A Solomonic Solution to Blockchain Front-Running

Joshua S. Gans and Richard Holden*

December 2, 2022

Abstract

Blockchain front-running involves multiple agents, other than the legitimate agent, claiming a payment from performing a contract. It arises because of the public nature of blockchain transactions and potential network congestion. This paper notes that disputes over payments are similar to classic ownership disputes (such as King Solomon's dilemma). We propose a simultaneous report mechanism that eliminates blockchain front-running. In each case, the mechanism relies on threats to remove ownership from all claimants and preferences from the legitimate claimant over allocations to other agents.

Keywords: subgame perfect implementation, blockchain, front-running, mechanism design, ownership

* (Gans) Rotman School of Management, University of Toronto and NBER; (Holden) Department of Economics, UNSW Business School.

** All correspondence to joshua.gans@utoronto.ca. The latest version of this paper is available at `https://www.joshuagans.com/` and `https://github.com/solomonic-mechanism`. Thanks to Ethan Buchman, Scott Kominers, DJ Thornton and seminar participants at the Luohan Academy and New York University for useful discussions and Raphael Mu for excellent research assistance.

1 Introduction

Front-running has become a serious issue for smart contracts in blockchain ecosystems; threatening to completely undermine its potential.[1] The problem is straightforward. When a contract is placed on a blockchain such as Ethereum, there is a performance obligation on one party that, when achieved, triggers a payment in tokens from another party. Sometimes these contracts are open offers – such as a bounty or reward. When performance occurs, the intended payee sends a message to the payor that is akin to an invoice for payment together with evidence that the obligation was met. Being the blockchain, these messages are public prior to being committed to a block. Also, as there is potential congestion on the network, a message is sent with a delay depending upon the transaction fee nominated by the payee. In the intervening time, front-runners, or bots programmed to front-run, see the message and can resend it, substituting in their own address for payment and a higher transaction fee to achieve priority (Daian et al. (2019), Eskandari et al. (2019)). The payor's account for that contract is then drained of tokens before the intended payee can be paid.[2]

While such front-running is akin to the leap-frogging activities of high frequency traders,[3] in this case, it threatens to undermine the ability to offer smart contracts on any blockchain system.[4] Fearing non-payment, a contract payee may not perform or enter into a contract at all. This harms both parties and will likely stifle the development of smart contracts and the ensuing gains from trade. While some solutions involving encrypting messages have been posited, these can only potentially assist in some bilateral contracts between known and identifiable parties (Copeland (2021)) unless implemented at a platform level. Other solutions involving increasing the transparency of "front-running" races do not actually resolve the problem and merely place the payee on a more level playing field than front-running bots.[5]

[1]See Catalini and Gans (2020), Gans (2021) and Holden and Malani (2021) for overviews of the economic potential of the blockchain to solve contracting issues.

[2]The total value of tokens gained in this manner is estimated at almost \$1 billion since January 2020 (`https://explore.flashbots.net/`) although the vast majority of that is from arbitrage front-running rather than liquidition front-running which is the focus of this paper. See also, Ferreira Torres et al. (2021).

[3]This occurs were a large trade is placed and bots are able to trade in the market ahead of that trade and exploit arbitrage opportunities. This happens for cryptocurrencies on the blockchain as well using a technique called 'insertion' to front-run high value transactions; Ferreira Torres et al. (2021). However, this is not the type of front-running considered in this paper.

[4]See Robinson and Konstantopoulos (2020). The problem was first identified in 2014 in a Reddit post from pmcgoohan; see Stankovic (2021) for an overview. It is also possible that this activity could undermine the consensus layer of blockchains through front-running on past blocks using a time bandit attack; Daian et al. (2019).

[5]`https://ethresear.ch/t/flashbots-frontrunning-the-mev-crisis/8251` and the critique by Ed Felten `https://medium.com/offchainlabs/meva-what-is-it-good-for-de8a96c0e67c` Such auctions may also reduce the congestion effects generated by front-running; Buterin response.

In this paper, we provide and examine a mechanism designed to resolve ownership disputes that fall into a specific class; of which front-running of the type describe here is an example. Another famous example is the biblical dispute heard by King Solomon.[6] The class of disputes have the following characteristics:

1. The legitimate claimant is part of the set of agents making an ownership claim.
2. Legitimate and illegitimate claimants know if they are legitimate or not.
3. Legitimate and illegitimate claimants have different preferences over who, other than themselves, are allocated ownership.

In the case of Solomon's adjudication over who was the true mother of a baby, it was known that the true mother was one of the set of two claimants, each claimant knew their own status and, as we will discuss, it was a feature of the story that the true mother had different preferences than the other agent over what happened to the baby if ownership was not allocated to them. For blockchain front-running, the nature of the problem necessitates the legitimate claimant being part of the relevant claimant set, claimants knowing their own status and illegitimate claimants being indifferent was to other outcomes which may not be the case for the legitimate claimant.

The mechanism we deploy is a simple special case of the Simultaneous Report (SR) mechanism developed by Chen et al. (2018) that itself is a simplification of mechanisms explored by Moore and Repullo (1988) and Moore (1992).[7]

This paper proceeds as follows. In the next section, we set up the front-running problem and provides a mechanism (a Solomonic clause), implementable on blockchains, that resolves it entirely. Section 3 concludes.

2 Blockchain Front-Running

Our unit of analysis is a given contract. That contract comprises certain performance obligations whose performance can be verified by party a sending a message $M_a = \{\alpha, E\}$ where α is a's wallet address and E is verifiable evidence of performance to the network as a transaction. That transaction is then confirmed to a block and recorded on the blockchain. At that point, any payment, T, triggered by the receipt of M_a involves T in tokens being transferred to a's wallet. Note that any agent, i, sending a message that is confirmed to a block specifies and pays a transaction fee, $f_i > 0$.

[6]See Gans and Holden (2022) for more discussion.

[7]It is, however, distinct from the divided ownership processes examined by Ayres and Talley (1994) as it envisages a solution outcome whereby ownership is not divided.

Front-running arises when b observes M_a as it is broadcast to the network but before it is confirmed to a block and b chooses to send a message $M_b = \{\beta, E\}$ to the network. If M_b is confirmed to a block ahead of M_a, then T is automatically sent to b's wallet and a receives no payment. This could arise if M_b is confirmed to a block earlier than the block M_a is confirmed on or if it is confirmed to the same block with an earlier order amongst transactions in that block.

Given that M_a is broadcast first, how could M_b be recorded on the blockchain with an earlier time-stamp? Note, first, the messages are initially broadcast to a mempool. Those transactions are then validated and confirmed by miners or validating nodes who are responsible for ordering the transactions.[8] All valid transactions are recorded on the blockchain at which point the transaction with the earliest timestamp will trigger the contracted actions. Miners or validating nodes will then choose the order of transactions. On the Ethereum blockchain, miners will try and maximise transaction fee revenue by prioritizing transaction recording based on the transaction fee bids (or offered 'gas') that accompany a message. Thus, M_b can, by offering to pay a higher transaction fee, be ordered ahead of M_a in a block. Of note is the fact that, because miners have the power to order transactions, to the extent that transaction ordering matters, the ability to earn payments based on ordering power has been termed the *miner-extractable value* (or MEV) (see Daian et al. (2019)).[9]

It is useful to illustrate the severity of this issue for contracting. Suppose that a party contracts a to provide a service using a contract recorded on the blockchain. If a performs the contract, assume that it costs them, $c > 0$, to do so and T will be paid if evidence of performance is submitted. In the absence of front-running, a sends a message, M_a on the blockchain for a fee f_a that can be arbitrarily small and ends up with a payoff of $T - c - f_a$ which is assumed to be positive.

Suppose now that front-running can occur. If b sends a message M_b for a fee of f_b they can potentially earn T. If $f_a = f_b$, then b's expected payoff is $\frac{1}{2}T - f_b$ and a's falls to $\frac{1}{2}T - c - f_a$. In effect, the payment to a is taxed at 50 percent assuming there is only one front-runner. If there are more than one, the effective tax is higher.

This analysis, however, assumes that transaction fees are fixed. However, these are chosen by agents recording transactions on the blockchain. Typically, if $f_b > f_a$, b would be recorded at an earlier time-stamp and their payoff becomes $T - f_b$ while a's drops to $-c - f_a$. In reality, a and b choose their fees as part of a first-price, sealed bid, all-pay auction for priority. Here, b will choose a fee up to $f_b = T$ requiring a to exceed that to achieve priority; something

[8]Miners are responsible in proof-of-work protocols while validating nodes are responsible in proof-of-stake protocols.

[9]For a demonstration of a smart contracting being front-run in this manner see Scott Bigelow, "How To Get Front-Run on Ethereum mainnet", YouTube June 17, 2020.

that is not worthwhile. Having both post fees equal to T is not an equilibrium outcome if priority is then randomly assigned. Instead, one sets a fee at T while the other sets an arbitrarily low fee or does not choose to send a message. Given this, either a sets $f_a = T$ and earns a payoff of $-c$ or it sets a low or zero fee and earns the same payoff. Under these conditions, a chooses not to incur c and perform the contract regardless of how high T is.

Under these conditions, contracts that require settlement on the blockchain will not arise in equilibrium. Various solutions have been proposed to mitigate such issues. These have included auctions to make priority a more transparent process (Daian et al. (2019), Buterin (2021)). However, these auctions, do not prevent front-runners from participating and that competition still immiserizes contract safety as outlined above. A second solution involves adjusting blockchain protocols to improve time-stamping. However, as there are always lags of some kind achieving this is difficult. A third set of solutions involves encrypting messages until they are confirmed on the blockchain (Aune et al. (2018)). This can resolve this problem but it requires implementation at the blockchain protocol level, encrypting all messages which is computationally expensive, and moves away from the public nature of blockchain interactions.

Compared with contracting outside of the blockchain, the reason why such front-running is a threat is because there is no proof of identity required for payment. This is by design as the privacy of parties on the blockchain is a feature. Thus, blockchain smart-contract systems are characterised by contracts specify performance objectives but not the identity of those performing them. This allows anonymity in payments to be preserved. If the contract specified that following performance, payments would be made to a's wallet (α) specifically, front-running could not occur as M_b would not trigger a payment to β. However, anonymity means that the addressee for payment can be substituted without altering the contract. Our examination here is made on the basis that this blockchain feature needs to be preserved.

We make the following assumption with regard to agent preferences. The agent who has actually performed the contractual obligation and broadcasts a message of that performance earns T in utility if they receive payment for that performance, $-\theta$ if that payment goes to an illegitimate claimant and 0 otherwise.

2.1 The need to discretize time

In the literature on front-running in financial markets, one of the proposed solutions was to change time on an exchange from continuous to discrete time (Budish et al. (2015)). In order to operate a mechanism involving multiple agents, to resolve front-running on blockchains we must similarly discretize time. This is done by the smart contract proposing a time period counted from the time a first message M_i is recorded on the blockchain during which all such messages are collected and the mechanism we propose is run on them. The length of the time period, let's call it Δ, is a parameter that can be chosen.[10] Increasing Δ reduces the need for claimants to pay higher transaction fees so as to participate in the mechanism but also results in a delay in payment. If there is a single message received during Δ, there is no ownership dispute over the payment and the payment is made to the wallet addressee. If there is more than one message received, there is dispute and the mechanism we propose is run to resolve the dispute immediately upon Δ's end.

2.2 The single legitimate claimant case

While our mechanism applies for an arbitrary number of ownership claimants to T, initially we assume that there are two claimants, a (the legitimate claimant) and b (the would-be front runner or illegitimate claimant). Each claimant knows their own status but this is known to the mechanism designer. The designer's goal is that the payment only be made to the true claimant.

The process is initiated as soon as a claim is validated and confirmed on the blockchain. Consider the following mechanism:

1. If, in a time period, Δ, there is a single message, $\{i, E\}$, send T to i.
2. If, in a time period, Δ, there are two messages, $M_a = \{\alpha, E\}$ and $M_b = \{\beta, E\}$, the challenge stage begins. (Figure 1 illustrates the process by which claims are assembled on a blockchain).

The **challenge stage** involves:

1. One agent is chosen at random and given the opportunity to withdraw their claim.
2. If the claim is withdrawn, the other agent is paid T.
3. If the claim is asserted, T is paid to a third party (or, equivalently, the tokens burned) and the contract is nullified.

Thus, as is depicted in Figure 1, the legitimate claimant first broadcasts a message to the mempool where it can be seen by others triggering illegitimate claims. All claims pay the requisite fees and are confirmed to blocks in the specified time period, Δ. The mechanism is then run drawing from confirmed claims.

Given this, we can prove the following:

[10] Δ can be measured in time units or in blocks with the first block being the one where claim(s) are first confirmed.

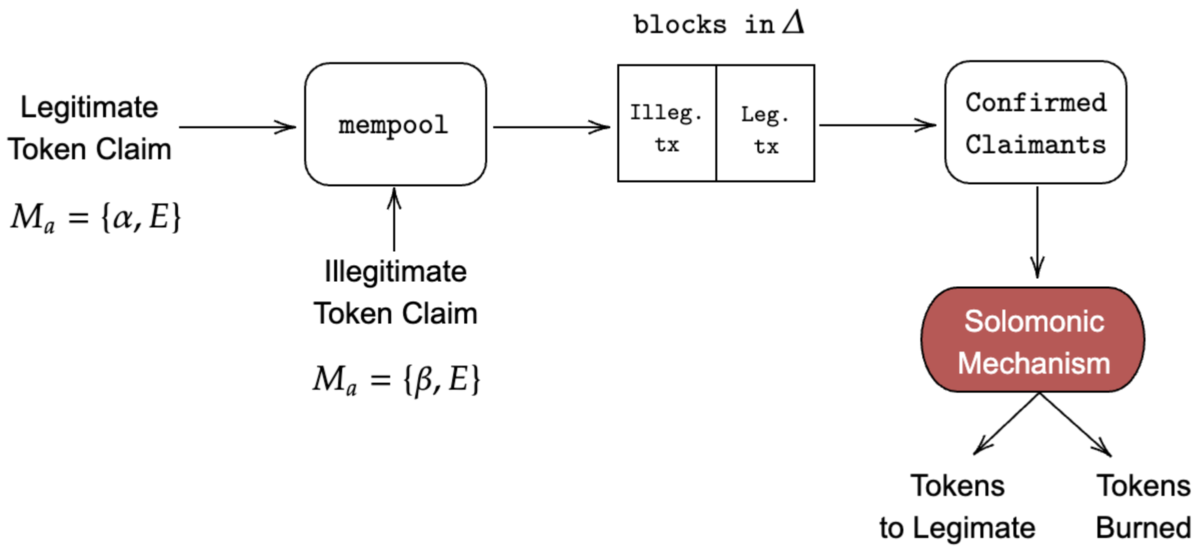


Figure 1: **Assembling Competing Claims**

Proposition 1 *Suppose that $\theta > 0$. The unique subgame perfect equilibrium involves a single claimant who is the agent who performs the obligation.*

Proof. Without loss in generality, suppose that a is the true claimant and the challenge state is initiated as b also makes a claim. Thus, both agents have incurred the transaction fee, f. There are two cases to consider:

1. If a is given the opportunity to renounce their claim, they will receive $-\theta$ if they renounce their claim (as they know the other claimant is illegitimate) and 0 otherwise. Thus, if $\theta > 0$, a will continue to assert their claim and T will be sent to a third party with each agent ultimately earning $-f$.

2. If b is given the opportunity to renounce their claim, they will receive 0 regardless (as they know the other agent is legitimate). Thus, they will be indifferent between renouncing or not and their ultimate payoff will be $-f$.

We now examine each agent's incentive to make a claim. As a moves prior to b, we work backwards by considering b choice. If b makes a claim (by front-running), they expect to earn $-f$ as a will never renounced their claim if $\theta > 0$. Thus, b will not make a claim. Given that b will not make a claim, a will make a claim and earn $T - f$. ■

Note that the mechanism requires that the true claimant have a strict preference regarding whether a payment is made to an illegitimate claimant. Otherwise, if there is a possibility that the legitimate claimant may renounce their claim, this opens up a potential return to illegitimate claimants. A weaker assumption that leads to this same outcome would be that if a true claimant is indifferent as to where the payment is made, if not to themselves, they choose to assert the claim. In equilibrium, if the true claimant were programming in their assertion response at the time they submit M_a, then it is optimal for them to assert their claim. Thus, the $\theta > 0$ assumption does not play role if agents precommit their mechanism responses as might arise in a Blockchain environment.

Remark 1 *The mechanism also yields a single true claimant if there are many potential illegitimate claimants. The only difference is that one claimant out of the pool of claimants is given the opportunity to renounce their claim in the challenge stage. Because the true claimant knows they are the true claimant, they will also choose to assert their claim if* $\theta > 0$.[11]

Remark 2 *There is a possibility that there could be multiple claimants with the illegitimate claimants forming a coalition. In this case, a randomly selected agent could chose to withdraw their claim but there still be multiple claimants. In this case, what would happen to the tokens? One way of overcoming this is, when there are more than 2 claimants, a set of agents are randomly selected. If any assert their claims, T is paid to a third party. If all withdraw their claims, those agents are removed from the pool of claimants and a new set of agents (half or just under one half of the remainder) are randomly selected and the mechanism is repeated. Eventually, an agent who is asserting their claim will be selected and T will be paid to a third party. There is guaranteed to be one such agent as the true claimant is amongst the starting pool.*[12]

Remark 3 *What if, due to network issues, the true claimant is not amongst the pool of claimants when the mechanism begins? If illegitimate claimants believe that this is a possibility, they have an incentive to make such claims. Clearly, if they are the only claimant, they will be able to capture T. If they are amongst multiple claimants, this is not possible unless, those multiple claimants are controlled by them. Thus, there would have to be some collusive mechanism amongst illegitimate claimants to subvert the mechanism in this way. So long as the probability that the legitimate claimant is not in the relevant pool at the time the mechanism is run is low enough, the deterrence effect of the mechanism remains.*

[11] One possible front-running strategy would be for a front-runner to send multiple messages for the same wallet address. To avoid this, a claim to be resolved would draw based on messages. Of course, front-runners could send messages for different wallets. This, however, would not exclude the true claimant and so would ultimately fail.

[12] If there were concerns that this process may take time, then a cost could be imposed on each round of participation.

Remark 4 *Since Aghion et al. (2012) it has been understood that certain mechanisms may not be robust to small perturbations from common knowledge. Chen et al. (2018) show how suitably-designed lotteries can ensure that the mechanism used in this paper is robust to private-value perturbations. It would be straightforward to do so in this environment if desired, although it would make the mechanism slightly more involved.*

Remark 5 *We could have specified a form of "mutually assured destruction" by automatically burning the tokens if there is more than one claimant. This is a reasonable approach, but our challenge stage permits the construction of lotteries mentioned in Remark 4 that make it the unique optimal choice for an illegitimate claimant to withdraw their claim (rather than being indifferent). A practical feature of our mechanism is that an illegitimate claimant must pay an additional transaction fee to send the message at the challenge stage, as this must be written to the blockchain. Such a fee also breaks their indifference, while the true claimant has preferences that make them willing to pay a certain fee to assert at the challenge stage and ensure that an illegitimate claimant does not receive the tokens.*

2.3 The multiple legitimate claimant case

The above analysis envisages a contract on the blockchain where there is only one agent who can be the legitimate claimant. However, in some applications – say involving bounties or rewards for performance – can have multiple legitimate claimants. In the absence of front-runners, such rewards would be made based on some time verification of the messages from agents. That may result in multiple claimants but the contract could specify a tie-breaking rule or another measure to award the bounty including splitting it. When there are illegitimate claimants, however, those rules would create an incentive to front-run the contract.

A potential solution in this case would be to run the mechanism as proposed for the one legitimate claimant case. This might be done by shortening the time (Δ) where claims will be evaluated even if this results in potentially higher transaction fees. Reducing Δ means that any true claimant will more likely to believe that no other legitimate claimant will submit a competing claim during that period and there will be one true claimant. In that case, the fact that front-runners do not have an incentive to claim, will preserve the contractual incentives. The cost is that this will limit the tie-breaking options that might otherwise be used in such contracts. Such options are important if they play a role in providing incentives to compete and perform the contract obligations.

There is, however, a counter-risk that arises in this particular case: the payor may have an incentive to front-run their own mechanism. This would arise if it could not be guaranteed that T was being sent to a party other than the payor. Thus, the mechanism would have to specify that the tokens be burnt or sent to a legitimate charity account that is publicly verified.

These potential issues, however, need to be weighed against the real possibility that the contract would be otherwise completely unworkable if front-running was possible. Thus, the use of the mechanism expands the feasible contract space but does not obtain the full range of options that would be available if front-running were not possible at all.

2.4 A note on implementation

The mechanism we analyse here can be easily implemented on existing blockchains. Indeed, we have already provided code for a generic smart contract on the Ethereum network.[13] In effect, it is a Solomonic clause added to existing smart contracts. Figure 2 shows a flow chart of its operation.

There are many open design choices in implementing Solomonic clauses that we list here but that their resolution is beyond the scope of the present paper. These include:

- *Hard-coded challenge response*: the mechanism as outlined includes a claim message followed by a message in the challenge stage if a potential claimant is selected. However, it could be envisaged that the initial message contains the contingent response in the challenge stage rather than requiring a separate message and fee payment.

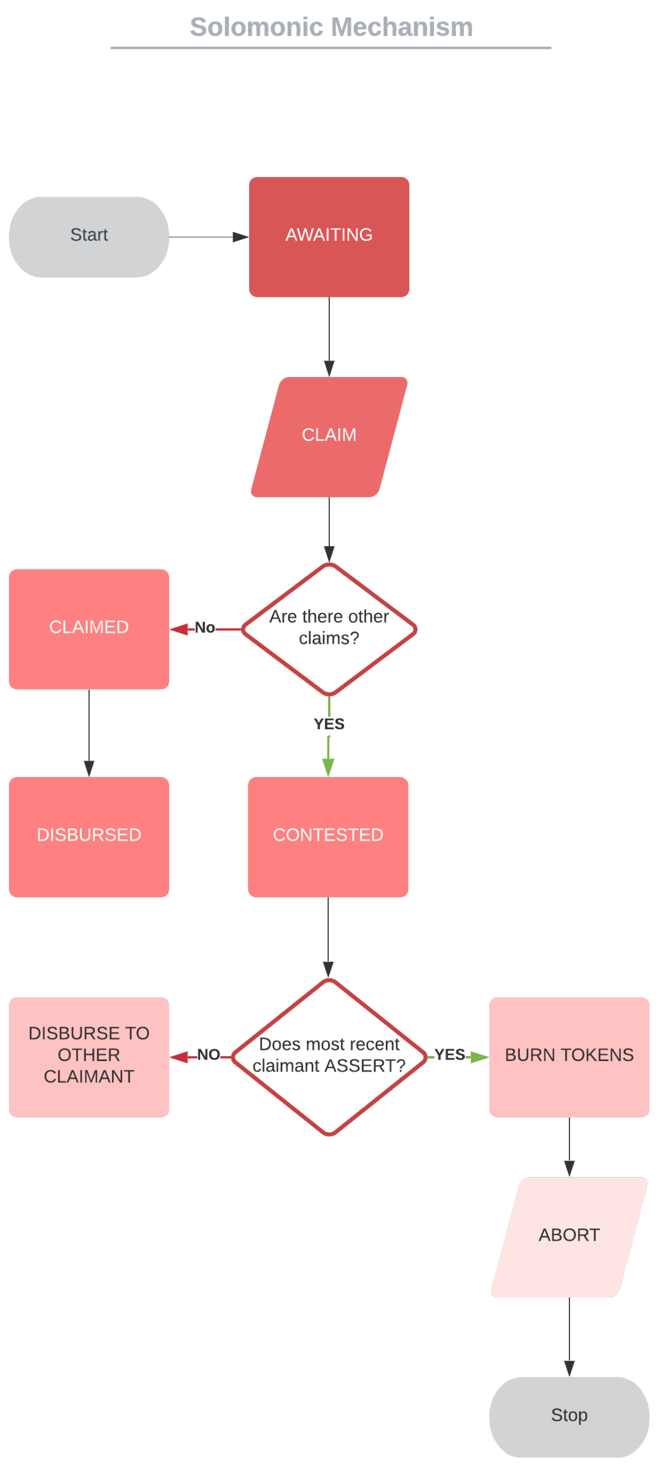
- *Randomization*: implementing randomization on a blockchain virtual machine can be challenging and often requires a call to an Oracle that is off-chain. In our implementation example, the agent chosen in the challenge stage was not chosen at random but was the agent with the most recent time-stamp on recorded on the blockchain. Theoretically, this the agent most likely to be the true claimant as they would not be putting forward higher transaction fees as part of a front-running strategy. However, due to latency on the internet, that agent is, in part, determined randomly and thus, this would be a useful alternative to pure randomization.

- *Time period*: in our implementation we set the time period, Δ to 60 seconds (or 4 blocks on Ethereum). The actual time period chosen would depend on other factors including network congestion and the need to clear token payments quickly or not.

- *Token burning*: If a claim is asserted in a challenge stage, then the tokens need to be transferred away from any party in the arrangement for the mechanism to work.

[13]See the repository at `https://github.com/solomonic-mechanism`.

Figure 2: **Flowchart of Solomonic Clause**



This could involve burning the tokens (sending them to a null address) or, alternatively, having the tokens become part of a fund or non-profit. As the mechanism, if successful, should involve little of this in equilibrium, where the tokens are sent is a decision that should be made to ensure that the mechanism is not attacked by malicious agents trying to force the tokens to be burnt or otherwise undermine the operation of the smart contract.

- *Signaling*: A contract with a Solomonic clause could involve a message for payment indistinguishable from contracts without that clause or one that indicated the existence of the Solomonic clause. The distinction would impact on front-running and its attempts. When there is a clear signal, front-runners would avoid these contracts. When there is no clear signal, they may not unless there was a sufficient share of contracts with a Solomonic clause in which case, front-running on all contracts may not be worthwhile. The use of such signals is, therefore, an important implementation choice.

3 Conclusion

We have outlined a mechanism which addresses front-running of smart contracts. An advantage of our approach is that the mechanism is embedded in a given smart contract, rather than needing to be deployed on the blockchain itself. The code we have provided shows how implement the mechanism in a smart contract, and we have trialed such contracts on the Ethereum blockchain (see `https://github.com/solomonic-mechanism` for details.) By removing a major impediment to smart contracting, we hope that such contracts will be able to achieve their potential, including the ability to write renegotiation-proof contracts that are not possible in traditional contracting environments.

References

Aghion, P., Fudenberg, D., Holden, R., Kunimoto, T., and Tercieux, O. (2012). Subgame-perfect implementation under information perturbations. *The Quarterly Journal of Economics*, 127(4):1843–1881.

Aune, R. T., Krellenstein, A., O'Hara, M., and Slama, O. (2018). Footprints on a blockchain: Trading and information leakage in distributed ledgers.

Ayres, I. and Talley, E. (1994). Solomonic bargaining: Dividing a legal entitlement to facilitate coasean trade. *Yale LJ*, 104:1027.

Budish, E., Cramton, P., and Shim, J. (2015). The high-frequency trading arms race: Frequent batch auctions as a market design response. *The Quarterly Journal of Economics*, 130(4):1547–1621.

Buterin, V. (2021). Proposer/block builder separation-friendly fee market designs.

Catalini, C. and Gans, J. S. (2020). Some simple economics of the blockchain. *Communications of the ACM*, 63(7):80–90.

Chen, Y.-C., Holden, R., Kunimoto, T., Sun, Y., and Wilkening, T. (2018). Getting dynamic implementation to work. *Unpublished manuscript*.

Copeland, T. (2021). A dex on cosmos is working on a way to prevent front running. *The Block*.

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., and Juels, A. (2019). Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*.

Eskandari, S., Moosavi, M., and Clark, J. (2019). Sok: Transparent dishonesty: front-running attacks on blockchain.

Ferreira Torres, C., Camino, R., et al. (2021). Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *USENIX Security Symposium, Virtual 11-13 August 2021*.

Gans, J. S. (2021). The fine print in smart contracts. In M. Corrales Compagnucci, Fenwick, M. and Wrbka, S., editors, *Smart Contracts Technological, Business and Legal Perspectives*, chapter 2. Hart Publishing.

Gans, J. S. and Holden, R. T. (2022). A solomonic solution to ownership disputes: Theory and applications. Technical report.

Holden, R. and Malani, A. (2021). *Can Blockchain Solve the Hold-up Problem in Contracts?* Elements in Law, Economics and Politics. Cambridge University Press.

Moore, J. (1992). Implementation, contracts, and renegotiation in environments with complete information. *Advances in economic theory*, 1:182–282.

Moore, J. and Repullo, R. (1988). Subgame perfect implementation. *Econometrica*, pages 1191–1220.

Robinson, D. and Konstantopoulos, G. (2020). Ethereum is a dark forest. *Medium, Aug.*

Stankovic, S. (2021). What is mev? ethereum's invisible tax explained. *Cryptobriefing.*