Commit-Reveal Schemes Against Front-Running Attacks

Andrea Canidio
IMT School for Advanced Studies, Lucca, Italy
CoW Protocol, Paris, France

Vincent Danos
CNRS, Paris, France
École Normale Supérieure, Paris, France

Abstract

We provide a game-theoretic analysis of the problem of front-running attacks. We use it to study a simple commit-reveal protocol and discuss its properties. This protocol has costs because it requires two messages and imposes a delay. However, we show that it prevents the most severe front-running attacks ("bad MEV") while preserving legitimate competition between users, guaranteeing that the earliest transaction in a block belongs to the honest user who values it the most ("good MEV").

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases Front running, Game theory, MEV, Transactions reordering, commit-reveal

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.7

Category Extended Abstract

Related Version *Full Version*: https://arxiv.org/abs/2301.13785

Funding We gratefully acknowledge the financial support of the Ethereum Foundation (grant FY22-0840).

Acknowledgements We are grateful to Agostino Capponi, Jiasun Li, Christof Ferreira Torres, Arthur Gervais, Ari Juels, and the participants to UBRI Connect 2022, Tokenomics 2022 for their comments and suggestions.

1 Introduction

On the Ethereum network, each validator decides how to order pending transactions to form the next block, hence determining the order in which these transactions are executed. As a consequence, users often compete with each other to have their transactions included earlier in a block, either by paying transaction fees or by making side payments directly to validators.[1] This form of competition can be beneficial because it ensures that a scarce resource (i.e., having a transaction included earlier in the block) is allocated to the user who values it the most.[2] But at the same time, it opens the possibility of front-running attacks: because pending transactions are public, a malicious user can observe a victim's incoming transaction, craft a new transaction and then pay to place it before that of the victim.

[1] Competition through higher transaction fees occurs via "gas replacement" transactions, whereby a pending transaction is resubmitted with a higher fee. The resulting game is akin to an auction (see [3]). The most popular way to make side payments to validators is to use flashbots (see https://github.com/flashbots/pm).
[2] Whether it is the most efficient to achieve this goal is a different issue we do not address here.

In this paper, we propose a game-theoretic model of front-running. We use it to study a simple commit-reveal protocol that can be implemented at the smart contract level without modifying the underlying Ethereum infrastructure or introducing third parties (or layer-2 networks). We derive conditions under which an honest player is better off using the protocol than Ethereum's standard procedure. On the cost side, the protocol requires sending two messages instead of one and imposes a delay. Hence, if the cost of sending messages or waiting is high, the protocol is worse than the standard way to send transactions; if they are low, the protocol is preferred. On the benefit side, the protocol can eliminate front-running attacks, especially when it is difficult for an attacker to guess, that is, when the expected payoff of an attacker who commits without knowing whether the victim committed and what message was committed is low. We also argue that our protocol does not impede legitimate competition between honest users, that is, competition to have a transaction included earlier in a block between users who do not rely on observing each other's message.

Prior work

Our commit-reveal protocol is novel but similar to existing proposals. Our main contribution is the type of analysis. In particular, we show that our protocol can reduce front-running attacks while maintaining legitimate users' competition. Existing solutions instead are either primarily concerned with eliminating attacks (at the cost of also eliminating legitimate competition, see, for example, Aequitas protocol and the hedera-hashgraph project) or better organizing competition (at the cost of exacerbating attacks, see, for example, Flashbots). Furthermore, most of the literature has proposed solutions to reduce or eliminate front-running in Ethereum by changing its infrastructure or introducing third parties (See [5] for a review of the literature). Instead, our solution does not require third parties and can be implemented at the smart contract level, allowing for flexibility in its implementation.

With respect to existing solutions, our protocol can be seen as a simplified version of the submarine commitments protocol in [1]: in both cases, a message is first committed and then revealed, and the commitment can be hidden in the sense that the identity of the sender and receiver of the commit message cannot be observed. The main difference is that we adopt a weaker notion of "commitment" because we allow users not to send a transaction after committing it. The notion of "commitment" in [1] is instead stronger because users are penalized for not following through with their commitment.

As already mentioned, we provide a game-theoretic analysis of the properties of this protocol, applicable to any smart contract.[3] With this respect, our work is inspired by [4], who develop a game-theoretic analysis of the problem of front-running arising when an honest user and an attacker claim the same reward. They also propose a protocol that eliminates these types of attacks. Their key assumption is that the legitimate claimant strictly prefers the reward to be burned rather than paid to the attacker. Therefore, these results are useful in some environments where front-running may emerge, but not all. For example, front-running attacks are a serious concern in the AMMs, but in this context, it may not be possible to "burn the reward".

[3] [1] analyze the properties of the submarine commitment scheme in the context of a bug-bounty scheme they propose.

2 The problem: front-running attacks

We start by developing a simple model of front-running attacks and later introduce the commit-reveal protocol.[4] There is a smart contract SC and two players: Alice and Bob. Absent front-running attacks, player A sends a message $\sigma_A \in \Sigma$ to the mempool (i.e., the set of pending transactions), where Σ is the set of possible messages that A may send. When the message σ_A is included in a block, the smart contract SC performs an action that generates a benefit P_A to player A.[5] Front-running attacks arise because messages in the mempool are public. Hence, after A sends a message to the mempool, this message is observed by B, who can send a counter-message $\sigma_B \in \Sigma$. If σ_B is included in the blockchain before A's message, then B earns $P_B(\sigma_A)$ while A earns nothing. Else, B earns nothing and A earns P_A.

Sending messages is costly. Each player can send a regular message by paying $c > 0$. If multiple regular messages are sent, they are included in the block in the order they are sent. Player B, however, can also pay $f > c$ to send a "fast" message that, with probability q, is included in the block before A's regular message, despite A's message being sent first. For example, f could be the cost of sending a transaction via a service such as flashbots, or could be a regular mempool transaction with a transaction fee significantly above the base fee. We consider the parameters q, c, and f as exogenous and determined by the technology available to A and B.

Equilibrium

We can easily solve the game by backward induction and assuming that each player maximizes his/her expected payoff. If A sends a message, then B attempts to front-run if and only if:

$$qP_B(\sigma_A) > f$$

Given this, we can derive A's optimal strategy. Suppose that $qP_B(\sigma_A) < f$, so that A expects no front running. In this case, she sends a message if and only if

$$P_A > c$$

If, instead, $qP_B(\sigma_A) > f$, then A anticipates that B will try to front-run. In this case, A sends a message if and only if

$$(1-q)P_A > c$$

Hence, front running does not happen when its benefit is low (i.e., $P_B(\sigma_A) \leq f/q$). If, instead, its benefit is large (i.e., $P_B(\sigma_A) > f/q$), B will attempt to front run A whenever A sends a message. In particular, when $P_A > c$ but $(1-q)P_A < c$ the threat of front running prevents A from sending the message in the first place, therefore destroying the value of the exchange between A and SC.

3 Preventing front-running via commitment

We now use the model developed in the previous section to study how a commit-reveal protocol can mitigate front-running attacks. In terms of notation, we call player A's commit message $\sigma_{A,1}$ and reveal message $\sigma_{A,2}$. Similarly, player B's counter-messages are $\sigma_{B,1}$ and $\sigma_{B,2}$.

[4] For a more detailed analysis, see [2].
[5] For simplicity, here we assume that P_A is independent on the message σ_A. See [2] for the case in which A's payoff depends on her message.

Formally, the protocol has a commitment period and a reveal period, which here are two subsequent blocks. If player A wants to send message $\sigma_A \in \Sigma$ to SC, in the commit period A sends the commit message

$$\sigma_{A,1} = S(addr, \sigma_A)$$

to SC where $addr$ is an address that A controls and $S()$ is a function with an intractable pre-image problem (for example, $Hash\,(addr|\sigma_A)$ where $Hash()$ is the SHA-256 hash function). Once the commit message is included in a block, A sends the reveal message $\sigma_{A,2} = \sigma_A$ to SC from the address $addr$, which is then included in the next block. Upon receiving the message, SC computes $S(addr, \sigma_A)$ and checks whether it received message $S(addr, \sigma_A)$ in the previous block.

It follows that if B wants to front run A he will need to commit a message at the commit stage and then reveal it at the reveal stage. There is a common discount factor $\beta \in [0, 1]$, so when a given payoff is earned with a block delay, this payoff is discounted by β. Finally, A does not observe B's commit message and hence cannot detect B's attempt to front running. At the same time, we assume B observes A's commit message.

3.1 Equilibrium

The first, rather immediate, result is that there is no equilibrium in which B sends the same commit message as A. To see this, suppose that player A sends the commit message $S(addr, \sigma_A)$ and player B sends the same commit message. If in the next period B sends the message $reveal_B = \sigma_A$, then the SC will consider B's reveal message as invalid because sent from an address different from $addr$. It is also easy to see that there is no equilibrium in which A commits but then does not reveal because A can do better by not committing at all. The next lemma summarizes these observations.

▶ **Lemma 1** (No cloning in equilibrium). *There is no equilibrium in which $\sigma_{B,1} = \sigma_{A,1}$. There is also no equilibrium in which A sends the commit message but not the reveal message.*

In equilibrium, therefore, if B wants to attack, he needs to craft a commit message while being completely uninformed about the contents of A's message. However, B anticipates that he will observe A's message and, at that point, will decide whether or not to send the message he initially committed. Therefore, the protocol severely limits but does not fully eliminate B's ability to act upon his observation of A's message.

Formally, suppose $\sigma_{A,1} \neq \emptyset$ (so that A sent the commit message), B committed a message with content σ_B and then observed A's reveal message. In this case, B's expected payoff from front-running is

$$q \cdot P_B(\sigma_B, \sigma_A) - f.$$

Hence, B will try to front run if and only if $q \cdot P_B(\sigma_B, \sigma_A) > f$.

In the commitment phase, B's choice of what message to commit is made in anticipation that he will decide to front run after observing A's reveal message. We assume that B has a prior belief over what message A may send. His expected future payoff is, therefore:

$$\pi \equiv \max_{\sigma_B \in \Sigma} E_{\sigma_A} \left[\max\{q \cdot P_B(\sigma_B, \sigma_A) - f, 0\} | \sigma_{A,1} \neq \emptyset \right],$$

where the expectation is with respect to σ_A. Hence, if A sends a commit message and B tries to front run, B's expected payoff is $\beta\pi - c$. We, therefore, have the following proposition:[6]

[6] The existence of the equilibrium follows from the fact that the players' strategy space is finite, as noted already in [6].

▶ **Proposition 2.** *If $\pi \leq \frac{c}{\beta}$ (i.e., "guessing is hard for B"), then there is no front-running in equilibrium. If instead $\pi > \frac{c}{\beta}$ (i.e., "guessing is easy for B"), front running occurs with strictly positive probability in equilibrium.*

Note that in case "guessing is easy for B", there could be a pure strategy equilibrium in which B commits with probability 1 whenever A commits, or a mixed strategy equilibrium in which B commits with some probability. In either case, after committing, B attempts to front-run A or not depending on A's reveal message.

It is easy to check that in the "guessing is hard for B" case, A's equilibrium payoff is

$$\max\{-c + \beta(P_A - c), 0\}$$

Therefore, the protocol generates both costs and benefits for player A. The benefit is that the simple commit-reveal protocol effectively dissuades B from attacking, yet this comes at a cost: one additional message is required, and the payoff is earned with a one-block delay (and hence is discounted by the parameter β).

4 Conclusion

We conclude by informally discussing two properties of the commit-reveal protocol. As already mentioned, π measures "how easy" it is for B to guess what message he should commit. Therefore, it measures how much, absent the commit-reveal protocol, B relies on observing A's message to attack. At an intuitive level, it can be interpreted as a proxy for the severity of a front-running attack: high values of π imply less severe attacks because B is already informed and relies less on observing σ_A; low values of π imply less severe attacks because B is uninformed and relies heavily on observing σ_A. Therefore, our protocol is most effective at preventing the most severe front-running attacks.

Finally, it is also possible that π is so large that B always wants to commit and reveal a message, whether he observes A's commit message or not. In this case, B acts more like a legitimate competitor because he would commit even if he were to move first. In this case, our commit-reveal protocol preserves competition because both A and B commit their messages and then compete in the reveal stage to have their message included earlier in the block.

References

1 Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, 2018.

2 Andrea Canidio and Vincent Danos. Commitment against front running attacks, 2023. `arXiv:2301.13785`.

3 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint*, 2019. `arXiv:1904.05234`.

4 Joshua S Gans and Richard T Holden. A solomonic solution to ownership disputes: An application to blockchain front-running. Technical report, National Bureau of Economic Research, 2022.

5 Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint*, 2022. `arXiv:2203.11520`.

6 John Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.