In https://eprint.iacr.org/2022/898, a protocol to achieve mempool privacy through threshold encryption is proposed. In private mempool settings, block proposers can see the content of transactions only after the order of transactions is already fixed. The goal is to prevent some forms of MEV this way, such as front-running and sandwiching.

The protocol in a nutshell: users encrypt their transactions to a (threshold) public key, and the block proposer creates a block with the encrypted transactions. As soon as 2/3s of validators voted on that block, transaction can be decrypted and executed.

Below are a few questions that I would love to discuss, regarding mempool privacy using threshold encryption in general and the proposed protocol specifically.

Validator collusion

What are the incentives for validators not to collude, given the profitability and deniability? If the colluders went for every front-running and sandwiching opportunity, it would be obvious that the threshold protocol does not fulfill its intended purpose. But the colluders could still go for many front-running opportunities and for the most profitable sandwiches while remaining undetected.

Preventing spam transactions

If transactions could be submitted for free, an attacker could practically run a DoS attack against the blockchain. In order to prevent such attack, the sender, transaction fees, and gas limits in transactions remain public to ensure the payment of transaction fees. How much information does that leak, and is the leaked information sufficient to opportunistically extract value? Furthermore, inefficiency is increased, because users have to pay for the gas limit and not for gas used. Will more transactions revert because of too tightly chosen gas limits?

Instead of leaving some information unencrypted, could a proof be constructed that the ciphertext contains a transaction that will pay reasonable transaction fee and that the sender indeed has the required funds to pay for the transaction fee? It would furthermore be necessary to ensure that a proof cannot be reused. Is that possible somehow?

Arbitrage and backrunning opportunities

One validator is selected as proposer for a particular block and creates that block with encrypted transactions from the mempool. For that block, each validator computes its decryption share for each tx and includes all the decryption shares in its vote for that block. Transactions remain private until ≥ 2/3 of the decryption shares are published during the voting phase.

At some point during the voting phase 2/3 - 1 decryption shares are published. Now any of the remaining validators, who has not published their decryption shares yet, can decrypt the transactions in the block locally, without having to publish its decryption shares. In this way, the remaining validators can get a head start in exploiting arbitrage and backrunning opportunities. Is it a desirable property that validators get privileged access to arbitrage and backrunning opportunities, given that it may act as a centralizing force (exploiting opportunities → higher profit → larger stake as a validator)? Wouldn’t that potentially destabilize the voting protocol, because incentives are misaligned - the protocol expects validators to publish their decryption shares but exploiting arbitrage opportunities tells them not to? Could an MPC protocol with guaranteed fairness be used instead as part of the voting protocol?

Group membership

The set of validators is determined by staking. But having a stake does not necessarily guarantee successful participation in the distributed key generation protocol - those who hold secret key shares may not want new (staking) members to join the distributed key generation protocol. How can it be ensured that the two sets, validators who stake and those who hold secret key shares, do not diverge over time?

Low cost attack on storage

In order to reduce the required on-chain storage, decryption shares do not necessarily have to be stored with each transaction. Instead, decryption shares can be discarded, if a transaction was decrypted successfully, and only a single symmetric key is stored on chain. If a transaction can not be decrypted, however, all decryption shares need to be stored with that transaction - and the sender may only pay some minimal transaction fee. If that is indeed the case, it would present a low cost attack on the disk space required for full nodes.

Size of the validator set

The protocol seems to work only for relatively small validator sets (<200) due to communication overhead. How could that be improved to support larger validator sets such as the one in Ethereum?