Bribers, Bribers on The Chain, Is Resisting All in Vain?
Trustless Consensus Manipulation Through Bribing Contracts

Bence Soóki-Tóth[1,2**], István András Seres**[1], Kamilla Kara[1], Ábel Nagy[1], Balázs Pejó[3,4], Gergely Biczók[3,4]

[1] Eötvös Loránd University
[2] Aarhus University
[3] CrySyS Lab, Dep. of Networked Systems and Services, Budapest University of Technology and Economics
[4] HUN-REN-BME Information Systems Research Group

Abstract. The long-term success of cryptocurrencies largely depends on the incentive compatibility provided to the validators. Bribery attacks, facilitated trustlessly via smart contracts, threaten this foundation. This work introduces, implements, and evaluates three novel and efficient bribery contracts targeting Ethereum validators. The first bribery contract enables a briber to fork the blockchain by buying votes on their proposed blocks. The second contract incentivizes validators to voluntarily exit the consensus protocol, thus increasing the adversary's relative staking power. The third contract builds a trustless bribery market that enables the briber to auction off their manipulative power over the RANDAO, Ethereum's distributed randomness beacon. Finally, we provide an initial game-theoretical analysis of one of the described bribery markets.

Keywords: Proof-of-stake · Consensus Manipulation · Bribery Attacks · Smart Contracts · Ethereum · Algorithmic Incentive Manipulation

1 Introduction

The incentive compatibility of protocol participants is a central theme in cryptocurrency security research, as first analyzed by Nakamoto in the celebrated Bitcoin white paper [33, Section 6.]. The seminal work of Eyal and Sirer demonstrated the first incentive incompatibilities in the Bitcoin protocol, showing that an economically rational miner would not follow the protocol but rather employ a selfish mining strategy to increase its block rewards [18]. Subsequently, an extensive research direction has explored the incentives of protocol participants in various settings, such as Bitcoin without block rewards [13]. This paper studies the incentives of a cryptocurrency consensus protocol where trustless, cheap, and efficient Turing-complete bribery contracts are available. In a

** Bence Soóki-Tóth and István András Seres contributed equally to this work.

bribery attack, a motivated briber offers a payment to *economically rational validators* to deviate from the protocol, to *e.g.,* double spend [25,43], fork the blockchain [4,23], or censor transactions [5,29,31]. Such protocol deviations can often be verified efficiently using cryptographic primitives (*e.g.,* signatures or zero-knowledge proofs), facilitating the atomic exchange of the bribe for the requested deviant behavior.

Bribery attacks have a decade-long history, initiated by Bonneau at FC'16 [8]. Much of this prior work has focused on bribing validators cross-chain, which considerably limits their practicality [23,24,25]. These works often assume a funding blockchain that hosts the bribery contracts and another (typically Proof-of-Work (PoW)) blockchain whose validators are being bribed, introducing limitations related to synchrony, liquidity, and interoperability. To our knowledge, bribery attacks leveraging Turing-complete scripting capabilities native to a Proof-of-Stake (PoS) consensus algorithm have not been rigorously analyzed. This setting is particularly risky because verifying properties of a PoS consensus algorithm (*e.g.,* the current head of the chain [28] or the active validator set) is often more efficient than in a PoW context, especially with access to expressive smart contract functionality. Note that in our closest related work, the commitment attacks devised in [37] are not trustless, as the bribee must trust the briber.

This research gap is critical due to a common misconception about cryptoeconomic security. It is widely but wrongly believed that permissionless consensus protocols offer a security budget equal to the total value of staked assets ($\approx$ 115.3 billion USD in Ethereum). This is only true for long-term takeovers. Contrary to this belief, in more realistic models with economically rational validators, the true security budget is significantly lower. We emphasize this by demonstrating the surprisingly low cost of the attacks we introduce. In particular, we show that an adversary can fork the Ethereum blockchain by offering bribes of less than 0.09 Ether ($\approx$ 334 USD) to rational validators, *cf.* Section 5.1.

The attacks we present undermine the fundamental security guarantees of a cryptocurrency consensus protocol. Although cryptocurrencies pursue many security goals, their integrity rests on three core properties: safety, liveness, and fairness. Safety ensures that every honest validator agrees on the same ledger state. Liveness guarantees that all valid transactions are eventually included in the ledger. Finally, fairness dictates that a validator with α percent of the voting power should, in expectation, receive α percent of the protocol rewards.

Our Contributions. This work makes the following contributions.

Novel Bribery Contracts for PoS Ethereum. We design, implement, and evaluate three efficient, smart contract-based bribery attacks, each targeting a fundamental consensus property:

- PayToAttest**:** Violates **safety** by enabling a briber to fork the blockchain by buying votes (often validator attestations) for a competing chain.
- PayToExit**:** Threatens **liveness** by incentivizing validators to exit the active validator set, thereby increasing the adversary's relative stake.

– PayToBias**:** Impacts **fairness** by creating a market for biasing the RANDAO, Ethereum's randomness beacon. To our knowledge, this is the first public implementation of such a bribery market.

We have released our implementation as an open-source artifact to foster further research. [5]

Game-Theoretic Analysis of Practicality. We model the PayToExit attack as a Stackelberg game to characterize the adversary's optimal bribe. Our analysis demonstrates the attack's practicality, deriving an optimal per-validator bribe of 9.23 ETH ($\approx 34,225$ USD), an accessible amount for a motivated attacker.

Outline. The remainder of this paper is organized as follows. We review the pertinent background in Section 2. We describe our system and threat models in Section 3. We introduce our three bribery contracts in Section 4 and analyze the resulting incentives in Section 5. We discuss our related work in Section 6. We conclude our paper with potential extensions and open questions in Section 7.

2 Preliminaries

Notations. We model the consensus environment at a given time t as having $N(t)$ total active validators who collectively stake $S(t)$ amount of ETH. An adversary, or briber, controls α fraction of this total stake ($0 \leq \alpha \leq 0.5$). The remaining $(1-\alpha)$ stake belongs to non-adversarial validators, which we partition based on their behavior: a fraction β ($0 \leq \beta \leq 1$) are *economically rational* and will accept a profitable bribe, while the remaining fraction $(1-\alpha)(1-\beta)$ are *honest* and will always follow the protocol. The virtual weight for timely block proposals is denoted by p_{boost}, which is set to $p_{\mathsf{boost}} = 0.4$, at the time of writing.

2.1 The BLS signature scheme

The BLS signature scheme, proposed by Boneh, Lynn, and Shacham [7], is used in the Ethereum consensus protocol for validator attestations. The scheme utilizes a non-degenerate, efficiently computable, bilinear pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ over prime-order cyclic groups such that $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p$. It consists of the following algorithms:

$\mathsf{BLS.KeyGen}(1^\lambda)$**.** Samples uniformly at random a secret signing key $\mathsf{sk} \xleftarrow{\$} \mathbb{F}_p$ and sets the public verification key as $\mathsf{pk} = g^{\mathsf{sk}} \in \mathbb{G}_1$.

$\mathsf{BLS.Sign}(\mathsf{sk}, m)$**.** Computes the signature $\sigma := H(m)^{\mathsf{sk}} \in \mathbb{G}_2$, where $H : \{0,1\}^* \to \mathbb{G}_2$ is a hash-to-curve function, and modeled as a random oracle.

$\mathsf{BLS.Verify}(\mathsf{pk}, m, \sigma)$**.** Outputs 1 if $e(\sigma, g_2) = e(\mathsf{pk}, H(m))$ holds, and 0 otherwise.

[5] https://github.com/0xSooki/bribery-zoo.

```
class AttestationData(
    Container):
slot: Slot
index: CommitteeIndex
# LMD GHOST vote
beacon_block_root: Root
# FFG vote
source: Checkpoint
target: Checkpoint
```

`AttestationData` class

```
class VoluntaryExit(
    Container):
# Epoch when exit can
    be processed
epoch: Epoch
validator_index:
    ValidatorIndex
```

`VoluntaryExit` class

Fig. 1: Structures of the `AttestationData` and `VoluntaryExit` classes whose instances must be signed by validators whenever they attest to a proposed block or they want to cease to be a validator. Note that the `index` field was removed from the `AttestationData` for efficiency reasons as part of the Pectra hard fork.

The BLS signature scheme is unique and provides strong unforgeability under the co-CDH assumption [7]. The crucial property we exploit in this work is the scheme's support for signature aggregation [6]. More precisely, if the *same message* m is signed under multiple public keys $\{\mathsf{pk}_i\}_{i=1}^n$, the corresponding individual signatures $\{\sigma_i\}_{i=1}^n$ can be combined. The resulting aggregate signature $\sigma = \sigma_1 \cdot \ldots \sigma_n$ and aggregate public key $\mathsf{pk} = \mathsf{pk}_1 \cdot \ldots \mathsf{pk}_n$ can be validated with a single pairing check, reducing the verification complexity to $\mathcal{O}(1)$:

$$e\Big(\prod_{i=1}^{n} \sigma_i, g_2\Big) \stackrel{?}{=} e\Big(\prod_{i=1}^{n} \mathsf{pk}_i, H(m)\Big) \ . \tag{1}$$

The $\mathcal{O}(1)$ verification efficiency does not hold if validators sign *different messages* $\{m_i\}_{i=1}^n$. In this scenario, the verification equation requires $n+1$ pairings and n target group operations. Specifically:

$$e\Big(\prod_{i=1}^{n} \sigma_i, g_2\Big) \stackrel{?}{=} \prod_{i=1}^{n} e\Big(\mathsf{pk}_i, H(m_i)\Big) \ . \tag{2}$$

Figure 8 compares the on-chain gas cost of BLS batch verification for n signatures on a single message m versus on distinct messages $\{m_i\}_{i\in[n]}$.

2.2 Ethereum Proof-of-Stake essentials

The Ethereum Proof-of-Stake protocol [35] relies on validators to secure the network by proposing and attesting to blocks. Any user can become a validator by depositing 32 ETH into the staking contract, and can similarly exit by withdrawing their stake. Validators are compensated with protocol rewards for

correctly performing their duties, but their stake can be slashed for malicious behavior such as proposing or attesting to conflicting blocks. Crucially, the bribery attacks we introduce in this work *do not entail* any slashable offenses.

In Ethereum PoS, time is structured into epochs, with each epoch comprising 32 slots of 12 seconds each. Within the first four seconds of a slot, an assigned validator may propose a new block. During the remaining eight seconds, a committee of $\frac{N(t)}{32}$ validators is selected to attest to what they perceive as the head of the chain by BLS-signing an attestation message, *cf.* Figure 1.

If a fork occurs, the fork-choice rule dictates that the branch in the block tree with the greatest weight of attestations is considered the canonical chain. Attestation processing and rewards are described in further detail in Appendix A.3. Importantly for our work, validator duties (*i.e.,* block proposal and attestation roles) are publicly known at least 6.4 minutes in advance. For a comprehensive description of the Ethereum PoS protocol, we refer the reader to [35].

Recent Protocol Upgrades Several recent Ethereum Improvement Proposals (EIPs) are essential for implementing the efficient bribery contracts we present. EIP-4788 [41], included in the Dencun upgrade (March 2024), exposes the beacon block root to the Ethereum Virtual Machine (EVM), greatly simplifying the on-chain verification of consensus-related statements for smart contracts.

Additionally, three EIPs from the Pectra upgrade (May 2025) provide crucial functionalities. EIP-2537 [44] introduces gas-efficient precompiles for pairing operations on the BLS12-381 curve. EIP-7002 [36] allows validator exits to be initiated from the execution layer, which enables a smart contract to verify more efficiently that a validator has voluntarily exited the active set. Finally, EIP-7549 [17] moves the index field out of the AttestationData structure, *cf.* Figure 1. This change ensures that when validators attest to the same block, they all BLS-sign the exact *same message*. As a result, thousands of their signatures can be aggregated off-chain and verified on-chain with high efficiency, *cf.* Equation (1).

3 System Model

Our system model, depicted in Figure 2, consists of four main entities. The consensus participants within this model are categorized according to the BAR (Byzantine, Altruistic, Rational) fault model [1].

Briber The briber is a potentially extrinsically motivated entity controlling an α fraction of the total stake. Their objective is to manipulate consensus by offering rewards to other validators for deviating from the protocol.

Rational Rational validators, controlling a $(1-\alpha)\beta$ fraction of the stake, are profit-maximizing actors. We assume they will deviate from the protocol if an offered bribe is greater than the potential rewards from honest participation.

Altruistic Altruistic validators, who control a $(1-\alpha)(1-\beta)$ fraction of the stake, are honest participants. They adhere strictly to the protocol at all times, regardless of any external incentives or bribes.

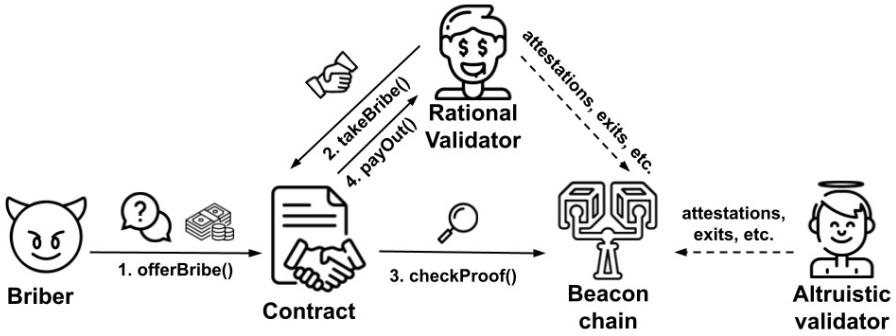


Fig. 2: Our simple bribery system model. Typically, Steps 2 and 3 are executed atomically in the same transaction. If not, then the transaction in Step 3 could be initiated by anyone; thus, it originates from the contract in the figure.

Contract To facilitate an atomic, fair exchange, which is impossible without a trusted third party [34], we model a smart contract that acts as this trusted intermediary. This on-chain entity has a transparent state, holds the briber's funds in escrow, and automatically releases payment to any validator who provably fulfills the briber's conditions (*e.g.,* provides a specific (BLS) signature, Merkle authentication path, or other zero-knowledge proofs).

For our analysis, we assume the staking power of each entity remains constant, with the notable exception of the PayToExit model, which analyzes dynamic changes in stake distribution. We also assume a synchronous communication model where the maximum network delay is bounded at four seconds, corresponding to one-third of the duration of an Ethereum slot.

3.1 Bribing contract interface

We foresee a future where consensus manipulation could become normalized and institutionalized, much like Maximal Extractable Value (MEV) is today [15]. Such a development would necessitate standard, open interfaces to allow bribers and bribees to interact seamlessly. Thus, we propose a baseline interface, IBribe.

We expect that rational agents will constantly monitor contracts implementing this interface by calling the view-only function bribeAmnt(), to automatically accept any bribe exceeding their profitability threshold. This behavior mirrors the automated strategies already seen in on-chain MEV extraction [39].

The IBribe interface, *cf.* Figure 3, defines two categories of functions. The core bribery interaction is handled by offerBribe() and takeBribe(), which facilitate the bribe itself (*cf.* Figure 2). Additionally, it includes three convenience functions for fund management: depositFunds(), updateBribeAmnt(), and withdrawFunds().

Offer bribe The briber initiates the process by calling the offerBribe() function. In this transaction, they deposit funds into the bribery contract and specify

```
interface IBribe {
  function bribeAmnt() public view returns (uint256);
  function depositFunds() external payable;
  function updateBribeAmnt(uint256 amnt) external;
  function withdrawFunds(uint256 amnt) external;
  function offerBribe(bytes calldata pubkey, uint256 amnt,
    bytes memory data) external;
  function takeBribe(bytes calldata signature) external;
}
```

Fig. 3: The IBribe interface for our proposed bribery contracts.

the required protocol deviation – typically a structured message that the bribee must sign (*cf.* Figure 1).

Take bribe To accept the offer, the bribee calls the takeBribe() function. This call must include a proof of the required deviation, which is usually a cryptographic signature (*e.g.,* BLS or ECDSA) over a specific protocol message, such as an attestation or a voluntary exit transaction.

Check proof and pay out Upon receiving the takeBribe() call, the contract first verifies the submitted proof (via the internal checkProof() function). If the proof is valid, the contract atomically pays the bribe to the bribee (via payOut() step). While these actions are often executed within the single takeBribe() transaction, they can be separated. For instance, in recurring bribes designed to maintain a specific behavior (*cf.* Section 4.2), the proof submission and payout may be handled in distinct transactions.

4 Bribing Contracts: *"Bribers, Bribers On The Chain!"*

4.1 The PayToAttest bribery contract

The PayToAttest contract enables a briber to purchase BLS signatures from a set of validators, $\{\mathsf{pk}_i\}_{i=1}^n$, for a specific block header m. To initiate the bribe, the briber commits the aggregate public key, $\mathsf{pk} = \prod_{i=1}^{n} \mathsf{pk}_i$, and the block hash, $H(m)$, to the smart contract. Participating validators can then claim their reward by providing a valid aggregate BLS signature σ that satisfies the verification equation in Equation (1). To ensure atomicity between the execution and beacon chains, EIP-4788 (*cf.* Section 2.2) can be used to verify that this aggregate signature σ was included in a beacon block.

This mechanism is a powerful tool for facilitating forking attacks. For example, an attacker can orchestrate an ex-ante reorg [38] to manipulate the RANDAO beacon [32] or to steal MEV [37], as we detail in Appendix D.2. While ex-ante reorgs are a known risk, it is widely believed that non-majority attackers cannot reliably launch ex-post reorgs [38]. However, we now describe how

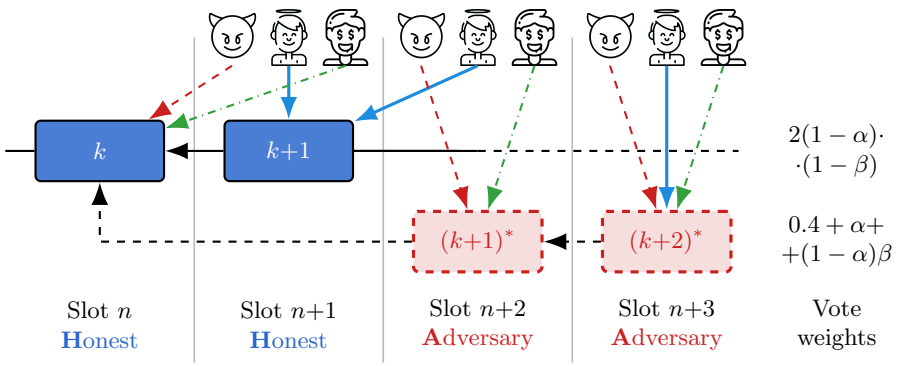


Fig. 4: Leveraging the PayToAttest bribery contract to ex-post reorg the block proposed in Slot $(n+1)$. Colored arrows indicate which blocks different validators vote for as the head of the blockchain. Black arrows represent hash pointers. Red (blue) blocks are proposed by the briber (honest validators).

the PayToAttest contract provides a novel mechanism for a *non-majority* briber to successfully perform such an attack.

The ex-post reorg proceeds as follows. For concreteness, we assume a scenario where an honest block is published in Slot $n+1$, followed by two slots where the briber is the proposer, as depicted in Figure 4. The adversary's goal is to fork out Block $k+1$, perhaps for its high MEV content. After Block $k+1$ is published, the adversary bribes rational attestators to instead vote for the preceding Block k. In the next slot, the adversary proposes their own block, Block $(k+1)^*$, which builds on Block k, and again bribes attestators to vote for it, while honest validators continue to vote for Block $k+1$. When the adversary publishes their second block, Block $(k+2)^*$, their fork has accumulated a weight of $p_{\mathsf{boost}}+\alpha+(1-\alpha)\beta$ attestations, compared to the honest fork's $2(1-\alpha)(1-\beta)$ votes. If the following condition holds,

$$2(1-\alpha)(1-\beta) \leq p_{\mathsf{boost}} + \alpha + (1-\alpha)\beta \ , \tag{3}$$

then the adversary's fork becomes canonical, and all validators then vote for Block $(k+2)^*$ in Slot $n+3$, completing the attack. A complete sequence diagram is shown in Figure 11, and the associated bribery costs are studied in Section 5.1.

4.2 The PayToExit bribery contract

The PayToExit contract creates a market where a briber can incentivize validators to voluntarily exit the Ethereum consensus protocol. A briber's motivation could be multifold: they can harm the network's liveness by reducing the number of active participants, increase their own relative staking power, or attract scarce capital from the deposit contract to an on-chain application (*e.g.,* a decentralized exchange or lending platform).

To claim the reward, a validator must prove to the contract that they have successfully exited the validator set after the bribe was offered. This proof consists of two components: a Merkle proof confirming their prior status as an active validator, and a valid BLS signature on a `VoluntaryExit` message (*cf.* Figure 1). Note that such a trustless PayToExit contract is not feasible in Proof-of-Work, as there is no on-chain mechanism to verify that a miner has permanently shut down their hardware. The pseudocode for this contract is detailed in Algorithm 2.

A key challenge in this model is the restaking loophole. A rational validator could exit, claim the bribe, transfer their funds to a fresh address, and rejoin the validator set. A more robust design could mitigate this by transforming the bribe into a recurring payment, rewarding the bribee as long as their withdrawn funds remain at the withdraw address, thus disincentivizing restaking. We leave the implementation of this extension of our PayToExit contract to future work.

4.3 The PayToBias bribery contract

Ethereum's RANDAO is a distributed randomness beacon used to select block proposers. Its output for an epoch e is calculated as $R^e := R^{e-1} \oplus (\oplus_{i=1}^{32} r_i^e)$, where r_i^e is the proposer's randomness contribution for each slot. Since each contribution r_i^e is a BLS signature ($\mathsf{BLS.Sign}(\mathsf{sk}_i, e)$), it is unique and cannot be grind. The only influence a validator has is the binary choice to publish their block (revealing their contribution) or withhold it. This design is known to be biasable, as it gives proposers in the final "tail" slots of an epoch significant power to influence the outcome, which in turn determines the proposers for epoch $e + 2$ [3,32,42,47].

A rational validator controlling k tail slots can therefore pre-compute 2^k possible RANDAO outcomes and select the one most beneficial to themselves. Alternatively, they can auction this right to influence the outcome to other parties. This concept is known as the RANDAO bribery market, and our PayToBias contract is, to our knowledge, its first practical implementation.

The auction mechanism is straightforward. The manipulator (the validator controlling k tail slots) first calls offerBribe() to reveal their k randomness contributions, *i.e.,* $\forall i \in [k] : r_{32-i}^e$. Once the epoch's other contributions are public, other validators bid on their preferred outcome by submitting a publish/withhold configuration ($\mathbf{c}_i \in \{0,1\}^k$). The manipulator is then incentivized to execute the configuration $\mathbf{c}_i$ that attracted the highest total value of bids.

Finally, to claim their payout, the manipulator must prove which configuration they executed by submitting the relevant block headers from the epoch's conclusion. The contract verifies the authenticity of these headers against the on-chain "blockhash" history, which is available for up to $8,191$ blocks ($\approx$27.3 hours) per EIP-2935 [12]. By inspecting the timestamps of the verified headers, the contract confirms which slots were missed, determines the executed configuration $\mathbf{c}_i$, and releases the funds. The complete pseudocode of the RANDAO bribery market is available in Appendix B.3. In Section 5.3, we compute a worst-case expected upper bound for the bribe amount to manipulate the RANDAO, whenever the manipulator has exactly k tail slots.

Contract \ Function	Constructor(·)	offerBribe(·)	takeBribe(·)
PayToAttest	805,876 (4.87 USD)	361,117 (2.18 USD)	252,110 (1.52 USD)
PayToExit	862,575 (5.21 USD)	225,123 (1.36 USD)	260,643 (1.58 USD)
PayToBias	1,296,940 (7.84 USD)	94,944 (0.57 USD)	137,897 (0.83 USD)

Table 1: The most important functions' gas costs in our bribing contracts. USD costs were computed using ETH/USD exchange price and gas prices (3,708 USD/ETH and 1.63 Gwei gas price) on July 25th, 2025.

4.4 Performance Evaluation and Qualitative Comparison

We implemented our proposed bribery contracts in Solidity and evaluated the gas costs incurred by both the briber and the bribees. Our evaluation confirms that the contracts are highly efficient, imposing negligible on-chain financial costs for participants in these bribery markets (*cf.* Table 1). A qualitative comparison of our contracts with related work on bribery attacks is provided in Table 3. The complete implementation is publicly available in our open-source repository.[6]

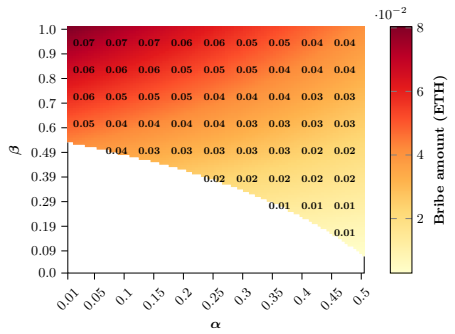
4.5 Privacy-preserving Bribery Contracts

The public nature of the blockchain may deter validators from participating in bribery markets, as their identifying details (e.g., public keys, validator indices) and transaction amounts are transparent. A natural extension of our work is therefore to incorporate privacy protections into these contracts. The core computations in our contracts, such as verifying Merkle proofs and BLS signatures, are well-suited for zero-knowledge techniques. These operations could be embedded within a zkSNARK, such as the widely-used Groth16 proof system [20], which is particularly efficient for on-chain verification. While we leave the implementation of a fully private bribery contract to future work, this represents a promising direction for making such bribery markets more practical.

4.6 Responsible disclosure and ethical considerations

We responsibly disclosed our findings to the Ethereum Foundation before publishing our work. We ran and tested our bribery contract implementations solely on a local test blockchain without interfering with the live Ethereum consensus.

[6] https://github.com/0xSooki/bribery-zoo



(a) Ex-post reorg costs (ETH) using the PayToAttest contract for the chainstring $\mathbf{H}^e_n\mathbf{H}^e_{n+1}\mathbf{A}^e_{n+2}\mathbf{A}^e_{n+3}$, *cf.* Section 4.1. The attack is unsuccessful in the white region.

(b) PayToExit bribery contract costs (ETH) in the number of bribed validators for different (Y, r) parameters (year, discount rate).

Fig. 5: PayToExit and PayToAttest (ex-post reorg) bribery contract costs.

5 Bribee Incentives: *"Is Resisting All in Vain?"*

5.1 PayToAttest incentives

As established in Section 4.1, a non-majority adversary can perform an ex-post reorg using a PayToAttest bribery contract to fork out an honest block $\mathbf{H}^e_{n+1}$ at Slot $k+1$ for a chain string $\mathbf{H}^e_n\mathbf{H}^e_{n+1}\mathbf{A}^e_{n+2}\mathbf{A}^e_{n+3}$ within epoch e. According to the winning condition in Equation (3), the briber must purchase attestations from a $(1-\alpha)\beta$ fraction of the $\frac{N(t)}{32}$ validators in the attestation committee. The protocol rewards each correct and timely attestation with an amount proportional to the total stake, approximately $c \cdot \sqrt{S(t)}$ ETH. Therefore, to succeed, the briber only needs to offer a total bribe slightly greater than the total rewards these validators would otherwise receive (*i.e.,* $(1-\alpha)\beta\frac{N(t)}{32}(c\sqrt{S(t)})$).

Our analysis of the concrete costs, using network data from April 1, 2025, is presented in Figure 5a. The results are alarming: for all successful attack parameters (α, β), the total bribe required is less than 0.09 ETH ($\approx$ 334 USD). This cost is significantly lower than the typical MEV found in a single Ethereum block [46], implying that such an attack could be highly profitable.

5.2 PayToExit incentives

We analyze the PayToExit bribery market through a game-theoretic lens, modeling it as a two-stage Stackelberg market exit game [40] with a single leader (the briber) and multiple followers (the validators). First, the leader commits to a bribe b; second, each follower decides whether to accept the bribe and exit, or to refuse and remain. The objective of this analysis is to demonstrate the practicality and economic rationality behind this attack. To this end, we analyze a simplified game that captures these essential dynamics, as illustrated in Figure 6.

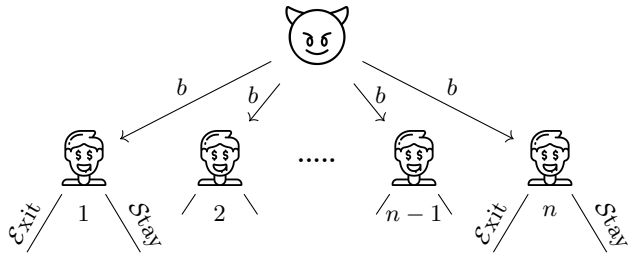


Fig. 6: PayToExit: a single-leader multiple-follower Stackelberg market exit game. A briber offers a bribe b to n rational validators to increase its market share α.

Assumptions. Our analysis is based on several simplifying assumptions. We assume the briber's (leader's) objective is to achieve a target market share of $\alpha^* > \alpha$, which requires a total of k^* validators to exit. We assume the following.

Homogeneous Validators The follower set is homogeneous, *i.e.,* all rational validators have identical utility functions and a uniform stake (= 32 ETH).

Sufficient Rational Validators The network has N total validators, where the number of rational validators is $n = (1 - \alpha)\beta N$. This value is strictly larger than the required number of exits k^*.

Constant Bribe A single, constant bribe b is offered. This represents an upper bound for the required bribe, as it is the amount needed to convince the marginal (last) validator to exit, *cf.* Appendix E.2.

Perfect Information The game has perfect information, as the PayToExit smart contract makes all relevant data – such as the number of validators who have already committed to exiting – publicly visible to all participants.

Simplified Exits We do not model the sequential nature of the exit process, *i.e.,* queueing delays or specific timestamps; our follower game is single-shot.

Players and Strategies. The game has two types of players, *cf.* Figure 6. The leader is a single briber with α stake, whose objective is to increase their market share to α^*, an outcome requiring k^* validators to exit. The followers are the $n > k^*$ rational validators who aim to maximize their individual profit. The leader's strategy set, $S_0 = \{b | b \in \mathbb{R}^+\}$, is the choice of a bribe amount b. Each follower's strategy set is the binary choice $S_i \in \{\mathcal{E}, \mathcal{S}\}$, where $\mathcal{E}$ represents accepting the bribe and exiting the protocol, and $\mathcal{S}$ represents refusing the bribe and staying.

Leader's Utility. The leader's (briber's) utility, Π_0, is defined in Equation (4). It consists of the opportunity gain from the increased market share, $g(k)$, plus the extra profit, Π_0^e, collected from sources exogenous to the consensus protocol, minus the total cost of the bribes, *i.e.,* $k \cdot b$ where k is the number of validators who exit. Exogenous motivations of a PayToExit briber may be to disrupt Ethereum by creating liveness problems. Another, exogenous profit source of a PayToExit

briber could be to attract scarce capital from the deposit contract to their own decentralized applications, akin to vampire attacks [21]. The gain function $g(k)$ represents the net present value of the briber's increased future staking rewards. It is calculated by multiplying the annual reward gain, $\mathrm{R}(N-k)-\mathrm{R}(N)$, by a present value multiplier, $\mathrm{PV}(r,Y)$. More specifically, $\mathrm{R}(n)$ is the total annual staking reward for a single validator when n validators are active: it consists of the attestation and block proposer rewards (sync committee rewards are not modeled) [35]. We estimated the yearly average MEV amount per validator using on-chain data, *i.e.,* we computed the average payment value that block builders paid to Ethereum block proposers from September 15, 2022, to September 12, 2025. The term $\mathrm{PV}(r,Y)$ is the present value multiplier, with a discount factor r over a time horizon of Y years as in [10]. We set this discount rate to $r=0.08$: this value is higher than the return on risk-free assets like treasury bonds ($4-5\%$) to account for Ethereum's volatility, yet lower than returns on more speculative assets ($\geq 10\%$), aligning with the crypto staking literature [14]. For the time horizon Y, we use an effective horizon based on the Present Value half-life [10], which is approximately 9 years in this case (*cf.* Appendix E.1).

$$\Pi_0 = \underbrace{g(k)}_{\text{opp. gain}} + \underbrace{\Pi_0^e}_{\text{extra profit}} - \underbrace{k\cdot b}_{\text{cost}} = \alpha\cdot N(R(N-k)-R(N))\cdot \mathrm{PV}(r,Y)+\Pi_0^e - k\cdot b \tag{4}$$

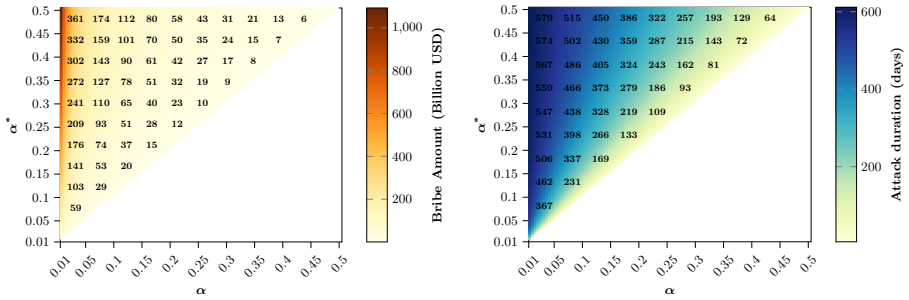
where

$$R(n) = \underbrace{32}_{stake}\cdot\left(\underbrace{\frac{2940.21}{\sqrt{n}}}_{\text{protocol rewards}} + \underbrace{\frac{1078543.3}{n}}_{\text{estimated MEV}}\right) \quad \text{and} \quad \mathrm{PV}(r,Y) = \frac{1.08^{-9}}{0.08\cdot 100} \tag{5}$$

Follower's Utility. A follower's (validator's) utility, Π_i, is defined in Equation (6), depending on their choice to either exit or stay. We define the utility for each action in line with our assumption on a worst-case flat bribe b: assuming $k-1$ other validators have already exited. The utility when the follower exits is the value of the bribe b: the exiting validator forgoes its future staking rewards, realizing an opportunity cost. Conversely, the utility when staying is the present value of the increased future staking rewards that result from $k-1$ other validators leaving the active set.

$$\Pi_i = \begin{cases} b & \text{if } \ S_i = \mathcal{E} \\ R(n-k+1)\cdot \mathrm{PV}(r,Y) & \text{if } \ S_i = \mathcal{S} \end{cases} \tag{6}$$

Equilibrium analysis. We solve the game using backward induction. We first find the equilibrium in the follower subgame and then substitute this result into the leader's problem to determine the optimal bribe, b^*. An equilibrium in the follower subgame is a state where no validator has an incentive to unilaterally



(a) PayToExit optimal bribe costs (b) PayToExit attack duration

Fig. 7: PayToExit bribe costs (in USD) and attack duration (in days) whenever the briber wants to increase its relative staking power from α to α^*, ($\alpha \leq \alpha^*$).

change their strategy, *i.e.,* to switch from exit to stay and vice versa. For an equilibrium state with k exiting validators, this means $b - R(n-k+1) \cdot \mathrm{PV}(r, Y) \geq 0$ and $R(n-k) \cdot \mathrm{PV}(r, Y) - b \leq 0$. These conditions establish the range for a bribe b that results in exactly k exits. Hence, to incentivize the precise number of desired exits (*i.e.,* k^*), the leader must offer an optimal bribe b^* such that

$$R(n-k+1) \cdot \mathrm{PV}(r, Y) \leq b^* \leq R(n-k) \cdot \mathrm{PV}(r, Y). \tag{7}$$

Note that a rational briber requires $\Pi_0 > 0$ to participate in the game. A quick calculation shows that this requires extra exogenous profit $\Pi_0^e > 0$ for all optimal bribes b^*. In any realistic (and now standard) cryptocurrency adversarial model, one endows the adversary with exogenous motivations [19].

Numerical Example. At the time of writing, there are approximately $N = 1\,123\,611$ active validators, the largest staking entity (Lido) controls about 23.9% of the stake. If this entity aimed to amass a 33% share – a threshold that could threaten the liveness of the Ethereum protocol – it would need to compel a significant number of other validators to exit. Our calculations show that to reach this target, a total of $k^* \approx 317\,982$ validators would need to leave the active validator set. Using the equilibrium bribe b^* from Equation (7), this scenario would require a bribe of 9.23 ETH per validator. As of September 20, 2025, this translates to approximately 41 332.91 USD for each exiting validator. To make this scenario plausible for a rational briber, the extra exogenous profit should be more than 250 541.1 $\frac{ETH}{\text{year}}$, thus, limiting the attack's realistic threat profile.

5.3 PayToBias incentives

We give a worst-case upper bound on the expected bribe amount whenever a bribee with $(1-\alpha)\beta$ stake auctions off the manipulative power of k consecutive tail slots in epoch e. The briber must compensate the bribee with the costs incurred in epoch e and $e+2$, *i.e.,* let us denote this by c_e and c_{e+2}. First, in

epoch e, in the worst case, the briber may request the bribee to forfeit proposing blocks in all its k consecutive slots. Thus, the briber must pay at least the block rewards, transaction fees, missed attestation rewards, and maximal extractable value that the bribee gives up in those k tail slots. We estimate $c_e \approx k \cdot \mathsf{v_b}$, where $\mathsf{v_b}$ is the value of a single Ethereum block and the attestation rewards of the bribee's validators. Second, the bribee could have proposed $X \sim Binom(32, \alpha)$ blocks in epoch $e+2$, but with the different seed $Y \sim Binom(32, \alpha)$. In the worst case $X > Y$, we shall pay for the difference $X - Y$, which is expected to be $c_2 \approx \mathbb{E}(\max(X - Y, 0)) \cdot \mathsf{v_b}$. Concretely, for $1 - \alpha = 0.2, k = 3$, we have that the expected bribe amount is $c_e + c_{e+2} \approx 0.965\ ETH$. A thorough analysis of the economic incentives in PayToBias markets, often called RANDAO bribery markets, are the subject of extensive study in concurrent research, such as the work of Alptürer [2]. Given this focus in other contemporary work, a detailed (game-theoretic) analysis of this specific market is beyond the scope of our paper.

5.4 Countermeasures

We envision several potential countermeasures, which can be categorized as either general economic deterrents or targeted protocol upgrades.

General Defenses. These defenses apply broadly to all three of our proposed attacks by altering the economic incentives of rational validators, thereby protecting safety, liveness, and fairness.

Increased Protocol Rewards A natural countermeasure is to increase the rewards for honest participation. This raises the opportunity cost of deviation, making any bribery attack more expensive. However, this approach likely requires higher inflation, and the trade-off between protocol security and economic stability warrants further study.

Whistleblowing Incentives Bribery attacks, as a form of collusion, can be thwarted by enabling participants to whistleblow in a publicly verifiable manner. Protocol-level incentives for whistleblowing could disrupt any bribery market, as explored in [27].

Targeted Defenses. These defenses are specific consensus upgrades that would mitigate individual attacks by making them technically infeasible.

Faster Finality To enhance safety, new consensus protocols aim to achieve faster finality, *e.g.,* three slots [16], making long-range attacks impossible. However, not even these can prevent (ex-post) reorgs on the shorter, unfinalized head of the blockchain enabled by PayToAttest.

Unbiasable Randomness To protect fairness, the deployment of a cryptographically secure and unbiasable randomness beacon would directly mitigate the PayToBias attack by removing the beacon manipulator's ability to influence proposer selection.

Time-locked Stake Retrieval To protect liveness, the protocol could enforce a significantly extended withdrawal period for an exiting validator's stake. The long delay imposes a direct economic cost due to the time value of money, increasing the opportunity cost of exiting and making the PayToExit attack financially impractical.

6 Related work

The first works studying short-term consensus manipulations (*cf.* Table 3), *e.g.,* forking, censorship, etc., enabled by bribing focused on Bitcoin [4,8,9,22], *i.e.,* Nakamoto consensus. Expressing the bribery contract logic on Bitcoin is cumbersome, thus, follow-up works have moved the bribery contract to a different, Turing-complete chain, *e.g.,* Ethereum, largely limiting these bribery attacks' practicality [24,25]. We are interested in bribery contracts and the implied incentives, where both the briber and the bribee act on the same chain [26]. McCorry et al. designed bribery contracts on Ethereum PoW for the first time [29]. Since then, Ethereum has transitioned to a PoS consensus. Thus, we continue their seminal work, but in the setting of Ethereum PoS. Many works analyze the negative externalities of the technically easiest form of bribery: censorship [5,45,48]. Closest to our work is that of [37], in which bribery attacks on Ethereum PoS had been suggested; here, the bribee is only provided with game-theoretic guarantees (not cryptographic) that the briber will compensate her. To overcome these limitations, we design, implement, and evaluate novel, *trustless* bribery contracts that are not verifiable in PoW (PayToExit), and were not efficient prior to recent protocol upgrades (PayToAttest), *cf.* Section 2.2.

7 Conclusion and Future Directions

In this work, we demonstrated that the features enabling scalable, permissionless blockchains – namely expressive smart contracts and efficient consensus mechanisms – can be repurposed to create novel attack vectors. We designed, implemented, and evaluated three trustless bribery contracts (PayToAttest, PayToExit, and PayToBias) that allow an adversary to undermine core consensus properties in Ethereum. Our initial game-theoretic analysis of the PayToExit market quantifies the practical incentives for validators to participate in such attacks.

Our findings open several promising avenues for future research:

Anonymity and Privacy The details of our bribery contracts (*e.g.,* participants, validator indices, and bribe amounts) are public. These could be concealed using confidential transaction schemes [11] and zero-knowledge proofs to increase the stealth of such attacks.

Additional Bribery Contracts The interface we developed, *cf.* Figure 3, can be used to implement other known bribery attacks, such as a PayToInclude contract that pays proposers to include or censor specific transactions [5,45,48].

Game-Theoretic Extensions Our analysis of the PayToExit market serves only as a starting point. Promising extensions include modeling temporal dynamics, variable bribes, and heterogeneous validators. Furthermore, formal game models for the PayToAttest and PayToBias attacks are needed to fully characterize the related economic incentives.

Attack Robustness vs. Inflation While increasing protocol rewards through inflation could potentially deter bribery, the relationship between a cryptocurrency's monetary policy and its resilience to such economic attacks is not yet well understood and warrants further study.

Acknowledgements. We are grateful to Kaya Alptürer for insightful discussions and encouragement. We are thankful to the Financial Cryptography and Data Security 2026 reviewers for their constructive feedback that helped improve this manuscript. István András Seres was supported by the Ministry of Culture and Innovation and the National Research, Development, and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004). Balazs Pejo was supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory. Gergely Biczók was supported by Project no. 138903 of the Ministry of Innovation and Technology, Hungary, from the NRDI Fund, financed under the FK_21 funding scheme.

References

1. Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, 2005.
2. Kaya Alpturer. Incentive analysis in randao bribery markets, 2025.
3. Kaya Alpturer and S. Matthew Weinberg. Optimal RANDAO Manipulation in Ethereum. In Rainer Böhme and Lucianna Kiffer, editors, *6th Conference on Advances in Financial Technologies (AFT 2024)*, volume 316 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
4. Zeta Avarikioti, Pawel Kedzior, Tomasz Lizurej, and Tomasz Michalak. Bribe & fork: Cheap PCN bribing attacks via forking threat. In *AFT*, volume 316 of *LIPIcs*, pages 11:1–11:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
5. Ben Berger, Edward W. Felten, Akaki Mamageishvili, and Benny Sudakov. Economic censorship games in fraud proofs. In *EC*, pages 670–687. ACM, 2025.
6. Dan Boneh, Manu Drijvers, and Gregory Neven. Bls multi-signatures with public-key aggregation. *URL: https://crypto. stanford. edu/˜ dabo/pubs/papers/BLSmultisig. html*, 2018.
7. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.
8. Joseph Bonneau. Why buy when you can rent? bribery attacks on bitcoin-style consensus. In *International Conference on Financial Cryptography and Data Security*, pages 19–26. Springer, 2016.

9. Joseph Bonneau. Hostile blockchain takeovers (short paper). In *International Conference on Financial Cryptography and Data Security*, pages 92–100. Springer, 2018.
10. Richard A. Brealey, Stewart C. Myers, and Franklin Allen. *Principles of Corporate Finance*. McGraw-Hill Education, New York, 13th edition, 2020.
11. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.
12. Vitalik Buterin, Tomasz Stanczak, Guillaume Ballet, Gajinder Singh, Tanishq Jasoria, Ignacio Hagopian, Jochem Brouwer, and Sina Mahmoodi. EIP-2935: Serve historical block hashes from state. *Ethereum Improvement Proposals, no. 2935*, 2020. Accessed: 2025-07-22.
13. Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 154–167, 2016.
14. Lin William Cong, Zhiheng He, and Ke Tang. The tokenomics of staking. Technical report, National Bureau of Economic Research, 2025.
15. Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, pages 910–927. IEEE, 2020.
16. Francesco D'Amato, Roberto Saltini, Thanh-Hai Tran, and Luca Zanolini. 3-slot-finality protocol for ethereum. *arXiv preprint arXiv:2411.00558*, 2024.
17. dapplion and Mikhail Kalinin. Eip-7549: Move committee index outside attestation. *Ethereum Improvement Proposals,(7549)*, 2023. Accessed: 2025-07-22.
18. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.
19. Bryan Ford and Rainer Böhme. Rationality is self-defeating in permissionless systems. *arXiv preprint arXiv:1910.08820*, 2019.
20. Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.
21. John William Hatfield and Scott Duke Kominers. A simple theory of vampire attacks. *Available at SSRN 4377561*, 2023.
22. Junjie Hu, Chunxiang Xu, Zhe Jiang, and Jiwu Cao. Novel bribery mining attacks in the bitcoin system and the bribery miner's dilemma, 2023.
23. Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar Weippl. How much is the fork? fast probability and profitability calculation during temporary forks. In *Companion Proceedings of the Web Conference 2022*, pages 467–477, 2022.
24. Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gaži, Sarah Meiklejohn, and Edgar" Weippl. Pay to win: Cheap, cross-chain bribing attacks on pow cryptocurrencies. In *Financial Cryptography and Data Security. FC 2021 International Workshops*, pages 533–549. Springer Berlin Heidelberg, 2021.
25. Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gaži, Sarah Meiklejohn, and Edgar Weippl. Sok: Algorithmic incentive manipulation attacks on permissionless pow cryptocurrencies. In *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi,*

VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25, pages 507–532. Springer, 2021.
26. Dimitris Karakostas, Aggelos Kiayias, and Thomas Zacharias. Blockchain bribing attacks and the efficacy of counterincentives. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1031–1045, 2024.
27. Mahimna Kelkar, Aadityan Ganesh, Aditi Partap, Joseph Bonneau, and S Matthew Weinberg. Breaking omertà: On threshold cryptography, smart collusion, and whistleblowing. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 3505–3519, 2025.
28. Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In *International Conference on Financial Cryptography and Data Security*, pages 505–522. Springer, 2020.
29. Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2018.
30. Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.
31. Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*, pages 53–72. Springer, 2021.
32. Ábel Nagy, János Tapolcai, István András Seres, and Bence Ladóczki. Forking the randao: Manipulating ethereum's distributed randomness beacon. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 873–887, 2025.
33. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
34. Henning Pagnia, Felix C Gärtner, et al. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology . . . , 1999.
35. Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake under scrutiny. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 212–221, 2023.
36. Danny Ryan, Mikhail Kalinin, Ansgar Dietrichs, Hsiao-Wei Wang, and light-client. Eip-7002: Execution layer triggerable exits. *Ethereum Improvement Proposals,(7002)*, 2023.
37. Roozbeh Sarenche, Ertem Nusret Tas, Barnabé Monnot, Caspar Schwarz-Schilling, and Bart Preneel. Commitment attacks on ethereum's reward mechanism. In *EuroS&P*, pages 504–526. IEEE, 2025.
38. Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *International Conference on Financial Cryptography and Data Security*, pages 560–576. Springer, 2022.
39. Ozan Solmaz, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer. Optimistic MEV in ethereum layer 2s: Why blockspace is always in demand. In *AFT*, volume 354 of *LIPIcs*, pages 28:1–28:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
40. Heinrich von Stackelberg. Marktform und gleichgewicht. *(No Title)*, 1934.

41. Alex Stokes, Ansgar Dietrichs, Danny Ryan, Martin Holst Swende, and lightclient. EIP-4788: Beacon block root in the EVM. *Ethereum Improvement Proposals, no. 4788*, 2022. Accessed: 2025-08-18.
42. János Tapolcai, Bence Ladóczki, and Ábel Nagy. Slot a la carte: Centralization issues in ethereum's proof-of-stake protocol. *Cryptology ePrint Archive*, 2025.
43. Jason Teutsch, Sanjay Jain, and Prateek Saxena. When cryptocurrencies mine their own business. In *International conference on financial cryptography and data security*, pages 499–514. Springer, 2016.
44. Alex Vlasov, Kelly Olson, Alex Stokes, and Antonio Sanso. Eip-2537: Precompile for bls12-381 curve operations. *Ethereum Improvement Proposals,(2537)*, 2020.
45. Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, et al. Blockchain censorship. In *Proceedings of the ACM Web Conference 2024*, pages 1632–1643, 2024.
46. Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market. *arXiv preprint arXiv:2305.16468*, 2023.
47. Toni Wahrstätter. Selfish Mixing and RANDAO Manipulation. https://ethresear.ch/t/selfish-mixing-and-randao-manipulation/16081, July 2023. Accessed: 2024-11-12.
48. Zhipeng Wang, Xihan Xiong, and William J Knottenbelt. Blockchain transaction censorship:(in) secure and (in) efficient? In *The International Conference on Mathematical Research for Blockchain Economy*, pages 78–94. Springer, 2023.
49. Hulin Yang, Mingzhe Li, Jin Zhang, Alia Asheralieva, Qingsong Wei, and Siow Mong Rick Goh. Bride arbitrager: Enhancing arbitrage in ethereum 2.0 via bribery-enabled delayed block production, 2024.

A Additional Preliminaries

A.1 Additional Notations

The function $\textbf{blockhash}(n) : \mathbb{N} \to \{0,1\}^{256}$ takes a block height n as input and returns the corresponding block hash.

Ethereum extensively uses Merkle trees [30] as vector commitments or cryptographic accumulators. For instance, validator public keys are committed to in an incremental Merkle tree whose root hash is accessible on-chain to any smart contract. We denote the relevant Merkle tree functionalities as follows.

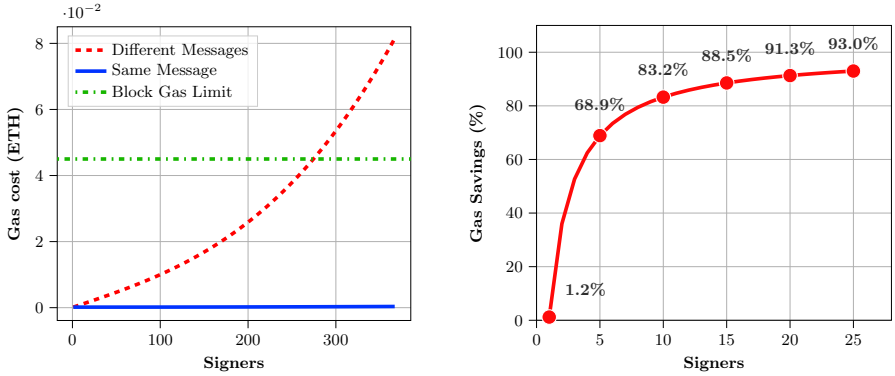
$\mathsf{Merkle.Insert}(T, x) \to T'$**.** Adds element x as a new leaf node to the Merkle tree T and returns an updated Merkle tree T'.

$\mathsf{Merkle.Prove}(T, x, i) \to \pi$**.** Generates a proof π proving the membership of element x at the index i in Merkle tree T.

$\mathsf{Merkle.Verify}(\mathsf{root}, x, i, \pi) \to \{0,1\}$**.** Given a root hash root of the Merkle tree, this algorithm verifies the proof π for the statement that x is committed in the ith position in the Merkle tree.

A.2 On-chain BLS batch verification

Our PayToAttest contract requires the on-chain batch verification of BLS signatures from validator attestations. The practical implementation of this contract is enabled by EIP-7549, the importance of which we now empirically demonstrate. EIP-7549 ensures that when validators attest to the same block header, they sign the exact same message. This allows for constant-time ($\mathcal{O}(1)$) BLS batch verification, independent of the number of signers, as shown in Figure 8 and Equation (1). In contrast, before this EIP, the linear-cost verification would have made our PayToAttest contract prohibitively expensive for bribing a large number of validators.



(a) BLS batch verification gas costs. At the time of writing, the Ethereum block gas limit is set to 45 087 931 gas.

(b) BLS batch verification gas savings when the same message m vs. different $\{m_i\}_i$ messages are signed.

Fig. 8: BLS batch verification measurements on Ethereum after EIP-2537 [44]. On-chain BLS batch verification gas cost (left) when the same message (blue line) or different messages (red line) are signed by multiple signers. If different messages are signed, then only fewer than 275 messages can be batch-verified in a single Ethereum block. Gas savings (right) are already significant for a handful of signed messages (if the same message is signed).

A.3 Attestation processing and rewards

During each epoch, a validator is expected to broadcast an attestation to its assigned subnet. As detailed in Figure 1, an attestation represents a validator's view of the chain and contains several components. The beacon_block_root is the validator's vote for the head of the chain under the LMD GHOST fork-choice rule. Concurrently, the target and source fields are votes for checkpoints under the Casper FFG finality gadget, which is responsible for finalizing the blockchain.

Timeliness	**1 slot**	**$\leq$ 5 slots**	**$\leq$ 32 slots**	**$>$ 32 slots (missing)**
Wrong source	0	0	0	0
Correct source	W_s	W_s	0	0
Correct source, and target	$W_s + W_t$	$W_s + W_t$	W_t	0
Correct source, target and head	$W_s + W_t + W_h$	$W_s + W_t$	W_t	0

Table 2: Attestation reward matrix for validators, weights, and proportion of a validator's reward for timely votes are as follows: source ($W_s = 21.9\%$), target ($W_t = 40.6\%$), and head votes ($W_h = 21.9\%$), respectively. The timeliness columns indicate when the attestation was included in the canonical chain after the validator needed to broadcast its attestation.

As Table 2 shows, a validator can earn a partial reward even if their attestation does not perfectly match the canonical chain. The primary condition for receiving any reward is that the attestation itself must be included in a canonical block [37]. During a fork, a validator risks forfeiting rewards if their attestation supports the losing (i.e., non-canonical) chain. However, during a short fork (fewer than 5 slots), even if a validator's head vote is incorrect, the validators can still earn a significant partial reward for correct source and target votes. This results in a reward with weight $W_s + W_t$, corresponding to 74% of the maximum possible reward.

B Bribery Contract Implementations

This section presents the pseudocode for our three proposed bribery contracts. The code focuses on the core logic and omits certain low-level, EVM-specific implementation details for the sake of clarity. For the complete Solidity smart contracts and the accompanying test suite, readers are referred to our publicly available open-source repository. [7]

B.1 The PayToAttest Bribery Contract

In a PayToAttest contract, a briber purchases attestations for one or more of their proposed blocks. The process begins when the briber calls the `offerBribe`(pk^*, m, t) function, specifying the aggregate public key pk^* of the target validators, the block header m, and a submission deadline t. As part of this initial transaction, the briber also transfers the total bribeAmnt in Ether to the contract. The set of validators corresponding to pk^* can then claim their reward by providing the

[7] https://github.com/0xSooki/bribery-zoo.

contract with a valid, aggregate BLS signature on m before the deadline expires, as detailed in Algorithm 1.

Algorithm 1 The PayToAttest contract: The pseudocode of the `offerBribe`() and `takeBribe`() functions as called by the briber and bribee, respectively.

function `offerBribe`($\mathsf{pk}^* \in \mathbb{G}_1, m \in \mathsf{AttestationData}, t \in \mathbb{N}$):
  $\mathsf{pk}_{agg} = \mathsf{pk}^*$ //Aggregate public key of the bribed validators $\{\mathsf{pk}_i\}_{i=1}^n$.
  $\mathsf{block.header} = m$ // This is the AttestationData, *cf.* Figure 1.
  $\mathsf{deadline} = t$ //The briber only rewards timely votes.
  Transfer bribeAmnt to the contract.

function `takeBribe`($\sigma \in \mathbb{G}_2$):
  assert $\mathsf{!claimed}[\sigma]$
  assert $\mathsf{block.timestamp} < \mathsf{deadline}$.
  assert $\mathsf{BLS.Verify}(\mathsf{pk}_{agg}, m, \sigma,)$ //This is implemented applying the same-message BLS batch verification check, *cf.* Equation (1).
  $\mathsf{claimed}[\sigma] = \mathsf{true}$
  Transfer bribeAmnt ether to validator.

The PayToAttest contract can be extended for greater flexibility by removing the block header m as an argument from the `offerBribe()` function, resulting in the signature `offerBribe`(pk^*, t). This change allows the briber to decide on the target block header after the bribe has been offered, but it necessitates an additional verification step within the `takeBribe`(σ) function. To prevent misuse, the contract must confirm that the attestation provided by the bribee is for a block actually proposed by the briber. This can be achieved by checking the block's randomness field – which contains the proposer's BLS signature on the epoch number (*cf.* Section 4.3) – to ensure it was signed with the briber's public key.

B.2 The PayToExit Bribery Contract

The PayToExit contract creates a trustless bribery market to incentivize Ethereum validators to voluntarily exit the active validator set (*cf.* Section 4.2). This contract rewards a validator with bribeAmnt in Ether if they can prove that the following three conditions have been met for their validator index $i \in \mathbb{N}$:

Membership in the current validator set The validator must prove they are a current member of the active validator set. This is achieved by providing a Merkle proof π showing that their public key, pk_i, is a leaf in the deposit contract's Merkle tree. The contract verifies this proof against the publicly accessible deposit root hash.

A signed exit transaction The validator must provide a valid BLS signature on a voluntary exit message (*cf.* Figure 1). The PayToExit contract verifies this signature to confirm the validator's authentic intent to exit the protocol.

Non-membership in the rewarded validator set To prevent duplicate payments, the contract must verify that the validator with index i has not already been rewarded for exiting. This is implemented by checking for non-membership in the claimed$[i]$ mapping, which tracks all validators who have already received a payout.

The logic for verifying these three conditions is implemented in the takeBribe() function, as detailed in Algorithm 2. If the conditions are satisfied, the contract rewards the validator by distributing the funds accordingly.

The contract's logic could be extended with additional constraints. For instance, a briber might add a time-based check to only reward voluntary exits that occurred within a specific timeframe. While our current implementation does not include this feature, it represents a straightforward extension.

Algorithm 2 The PayToExit contract: The pseudocode of the `offerBribe`() and `takeBribe`() functions as called by the briber and bribee, respectively.

```
1: function offerBribe()
2:     Transfer bribeCost to the contract.
3:
4: function takeBribe(i ∈ ℕ, σ ∈ 𝔾₂, π ∈ bytes[])
5:     assert Merkle.Verify(root, pk_i, i, π) //root is read from the deposit contract.
6:     assert BLS.Verify(pk_i, m_exit,i, σ) // The message m_exit,i is the VoluntaryExit object for validator i, cf. Figure 1.
7:     assert !claimed[i] // i is the validator's deposit index in the deposit contract.
8:     claimed[i] = true.
9:     Transfer bribeAmnt ether to validator i.
```

While our proof-of-concept implementation of the PayToExit contract uses a fixed bribeAmnt, the design can be easily extended to support a dynamic bribe amount. A straightforward extension would be to make the bribeAmnt a function of $N(t)$ – the total number of active validators – allowing the reward to scale with network participation.

B.3 The PayToBias Bribery Contract

Ethereum uses the distributed randomness beacon RANDAO to select block proposers. The beacon's randomness is known to be biasable by rational validators who can strategically reveal or withhold their randomness contribution [3,32,46]. This creates an economic incentive for manipulation, as being selected more often leads to greater rewards.

Our PayToBias contract creates an efficient market to auction off this manipulation right. For simplicity, our model considers an adversary who controls the final slot of an epoch, effectively auctioning a single bit of bias. Other rational validators can then bid on whether the adversary should publish or withhold

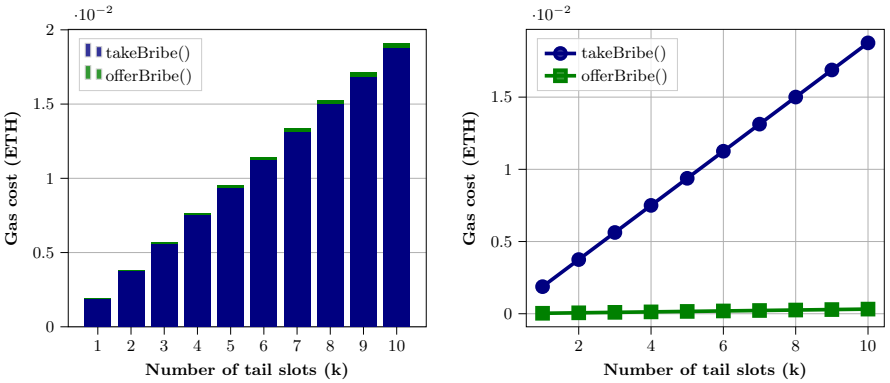
their block in that slot. Once the auction concludes, the contract pays the adversary the total amount bid on the winning outcome.

To collect this reward, the adversary must prove to the PayToBias contract which action they took. This proof must satisfy the following three conditions:

Block integrity The hashes of the provided blocks are verified against the built-in `blockhash` function in Solidity, ensuring that each block corresponds to its designated block height and that the block contents are authentic.

Block linkage The given two headers must be correspond to consecutive blocks, thus the second block must reference the hash of the first block.

Time difference To confirm a block was withheld, the contract must verify that no block was published in the designated final slot of the epoch. This is accomplished by analyzing the timestamps of the surrounding blocks, which are enforced by Ethereum's consensus rules: the contract checks if the timestamp of the first block in the next epoch is more than 12 seconds greater than the timestamp of the penultimate block of the preceding epoch. Since each slot is 12 seconds long, this gap proves that a slot was skipped, confirming the adversary withheld their block as promised.



(a) A stacked chart illustrating the linear gas cost in the number of auctioned tail slots in the takeBribe(·) function.

(b) Gas measurements for the PayToBias contracts offerBribe() and takeBribe() functions respectively.

Fig. 9: PayToBias gas measurements

Algorithm 3 presents the pseudocode for the PayToBias contract, which establishes a trustless auction market through three core functions. The offerBribe() function allows a manipulator to initiate an auction for their slot; bid() enables other parties to participate; and takeBribe() lets the manipulator claim the reward after verifying the required conditions. A key feature of this design is the on-chain auction itself, which efficiently prices the right to bias the RANDAO output. This auction model is a generalizable pattern that could be applied

to our other attacks, enabling the creation of efficient bribery markets without needing to deploy multiple contracts.

Algorithm 3 The PayToBias contract: The pseudocode of the `offerBribe`() and `takeBribe`() functions as called by the briber and bribee, respectively.

function offerBribe(pk $\in \mathbb{G}_1, e \in \mathbb{N}, \sigma \in \mathbb{G}_2$)
    assert BLS.Verify(pk, e, σ) // The beacon manipulator releases prematurely its randomness contribution σ. More precisely, $H(\sigma)$ is the randomness contribution.

function bid($withhold \in \mathbb{B}, bidAmnt \in \mathbb{N}, e \in \mathbb{N}$)
    $auction \leftarrow auctions[e]$ // Get bribery auction corresponding to epoch e
    $auction[withhold].bid+ = bidAmnt$

function takeBribe($e \in \mathbb{N}$, $B_1, B_2 \in$ `BlockHeader`)
$auction \leftarrow auction[e]$
if $block.timestamp < auction.deadline$ **then**
    assert H(B_1) = **blockhash**(n) and H(B_2) = **blockhash**(n+1)
    assert $B_2.parent$ = H(B_1)
    if $B_2.timestamp - B_1.timestamp > 12$ **then**
        Transfer bribeAmnt ether to the winner of the auction
    end if
else
    Transfer bribeAmnt ether to the winner of the auction
end if

C PayToAttest sequence diagrams

We provide two sequence diagrams illustrating the mechanics of *ex-ante* (*cf.* Figure 10) and *ex-post* (*cf.* Figure 11) reorganizations that use a PayToAttest contract to buy consensus votes. For clarity, both diagrams depict the simplest forking scenario to which our attacks apply.

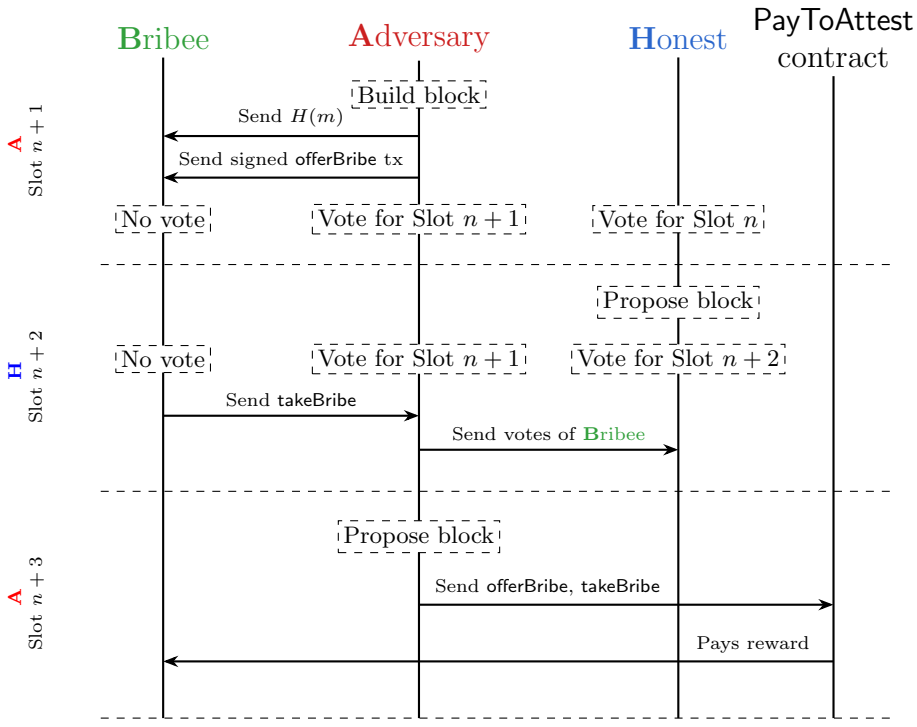


Fig. 10: Timeline of *an ex-ante reorg* attack where the attacker leverages a PayToAttest bribery contract. Time increases from top to bottom. In Slot $n+1$, after the block is built, the briber broadcasts the offerBribe() transaction to the bribee. The bribee withholds his votes, but broadcasts the signatures necessary for them in the takeBribe() transaction during Slot $n+2$. The adversary sends the aggregated votes to the honest validators. Finally, the adversary finishes the fork by building on the withheld block. The block contains the timely takeBribe() transaction, so the bribee receives the reward.

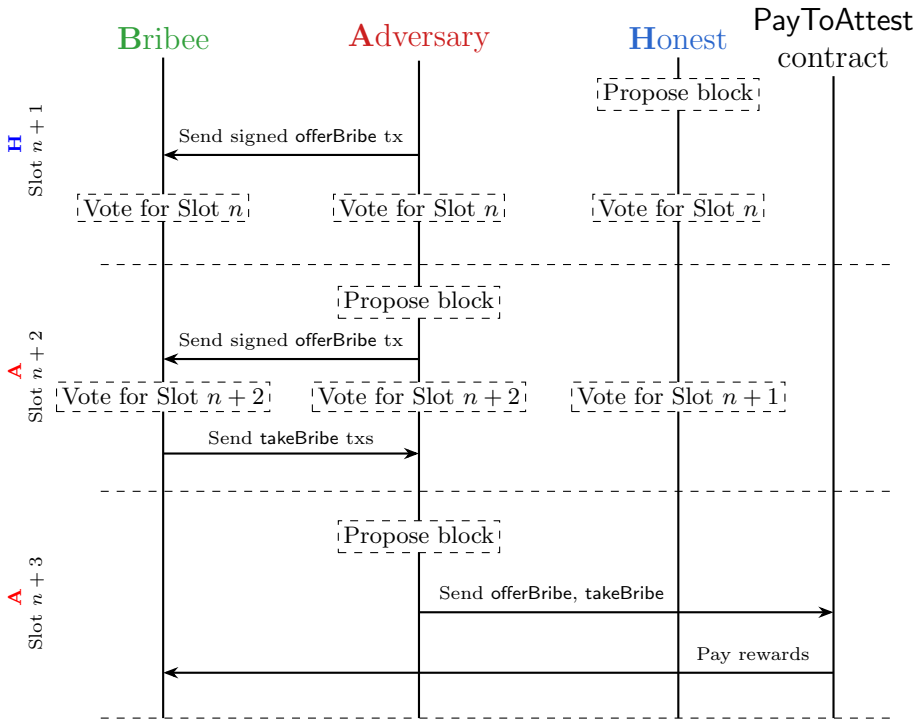


Fig. 11: Timeline of *an ex-post reorg* attack where the attacker leverages a PayToAttest bribery contract. Time increases from top to bottom. In Slot $n+1$, the adversary decided to fork out the new **H**, possibly due to its high MEV content. Thus, he offers a bribe to vote for the block in Slot n instead, which the bribee accepts. In the next slot, another offerBribe() is broadcast to vote for the new **A**. The bribee collaborates again, making the adversarial branch the heaviest, according to the LMD-Ghost rule. The attacker included the 2 offerBribe() and the 2 timely takeBribe() transactions, thus, the reward payments are issued.

Table 3: Qualitative comparison of existing bribery attacks on Ethereum PoS. A property is marked with ✓ if it is achieved and with ✗ otherwise, – is used if a property does not apply. If the symbol is within brackets, *e.g.,* (✓), this means that this property is achieved (or can be augmented), but this was initially not discussed or considered by the authors. ∼ denotes that the property cannot be clearly mapped to any of the previously defined categories without further details or discussion.

Attack	Transaction reversal	Transaction ordering	Transaction exclusion	Transaction triggering	Required interference with consensus	Requires smart contract	**Payment**	Trustless for attacker	Trustless for collaborator	**Subsidy**	Compensates if attack fails
PayToAttest (this work)	✓	✗	(✓)	✗	Deep fork	✓	in-band	✓	✓	✓	(✓)
PayToExit (this work)	✗	✗	✗	✗	No fork	✓	in-band	✓	✓	✓	✓
PayToBias (this work)	✗	✗	✗	✗	No fork	✓	in-band	✓	✓	✓	✓
Simple Attack [37]	✓	✓	✓	✗	Fork	✗	in-band	✓	(✓)	✓	✗
Strong Simple Attack [37]	✓	✓	✓	✗	Fork	✗	in-band	✓	(✓)	✓	✗
Extended Attack [37]	✓	✓	✓	✗	Deep fork	✗	in-band	✓	(✓)	✓	✗
BriDe Arbitrager [49]	–	✓	✓	✓	No fork	✓	in-band	✓	(✓)	✓	✓
Guided Bribing [26]	✓	✓	✓	✗	No fork	✗	out-of-band	✗	✗	✓	✓
Effective Bribing [26]	✓	✓	✓	✗	No fork	✗	out-of-band	✗	(✓)	✓	✗
Bribe & Fork [4]	✓	✓	✓	✓	Deep fork	✓	in-band	✓	✓	✓	✓
Bribery Semi-Selfish Mining [22]	✗	✓	✓	✗	Deep fork	✗	out-of-band	✗	✗	✗	✗
Bribery Stubborn Mining [22]	✗	✓	✓	✗	Fork	✓	out-of-band	✗	✗	✗	✗
CensorshipCon [29]	✗	(✓)	✓	(✓)	No forks	✓	in-band	∼	✗	✓	✗
HistoryRevisionCon [29]	✓	✗	✓	(✓)	Deep fork	✓	in-band	✓	∼	✓	✗
GoldfingerCon [29]	–	–	✓all	(✓)	No fork	✓	out-of-band	✓	✗	✓	✗
P2W Tx Excl. & Ord [24]	✗	✓	✓	(✓)	No fork	✓	out-of-band	✓	✓	✗	✓
P2W Tx Rev., Excl. & Ord. [24]	✓	✓	✓	(✓)	Deep fork	✓	out-of-band	✓	✓	✗	✓
P2W Tx Ord. [24]	✗	✓	✗	(✓)	No fork	✓	in-band	✓	✓	✗	✗
P2W Tx Excl. [24]	✗	✗	✓	(✓)	No fork	✓	in-band	✓	✓	✗	✗

D PayToAttest for General Forking Attacks

Note the original in-band payment idea using anyone-can-spend addresses for forking as proposed in [8] is not applicable in the Ethereum case. Ethereum blocks gain weight from the attestors, and attestators are typically not the same entities as the block proposers. Thus, anyone-can-spend addresses or simple transfers do not work.

PayToAttest bribery contracts can be used to perform reorg attacks for a much larger family of chain strings than was shown in Section 4.1.

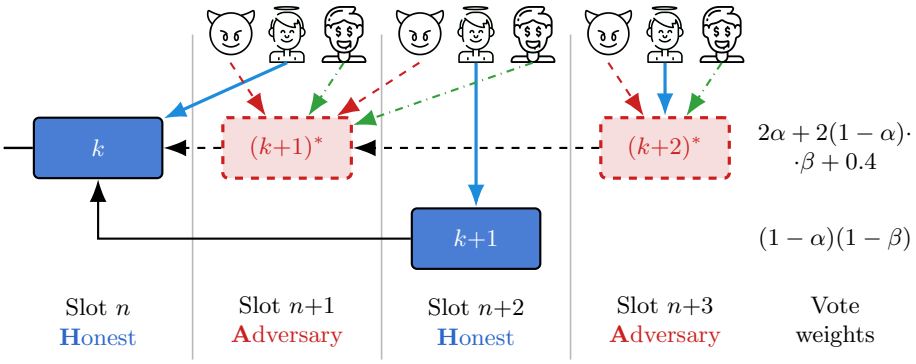


Fig. 12: Performing an ex-ante reorg with a PayToAttest bribery contract. Colored arrows indicate which blocks different validators vote for as the head of the blockchain. Black arrows represent hash pointers. Red (blue) blocks are proposed by the briber (honest validators).

D.1 General ex-post reorgs

In Section 4.1, we saw that the adversary can execute an ex-post reorg leveraging a PayToAttest contract to buy rational validators' attestations. Moreover, a motivated adversary can execute ex-post reorgs for many more chain strings as we show next.

Theorem 1. *Given the following chain string* $\mathbf{H}^h\mathbf{A}^a$*, the adversary can fork out* $\mathbf{H}^h$ *with the joint voting power of the adversary and bribees if Eq. (8) holds. More formally, the adversary's fork wins iff.*

$$(a-1)\cdot(\alpha+(1-\alpha)\beta)+p_{\text{boost}}>(h+a-1)\cdot(1-(\alpha+(1-\alpha)\beta))\ . \tag{8}$$

Proof. First, the honest branch $\mathbf{H}^h$ is proposed, receiving a total attestation of $h\cdot(1-(\alpha+(1-\alpha)\beta))$. During this period, the adversary and the bribees vote for block k (*cf.* Fig. 4). In the next $a-1$ slots they start the adversarial

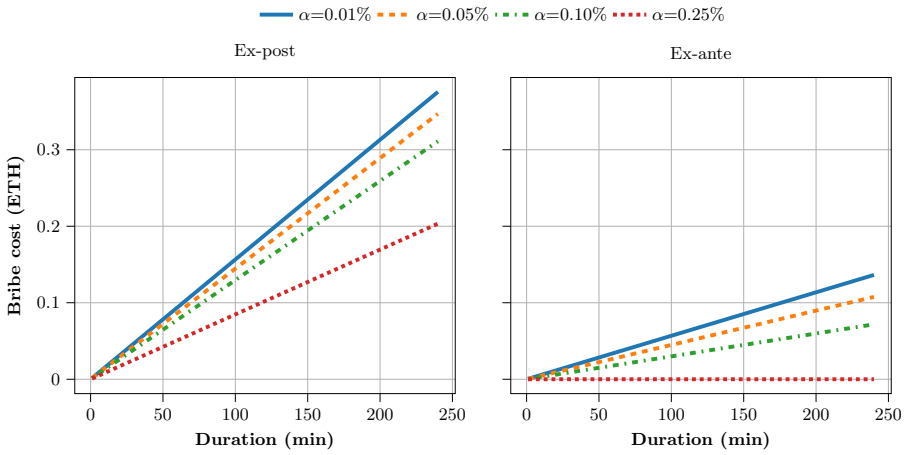


Fig. 13: Ex-post and ex-ante reorg bribe costs for continuous short-term reorg attacks. In these measurements, we assume that the briber leverages the PayToAttest bribery contract to bribe and reorg honest blocks at every possible occasion. Moreover, we assume that the minimal amount of rational validators accept the bribe necessary for the reorg to be successful, *cf.* Equation (3).

branch on top of block k, continuously voting for it: $(a-1)\cdot(\alpha+(1-\alpha)\beta)$. "Worst"-case scenario, the honest attestators still consider $\mathbf{H}^h$ as the canonical chain, thus getting votes of $(a-1)\cdot(1-(\alpha+(1-\alpha)\beta))$. Finally, the adversary builds and proposes the last block of the adversarial branch promptly, giving it the proposer boost p_{boost}. To calculate when the adversarial branch is considered the canonical chain by the LMD-Ghost rule, we get Equation (8).

D.2 General ex-ante reorgs

Ex-ante reorgs against the Ethereum PoS protocol were introduced in [38]. Utilizing our PayToAttest bribery contracts, an adversary can launch ex-ante reorgs for various chain strings as we show below.

Theorem 2 (Condition for ex-ante reorgs). *Given the following chain string* $\mathbf{A}^a\mathbf{H}^h\mathbf{A}$*, the adversary is capable of forking out* $\mathbf{H}^h$ *with the joint voting power of the adversary and the collaborating bribees if the following holds.*

$$(a+h)\cdot(\alpha+(1-\alpha)\beta)+p_{\mathsf{boost}} > h\cdot(1-(\alpha+(1-\alpha)\beta)) \ . \tag{9}$$

Proof. During the first a slots, the adversary secretly builds $\mathbf{A}^a$ blocks, which receive $(a+h)\cdot(\alpha+(1-\alpha)\beta)$ votes by the end of the honest branch. On the other hand, the honest validators consider the private blocks $\mathbf{A}^a$ missed, voting for block k instead, *cf.* Figure 12. The first honest block is built on top of block k, and the honest branch gets a total of $h\cdot(1-(\alpha+(1-\alpha)\beta))$ attestations. Finally, the secret branch is published when the adversary builds on top of $\mathbf{A}^a$, receiving the proposer boost. The branch is heavier whenever Equation (9) holds.

E Additional proofs and calculations

E.1 Present value (PV) half-life for a fixed discount factor

Let R denote the expected nominal annual staking reward. Consider a staker who would otherwise receive R each year in perpetuity. The present value (PV) of the infinite stream (a perpetuity) discounted at rate $r > 0$ is

$$PV_\infty \;=\; \frac{R}{r}\;, \tag{10}$$

as standard in financial mathematics [10].

If the staker instead receives rewards only up to a finite horizon T, the PV of the finite annuity is

$$PV(T) \;=\; R\sum_{t=1}^{T}\frac{1}{(1+r)^t} \;=\; R\frac{1-(1+r)^{-T}}{r}\;. \tag{11}$$

The *PV half-life* is defined as the horizon $T_{1/2}$ at which the PV of cumulative rewards equals one half of the perpetuity value. This yields

$$T_{1/2} \;=\; \frac{\ln 2}{\ln(1+r)}. \tag{12}$$

For $r = 0.08$, we obtain $T_{1/2} \approx 9$ years, implying that half of the PV of all future staking rewards is realized within the first decade of operation. This provides a transparent benchmark for comparing one-time bribes with the net present value of forgone staking income.

E.2 Visualizing the inefficacy of the constant-bribe upper bound

Recall that in Section 5.2, we computed an optimal *constant* bribe b^*, *i.e.,* in total, a briber pays $k^* \cdot b^*$ bribe amount to k^* exiting validators. However, Ethereum protocol rewards are set by the consensus protocol as a decreasing function in the number of validators, *cf.* Equation (5) and Figure 14. This means that the briber must pay increasing bribe amounts to the exiting rational validators since the annual return increases as the number of remaining validators decreases. As Figure 14 illustrates, our calculated optimal bribe is the bribe paid to the last exiting validator from the total exiting k^* validators. Thus, in a more realistic setting where the PayToBias contract pays dynamic bribe amounts, the adversary only needs to pay the area under the curve as opposed to the rectangle with b^* corresponding to a constant bribe amount b_{max}, *cf.* Figure 14. We leave it to future work to add the dynamic bribe amount feature to our PayToExit bribery contract implementation, depending on the number of remaining validators.

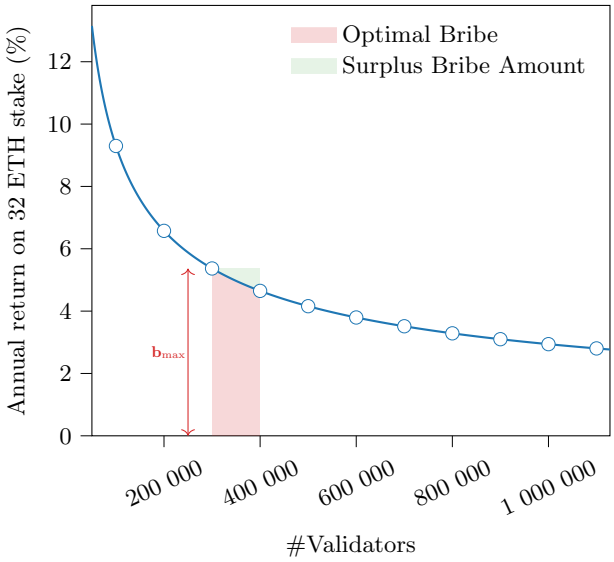


Fig. 14: **Upper-bound rectangle on APR vs. #Validators.** On the interval $[N_0 - k, N_0]$ we mark a constant-bribe upper bound $b_{\max} = R(N-k) \cdot \mathrm{PV}(r, Y)$. The rectangle's area $k \cdot b_{\max}$ is a conservative bribe cost upper bound, *not* an integral under the curve which could be achieved by dynamically changing PayToExit bribe amounts depending on the number of remaining active validators (left for future work).