5,48], which we subsume under the term *Ouroboros*. Hence, Ouroboros also offers probabilistic finality guarantees and the existence of temporary blockchain forks is possible.

Cardano is poised to adopt the *Extended UTXO* (EUTXO) model [12,10], that was conceived to leverage desirable properties of Bitcoin's UTXO design. [12] Conceptually, to support stateful Turing-complete smart contracts in EUTXO,

the UTXO model is extended in the following (from Chakravarty et al. [12]): i) outputs can contain arbitrary contract-specific data; ii) Scripts, which are referred to as *validators* in the EUTXO model, receive the entire transaction information, including its outputs, as *context* next to the contract specific data, and can impose arbitrary validity constraints on the transaction; iii) a *validity interval* is added for transactions, whereby any Scripts which run during validation can assume that the current time is within that interval;

A key property the EUTXO model inherits from the UTXO model is that the execution of a transaction during validation is entirely deterministic and solely determined by its inputs. Equivocation is hence required to achieve a different semantic result. In terms of our necessary properties to achieve robustness against semantic malleability, *replay equivalence* follows analogous to Bitcoin.

However, as Brünjes and Gabbay [10] crucially point out, the EUTXO model allows to restrict the validity of transactions to time intervals, which renders the result of transaction processing dependent on the ledger context. Unlike Bitcoin, in Cardano transactions can be permanently invalidated based on ledger context. Hence, *eventual replay validity* is not satisfied and semantic malleability possible. In particular, a transaction's validity interval can be used as a weaker fork oracle to trigger *OpAl* attacks where the transaction is reverted if a fork occurs, and it is not included before the validity interval expires. We refer the interested reader to Appendix C for a visualization of this attack.

Ethereum: We refer the reader to Section 3 for a practical example of an *OpAl* attack in Ethereum. Nevertheless, we briefly also provide an example why Ethereum's design is vulnerable to semantic malleability. The EVM offers primitives that can be used to query information from the ledger context at the time of execution, e.g., opcode $0x42$, which is defined in the Ethereum Yellowpaper [83] to place the current block's timestamp onto the stack. This implies that a transaction whose execution relies on such opcodes can semantically differ, depending on the block in which it is included. Hence, *replay equivalence* is not satisfied.

6 Mitigation Strategies against *OpAl*

Having outlined the principles behind algorithmic double-spending, we now discuss possible prevention or mitigation strategies. Hereby, we broadly distinguish between two categories, namely approaches that seek to limit the effects of *semantic malleability* and those that address instability in consensus, i.e., a *lack of finality*. Finally, a questionable course of action can also be to oneself engage in *OpAl* attacks, in order to reduce counterparty risk and try to hedge against the potentially detrimental effects of any deep blockchain fork, should it ever occur.

Mitigating Semantic Malleability: As we have shown in Sections 2 and 4 semantic malleability lies at the core of enabling algorithmic double-spending. In this regard we believe that the expressive transaction semantics associated with smart-contract functionality poses a fundamental challenge when trying to

combat algorithmic double-spending. Drawing upon the concept of *guard functions* from Luu et al. [57] and *context sensitive transactions* Gazi et al. [31] and Botta et al. [9] rely on, transaction validity of transactions should more explicitly be constrained to input states that only lead to desirable outcomes for the sender. While such patterns do not prevent the possibility of algorithmic double-spending, they can avert that a user's transaction executes in a state that leads to an undesirable outcome. In light of recent research in regard to *order-fairness* in consensus [47,89,53,46], the aforementioned pattern could also help to mitigate the potential negative impact of malicious orderings.

Another mitigation strategy is through the analysis and classification of transaction semantics in order to try and identify potential threats. Hereby, the challenges lie on the one side, in finding efficient techniques for static and dynamic code analysis that can be applied, in real-time, to identify potentially malicious transactions before they are processed, and on the other side, in how to define what is considered malicious behavior and also enforce any transaction rejection policies within decentralized systems. [72,81,26,84,87,36,27]

Mitigating OpAl through stronger consensus guarantees: Essentially, the majority of distributed ledgers rely on *consensus*[7] to agree upon the order of transactions among participants, in order to prevent double-spending. [34]

Our Definition 1 of double-spending highlights the need of some stale or invalid system state in order to fool a victim. The existence of hasty players, that are willing to act on such state, renders double-spending attacks feasible in practice, even if the consensus protocol in principle provides strong guarantees against it. In this regard, effective mitigation strategies to combat double-spending may entail a stricter enforcement of safe interaction patterns in client software and cryptocurrency wallets, and a better understanding of the behavior and mental models of cryptocurrency users. [21,52,60]

However, if the security assumptions of the underlying system are compromised, in particular Nakamoto-style distributed ledgers can suffer from deep forks where previously assumed stable ledger state is reverted. Aside from the potential of targeted attacks against the protocol [3,79], technical failures[8] can also lead to such a violation of the security assumptions. [61,56,64]

Notice that in this regard there is a crucial difference between *OpAl* and equivocation-based double spending. In the latter, an adversary has to actively monitor the network for forks and disseminate conflicting double-spending transactions that are at risk of being easily detected and prevented at the peer-to-peer layer [44,32]. *OpAl* attacks and algorithmic double-spending, on the other hand, may prove particularly severe. Any transaction that was included in a blockchain that is replayed on a fork faces the risk of triggering a hidden *OpAl* attack. If

[7] An interesting recent result in this regard is presented in Guerraoui et al. [34], which proves that for simple asset transfer consensus is, in principle, not necessary for double-spending prevention.

[8] We note that scheduled protocol updates carry a risk of unintentional forks, and an adversary may try to leverage this by performing *OpAl* transactions at that time.

a fork in excess of k blocks occurs, *OpAl* attacks which are triggered have a high probability of success. A possible mitigation strategy to limit the effects of *OpAl* in deep forks is the utilization of checkpointing [43]. Another line of research seeks to strengthen the guarantees of Nakamoto-consensus by achieving consensus finality [11,71,20,68]. It may also be preferable to sacrifice *liveness* by halting execution rather than risking systemic risk through *OpAl* attacks.

7 Conclusion

We have described and analyzed a novel class of double-spending attacks, called (opportunistic) algorithmic double spending (*OpAl*), and shown that *OpAl* attacks can readily be realized in stateful smart contract platforms by presenting a proof-of-concept implementation for EVM-based designs. *OpAl* itself does not increase the likelihood or severity of blockchain forks, which are a prerequisite for most double-spending attacks. Instead, *OpAl* allows regular transactions performed by *anyone* to opportunistically leverage forking events for double-spending attacks. Hereby *OpAl* evades common double-spending detection strategies and offers a degree of plausible deniability. A particularly worrying property of *OpAl* is the ability for already processed transactions to trigger hidden double-spending attacks whenever they are replayed in a fork. Attacks or technical failures that lead to deep forks may hence pose an even greater systemic risk than previously assumed. It would appear that the most promising mitigation strategy against *OpAl* is achieving fast consensus finality, combined with avoiding transaction semantic malleability.

We believe that the introduction of algorithmic double-spending as a novel attack category opens up new research directions and highlights the interconnectedness of many important insights in the domain of distributed ledgers. The advent of expressive smart contract systems has created a vast new range of exciting use-cases, but with them also come novel security challenges [58,42,15,39,91] that need to be thoroughly addressed.

Acknowledgment

This material is based upon work partially supported by (1) the Christian-Doppler-Laboratory for Security and Quality Improvement in the Production System Lifecycle; The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the Nation Foundation for Research, Technology and Development and University of Vienna, Faculty of Computer Science, Security & Privacy Group is gratefully acknowledged; (2) SBA Research; the competence center SBA Research (SBA-K1) funded within the framework of COMET Competence Centers for Excellent Technologies by BMVIT, BMDW, and the federal state of Vienna, managed by the FFG; (3) the FFG Bridge 1 project 864738 PR4DLT. (4) the FFG Industrial PhD project 878835. (5) the FFG ICT of the Future project 874019 dIdentity & dApps. (6) the European

Union's Horizon 2020 research and innovation programme under grant agreement No 826078 (FeatureCloud). We would also like to thank our anonymous reviewers for their valuable feedback and suggestions.

References

1. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D.: Uniswap v3 core. Tech. rep., Tech. rep., Uniswap (2021)
2. Androulaki, E., Capkun, S., Karame, G.O.: Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin. In: CCS (2012), `http://eprint.iacr.org/2012/248.pdf`
3. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: Routing attacks on cryptocurrencies. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 375–392. IEEE (2017)
4. Atzei, N., Bartoletti, M., Lande, S., Zunino, R.: A formal model of Bitcoin transactions. In: Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC). Springer (2018), `http://fc18.ifca.ai/preproceedings/92.pdf`
5. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 913–930 (2018)
6. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a Transaction Ledger: A Composable Treatment (2017), `https://eprint.iacr.org/2017/149.pdf`, published: Cryptology ePrint Archive, Report 2017/149
7. Bentov, I., Pass, R., Shi, E.: Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919 (2016), `https://eprint.iacr.org/2016/919.pdf`
8. Bonneau, J.: Why buy when you can rent? Bribery attacks on Bitcoin consensus. In: BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin and Blockchain Research (Feb 2016), `http://fc16.ifca.ai/bitcoin/papers/Bon16b.pdf`
9. Botta, V., Friolo, D., Venturi, D., Visconti, I.: Shielded computations in smart contracts overcoming forks. In: Financial Cryptography and Data Security-25th International Conference, FC. pp. 1–5 (2021)
10. Brünjes, L., Gabbay, M.J.: Utxo-vs account-based smart contract blockchain programming paradigms. In: International Symposium on Leveraging Applications of Formal Methods. pp. 73–88. Springer (2020)
11. Buterin, V., Griffith, V.: Casper the Friendly Finality Gadget (2017), `https://arxiv.org/pdf/1710.09437.pdf`, published: arXiv:1710.09437
12. Chakravarty, M.M., Chapman, J., MacKenzie, K., Melkonian, O., Jones, M.P., Wadler, P.: The extended utxo model. In: International Conference on Financial Cryptography and Data Security. pp. 525–539. Springer (2020)
13. Corduan, J., Vinogradova, P., Gudemann, M.: A formal specification of the cardano ledger (2019)
14. Crandall, J.R., Wassermann, G., De Oliveira, D.A., Su, Z., Wu, S.F., Chong, F.T.: Temporal search: Detecting hidden malware timebombs with virtual machines. ACM SIGOPS Operating Systems Review **40**(5), 25–36 (2006)

15. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges (2019), https://arxiv.org/pdf/1904.05234.pdf, published: arXiv preprint arXiv:1904.05234
16. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 66–98. Springer (2018)
17. Delgado-Segura, S., Pérez-Sola, C., Navarro-Arribas, G., Herrera-Joancomartí, J.: Analysis of the bitcoin utxo set. In: International Conference on Financial Cryptography and Data Security. pp. 78–91. Springer (2018)
18. Di Angelo, M., Salzer, G.: Wallet contracts on ethereum. In: 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–2. IEEE (2020)
19. Di Angelo, M., Salzer, G.: Wallet contracts on ethereum–identification, types, usage, and profiles. arXiv preprint arXiv:2001.06909 (2020)
20. Dinsdale-Young, T., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: Afgjort: A partially synchronous finality layer for blockchains. In: International Conference on Security and Cryptography for Networks. pp. 24–44. Springer (2020)
21. Eskandari, S., Barrera, D., Stobert, E., Clark, J.: A first look at the usability of bitcoin key management. In: Workshop on Usable Security (USEC) (2015), http://users.encs.concordia.ca/clark/papers/2015_usec_full.pdf
22. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: front-running attacks on Blockchain. In: arXiv preprint arXiv:1902.05164 (2019), https://arxiv.org/pdf/1902.05164.pdf
23. Ethereum Community: Replay attack protection: Include blocklimit and blockhash in each transaction issue#134 ethereum/eips (Jul 2016), https://github.com/ethereum/EIPs/issues/134
24. Ethereum Community: Units and globally available variables (Aug 2021), https://github.com/ethereum/solidity/blob/develop/docs/units-and-global-variables.rst
25. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security. pp. 436–454. Springer (2014), http://arxiv.org/pdf/1311.0243
26. Ferreira Torres, C., Baden, M., Norvill, R., Jonker, H.: Ægis: Smart shielding of smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2589–2591 (2019)
27. Ferreira Torres, C., Iannillo, A.K., Gervais, A., et al.: The eye of horus: Spotting and analyzing attacks on ethereum smart contracts. In: International Conference on Financial Cryptography and Data Security, Grenada 1-5 March 2021 (2021)
28. Fratantonio, Y., Bianchi, A., Robertson, W., Kirda, E., Kruegel, C., Vigna, G.: Triggerscope: Towards detecting logic bombs in android applications. In: 2016 IEEE symposium on security and privacy (SP). pp. 377–396. IEEE (2016)
29. Garay, J., Kiayias, A.: SoK: A Consensus Taxonomy in the Blockchain Era (2018), https://eprint.iacr.org/2018/754.pdf, published: Cryptology ePrint Archive, Report 2018/754
30. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Advances in Cryptology-EUROCRYPT 2015. pp. 281–310. Springer (2015), http://courses.cs.washington.edu/courses/cse454/15wi/papers/bitcoin-765.pdf

31. Gaži, P., Kiayias, A., Russell, A.: Stake-Bleeding Attacks on Proof-of-Stake Blockchains (2018), https://eprint.iacr.org/2018/248.pdf, published: Cryptology ePrint Archive, Report 2018/248
32. Grundmann, M., Neudecker, T., Hartenstein, H.: Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin. In: 5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer (2018), http://fc18.ifca.ai/bitcoin/papers/bitcoin18-final10.pdf
33. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: International Conference on Financial Cryptography and Data Security. pp. 201–226. Springer (2020)
34. Guerraoui, R., Kuznetsov, P., Monti, M., Pavlovič, M., Seredinschi, D.A.: The consensus number of a cryptocurrency. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 307–316 (2019)
35. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 129–144 (2015), https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-heilman.pdf
36. Hu, T., Liu, X., Chen, T., Zhang, X., Huang, X., Niu, W., Lu, J., Zhou, K., Liu, Y.: Transaction-based classification and detection approach for ethereum smart contract. Information Processing & Management **58**(2), 102462 (2021)
37. Iqbal, M., Matulevičius, R.: Exploring sybil and double-spending risks in blockchain systems. IEEE Access **9**, 76153–76177 (2021)
38. Iqbal, M., Matulevičius, R.: Exploring sybil and double-spending risks in blockchain systems. vol. 9, pp. 76153–76177 (2021). https://doi.org/10.1109/ACCESS.2021.3081998
39. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gazi, P., Meiklejohn, S., Weippl, E.: Sok: Algorithmic incentive manipulation attacks on permissionless pow cryptocurrencies. IACR Cryptol. ePrint Arch. **2020**, 1614 (2020)
40. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gazi, P., Meiklejohn, S., Weippl, E.: Pay to win: Cheap, crowdfundable, cross-chain algorithmic incentive manipulation attacks on pow cryptocurrencies (2019), https://ia.cr/2019/775
41. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gaži, P., Meiklejohn, S., Weippl, E.: Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies (2019), https://eprint.iacr.org/2019/775.pdf, published: Cryptology ePrint Archive, Report 2019/775
42. Juels, A., Kosba, A., Shi, E.: The ring of Gyges: Investigating the future of criminal smart contracts. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 283–295. ACM (2016), http://www.arijuels.com/wp-content/uploads/2013/09/Gyges.pdf
43. Karakostas, D., Kiayias, A.: Securing Proof-of-Work Ledgers via Checkpointing. Tech. Rep. 173 (2020), https://eprint.iacr.org/2020/173
44. Karame, G.O., Androulaki, E., Capkun, S.: Double-spending fast payments in bitcoin. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 906–917 (2012)
45. Karame, G.O., Androulaki, E., Roeschlin, M., Gervais, A., Čapkun, S.: Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. In: ACM Transactions on Information and System Security (TISSEC). vol. 18, p. 2. ACM (2015), http://www.syssec.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/system-security-group-dam/research/publications/pub2015/tissec15_karame.pdf, issue: 1

46. Kelkar, M., Deb, S., Kannan, S.: Order-fair consensus in the permissionless setting (2021)
47. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Annual International Cryptology Conference. pp. 451–480. Springer (2020)
48. Kerber, T., Kiayias, A., Kohlweiss, M., Zikas, V.: Ouroboros crypsinous: Privacy-preserving proof-of-stake. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 157–174. IEEE (2019)
49. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)
50. Kolluri, A., Nikolic, I., Sergey, I., Hobor, A., Saxena, P.: Exploiting The Laws of Order in Smart Contracts (2018), https://arxiv.org/pdf/1810.11605.pdf, published: arXiv:1810.11605
51. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: Symposium on Security & Privacy. IEEE (2016), http://eprint.iacr.org/2015/675.pdf
52. Krombholz, K., Judmayer, A., Gusenbauer, M., Weippl, E.R.: The Other Side of the Coin: User Experiences with Bitcoin Security and Privacy. In: International Conference on Financial Cryptography and Data Security (FC) (2016), https://www.sba-research.org/wp-content/uploads/publications/TheOtherSideOfTheCoin_FC16preConf.pdf
53. Kursawe, K.: Wendy, the Good Little Fairness Widget. arXiv preprint arXiv:2007.08303 (2020)
54. Larimer, D.: Transactions as proof-of-stake (November 2013), https://github.com/super3/invictus.io/blob/master/assets/pdf/TransactionsAsProofOfStake10.pdf
55. Liao, K., Katz, J.: Incentivizing blockchain forks via whale transactions. In: International Conference on Financial Cryptography and Data Security. pp. 264–279. Springer (2017), http://www.cs.umd.edu/jkatz/papers/whale-txs.pdf
56. Lovejoy, J.P.T.: An empirical analysis of chain reorganizations and double-spend attacks on proof-of-work cryptocurrencies. Ph.D. thesis, Massachusetts Institute of Technology (2020)
57. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making Smart Contracts Smarter. In: 23rd ACM Conference on Computer and Communications Security (ACM CCS 2016) (Oct 2016), https://eprint.iacr.org/2016/633.pdf
58. Luu, L., Teutsch, J., Kulkarni, R., Saxena, P.: Demystifying incentives in the consensus computer. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 706–719. ACM (2015), http://www.comp.nus.edu.sg/prateeks/papers/VeriEther.pdf
59. Maersk, N.: Thedaohardforkoracle (Jul 2016), https://github.com/veox/solidity-contracts/blob/TheDAOHardForkOracle-v0.1/TheDAOHardForkOracle/TheDAOHardForkOracle.sol
60. Mai, A., Pfeffer, K., Gusenbauer, M., Weippl, E., Krombholz, K.: User mental models of cryptocurrency systems-a grounded theory approach. In: Sixteenth Symposium on Usable Privacy and Security ({SOUPS} 2020). pp. 341–358 (2020)
61. McCorry, P., Heilman, E., Miller, A.: Atomically Trading with Roger: Gambling on the success of a hardfork. In: CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology (Sep 2017), http://homepages.cs.ncl.ac.uk/patrick.mc-corry/atomically-trading-roger.pdf

62. McCorry, P., Hicks, A., Meiklejohn, S.: Smart Contracts for Bribing Miners. In: 5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer (2018), http://fc18.ifca.ai/bitcoin/papers/bitcoin18-final14.pdf
63. Meissner, R., Gnosis community: Gnosis safe contracts - executor. https://github.com/gnosis/safe-contracts/blob/34c87b783dfd04ff09ef7c358c3182c3c151e086/contracts/base/Executor.sol, accessed: 2021-09-07
64. Moroz, D.J., Aronoff, D.J., Narula, N., Parkes, D.C.: Double-spend counterattacks: Threat of retaliation in proof-of-work systems. arXiv preprint arXiv:2002.10736 (2020), https://arxiv.org/pdf/2002.10736.pdf
65. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (Dec 2008), https://bitcoin.org/bitcoin.pdf
66. Natoli, C., Gramoli, V.: The blockchain anomaly. In: Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on. pp. 310–317. IEEE (2016), https://arxiv.org/pdf/1605.05438.pdf
67. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: 1st IEEE European Symposium on Security and Privacy, 2016. IEEE (2016), http://eprint.iacr.org/2015/796.pdf
68. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 446–465. IEEE (2021)
69. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2017), https://eprint.iacr.org/2016/454.pdf
70. Pass, R., Shi, E.: Thunderella: Blockchains with Optimistic Instant Confirmation (2017), http://eprint.iacr.org/2017/913.pdf, published: Cryptology ePrint Archive, Report 2017/913
71. Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–33. Springer (2018)
72. Rodler, M., Li, W., Karame, G.O., Davi, L.: Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks (2018), https://arxiv.org/pdf/1812.05934.pdf, published: arXiv:1812.05934
73. Rosenfeld, M.: Analysis of Hashrate-Based Double Spending, vol. abs/1402.2009 (2014), https://arxiv.org/pdf/1402.2009.pdf, publication Title: CoRR
74. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. In: ACM Computing Surveys (CSUR). vol. 22, pp. 299–319. ACM (1990), http://www-users.cselabs.umn.edu/classes/Spring-2014/csci8980-sds/Papers/ProcessReplication/p299-schneider.pdf, issue: 4
75. Sergey, I., Hobor, A.: A Concurrent Perspective on Smart Contracts (2017), https://arxiv.org/pdf/1702.05511.pdf, publication Title: arXiv preprint arXiv:1702.05511
76. Sompolinsky, Y., Zohar, A.: Bitcoin's Security Model Revisited. arXiv preprint arXiv:1605.09193 (2016), http://arxiv.org/pdf/1605.09193.pdf
77. Sonnino, A., Bano, S., Al-Bassam, M., Danezis, G.: Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 294–308. IEEE (2020)

78. Todd, P.: Op_checklocktimeverify (Oct 2014), https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki

79. Tran, M., Choi, I., Moon, G.J., Vu, A.V., Kang, M.S.: A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In: To appear in Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P) (2020), https://erebus-attack.comp.nus.edu.sg/erebus-attack.pdf

80. Victor, F., Lüders, B.K.: Measuring ethereum-based erc20 token networks. In: International Conference on Financial Cryptography and Data Security. pp. 113–129. Springer (2019)

81. Wang, X., He, J., Xie, Z., Zhao, G., Cheung, S.C.: Contractguard: Defend ethereum smart contracts with embedded intrusion detection. IEEE Transactions on Services Computing **13**(2), 314–328 (2019)

82. Wohrer, M., Zdun, U.: Smart contracts: security patterns in the ethereum ecosystem and solidity. In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 2–8. IEEE (2018)

83. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)

84. Wu, L., Wu, S., Zhou, Y., Li, R., Wang, Z., Luo, X., Wang, C., Ren, K.: EthScope: A Transaction-centric Security Analytics Framework to Detect Malicious Smart Contracts on Ethereum. arXiv:2005.08278 [cs] (May 2020), http://arxiv.org/abs/2005.08278, arXiv: 2005.08278

85. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: SoK: Communication Across Distributed Ledgers. IACR Cryptology ePrint Archive, 2019: 1128 (2019), https://eprint.iacr.org/2019/1128.pdf

86. Zamyatin, A., Stifter, N., Judmayer, A., Schindler, P., Weippl, E., Knottebelt, W.J.: A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In: 5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer (2018), https://eprint.iacr.org/2018/087.pdf, (Short Paper)

87. Zhang, M., Zhang, X., Zhang, Y., Lin, Z.: {TXSPECTOR}: Uncovering attacks in ethereum from transactions. In: 29th {USENIX} Security Symposium ({USENIX} Security 20). pp. 2775–2792 (2020)

88. Zhang, R., Preneel, B.: Lay down the common metrics: Evaluating proof-of-work consensus protocols' security. In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE (2019), https://www.esat.kuleuven.be/cosic/publications/article-3005.pdf

89. Zhang, Y., Setty, S., Chen, Q., Zhou, L., Alvisi, L.: Byzantine ordered consensus without byzantine oligarchy. In: 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20). pp. 633–649 (2020)

90. Zhou, L., Qin, K., Gervais, A.: A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. arXiv preprint arXiv:2106.07371 (2021)

91. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-frequency trading on decentralized on-chain exchanges. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 428–445. IEEE (2021)

A Responsible Disclosure and Ethical Considerations

After careful consideration and review of the potential impact of this work, we believe that the presentation of opportunistic algorithmic double-spending and *OpAl* attacks does not require prior notification or responsible disclosure. As we outline within this paper, the presented double-spending techniques do not fundamentally increase the success probability of attacks aimed at causing blockchain forks, which are a necessary prerequisite for successfully performing most double-spending attacks. Hence, exchanges, merchants, and users that adhere to best practices and wait for sufficiently many confirmations on transactions do not appear to face substantially higher risk compared to other forms of double-spending.

However, we do see a potential for systemic risks in case of severe technical failures or attacks that cause deep forks, because of blockchain interlinking [85,77], layer-two protocols [33] and the existence of large centralized cryptocurrency exchanges, which all face the risk of becoming victims of algorithmic double-spending during such events. The principal mechanisms that can lead to *unintentional* algorithmic double-spending are already present in various smart contracts today. Consider the *slippage tolerance* users can specify as part of a trade when interacting with DEXs such as Uniswap [1]. In case of a deep fork, the necessary preconditions for a user's trade to be executed within a transaction may change, e.g., due to transaction reordering. The effect can be similar or even equivalent to intentional *OpAl* attacks.

We are hence convinced that raising awareness of these potential issues through openly publishing our findings is the best course of action.

B Definition of Semantic Malleability

Definition 5 (semantic malleability). *A transaction t is semantically malleable if at least one of the following conditions hold:*

1. *Executing t on two states σ_a, σ_b results in different changes to their respective substates, i.e.,* $\text{DIFF}(t, \sigma_a) \neq \text{DIFF}(t, \sigma_b)$.

$$\exists \sigma_a, \sigma_b \left(\text{VALID}(t, \sigma_a) \wedge \text{VALID}(t, \sigma_b) \wedge \text{DIFF}(t, \sigma_a) \neq \text{DIFF}(t, \sigma_b)\right) \quad (3)$$

2. *After a transaction t becomes valid for execution in state σ_a, t is invalidated at some future state σ_b without being executed.*

$$\exists \sigma_a, \sigma_b \left(\text{VALID}(t, \sigma_a) \wedge \sigma_a \prec \sigma_b \wedge t \notin \text{PAST}(\sigma_b) \wedge \neg\text{VALID}(t, \sigma_b)\right) \quad (4)$$

C Alternative *OpAl* Attack Designs in Ethereum

We hereby illustrate alternative constructions to the *OpAl* attack that is outlined in Section 3. Figure 2 shows two different attack scenarios that rely on depth-1 fork oracles. Hereby the first part of the illustration (left) captures an attack that

relies on querying the *coinbase address* of the current block and conditionally branches if a particular mining pool address is returned. The second part of the illustration (right) shows a scenario where the block height is used to trigger the attack.

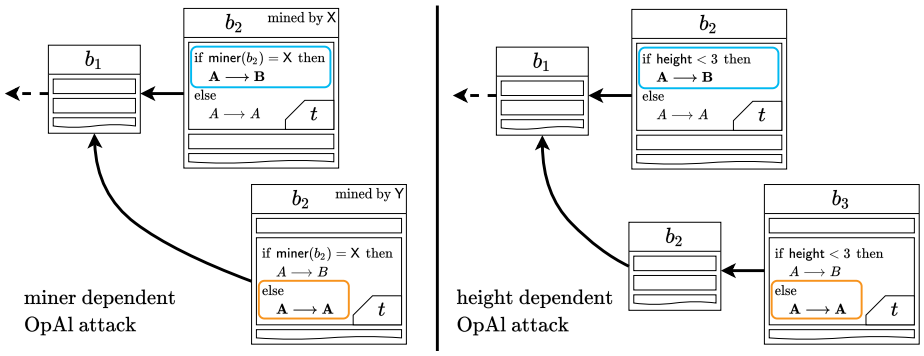

Fig. 2. Depth-1 *OpAl* attacks

The Figure 3 serves to illustrate how *OpAl* attacks can also be facilitated through *invalidation* of transactions, for example if the underlying protocol supports *context sensitive transactions*. Hereby, the transaction may only be valid for specific blockchain states, or during specific time intervals, at the protocol level, preventing the transaction, such as a payment to the merchant, to be replayed in a fork.

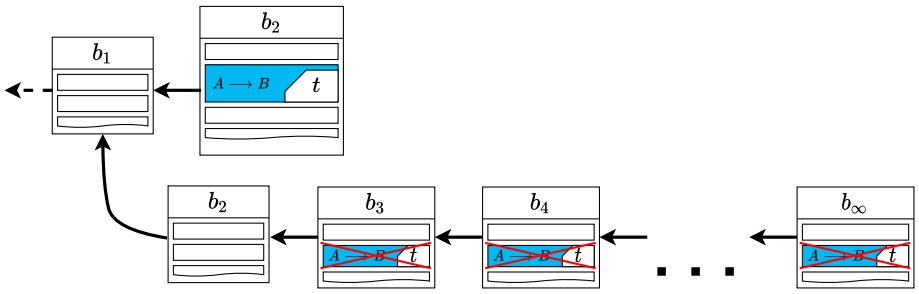

Fig. 3. *OpAl* attack based on transaction invalidation