TEEs on dYdX Chain

Author: Max Holloway

Introduction

The dYdX chain is the first of its kind in the way it brings perps fully on-chain. While dYdX V3 and other perps dex counterparts enable non-custodial perps trading, most of them still utilize a single central party to match orders. On dYdX V4, the matching engine is run by each validator, and hence is fully decentralized. This makes dYdX V4 the most decentralized perps dex. However, dYdX V4's full decentralization comes with a catch: it is now possible for dYdX chain to leak maximal extractable value (MEV) to chain validators.

In an effort to decrease the MEV available on the dYdX chain, we investigated how the chain could leverage trusted execution environments (TEEs). TEEs are a hardware technology on modern day CPUs that make it possible for one party to verify the software used by another party. In the case of dYdX chain, we hypothesized that validators could attest to the exact software they were running, and users could encrypt their transactions such that the transaction could only be included in a block proposed by a validator running attested-for software. The goal of this would be for users to be able to use the chain knowing that validators will not extract value at the expense of the user's orders.

In this post, we begin with an overview of TEEs. We'll then take a look at two types of MEV on the dYdX chain: that which uses order information, and that which doesn't use order information. We show how TEE-enabled dYdX chain could (mostly) prevent MEV that uses order information, and we show that TEEs are not

capable of solving MEV that is agnostic to order information. Finally, we review the benefits and drawbacks of implementing a TEE-enabled dYdX chain, and we conclude with a recommendation; TEEs do not eliminate MEV, and we do not recommend that dYdX chain utilize TEEs.

TEEs overview

Trusted execution environments are a hardware technology with two important properties: (1) the hardware uses encrypted memory to keep code and data in the TEE private, and (2) a software integrity mechanism through which an entity running software on a TEE can provide a proof that the software matches a known software distribution. These two properties can be combined such that people can run workloads on an untrusted third party's machine, while getting a guarantee that the third party is running the correct software (not a malicious modification of the software) and that the third party does not have access to the data manipulated by the software.

How are TEEs used?

TEEs are not a blockchain-specific technology. In fact, we can build intuition on the TEE offering via an example unrelated to blockchains whatsoever: DNA sequencing. DNA sequencing utilizes computationally demanding algorithms on gene data to determine the order of nucleotides (ref). Suppose a biotech researcher wants to run sequencing algorithms on gene data from a human, but the researcher's personal computer does not have resources to run the program. Furthermore, the researcher has a friend with a supercomputer, but that friend should not have the ability to view the sensitive gene data. Under this scenario, if the researcher wants to ensure that their friend cannot access the gene data, they could put the sequencing algorithms into a TEE on the supercomputer, send the encrypted sequencing data to the supercomputer, let the supercomputer run the workload, then send the encrypted results of the workload back to the researcher. In this example, the researcher was able to utilize a third party's computational resources, while knowing that the third party was running trusted software, and that the third party has no access to unencrypted program inputs or outputs.

Although malicious DNA sequencing is obviously not a concern of dYdX, malicious block building certainly is a concern. If validators were constructing blocks from within a TEE, then users of the dYdX chain would be able to verify the software running that is used to construct blocks, and they could pass in encrypted transactions.

Drawbacks

Of course, no technology is perfect, and the devil is in the details when it comes to TEEs. Most important of all of the TEE drawbacks is security: TEEs are subject to a host of attack vectors, including side-channel attacks (using a TEE in such a way that leaks information from the software running on the TEE) and hardware attacks (finding issues in the implementation of the TEE hardware that allow leaking TEE private information). These attacks are far from rare — particularly on the most common TEE, Intel SGX — and a sufficiently motivated security researcher can exploit TEEs if they have access to the hardware. In the case of dYdX TEEs, this means that one needs to either trust hardware cloud providers like AWS or Azure, or one needs to trust a person self-hosting their hardware. The former is not a great trust assumption, and the latter is an unacceptably bad trust assumption. So TEEs would not play nicely with validators self-hosting.

Furthermore, TEEs have drawbacks on what software they can run. For instance, Intel's SGX is mainly meant for running single-process userspace applications. If you want to run multi-process applications, then they need to each run in their own TEE and have developer-implemented encrypted communication between them. TEEs are also generally more difficult to develop on top of, since developing on a TEE requires utilizing specific hardware SDKs that differ between machines. Since this technology is hardware specific, developers cannot use multi-platform hardware virtualization development environments like dockerized dev environments.

TEEs are still a relatively new technology, and utilizing them would come with a number of warts for both developers and validators alike.

Two classes of MEV on dYdX Chain

For the purposes of this post, we will loosely define dYdX Chain MEV as value that a block proposer can extract via their ability to reorder, censor, and insert their own transactions into a block. There is a great research paper, written by validator firm Chorus One, which goes into detail on the many ways that validators can extract MEV on dYdX chain (ref). Importantly, we can bifurcate dYdX Chain MEV into two categories: MEV that requires transaction information, and MEV that does not require transaction information.

An example of MEV that requires transaction information is Uniswap-style front-running: the block proposer has a set of transactions, they use the plaintext of one of those transactions and see that the user will buy BTC at a price of $100,000, but the current price is $60,000, and the block proposer matches their own BTC buy order in front of the $100,000 limit price order. Note that the block proposer only knows to place their own front-running order if they see the victim's BTC buy order in the mempool. If the victim's order was encrypted or otherwise not visible to the block proposer at the time of block construction, this attack would not be possible.

On the other hand, there is another type of MEV that does not require access to transaction information. One example of this is blanket transaction censorship: block proposer censors all incoming transactions for most of their block proposal period, and they produce a block that only contains their own orders. In this case, the block proposer gets the option to taker against any cryptographically-valid maker orders, even if those makers broadcasted cancellation txs well before the end of the block construction period. In this example, if the fair value of BTC perps increases 10bps during the block construction interval, the BTC market's best-bid-offer is 0.2bps wide around the previous fair value, and the block proposer has 2.5bp taker fee tier, then the block proposer could send taker buy orders and trade at about 7.4bps discount from fair value (10bps - taker fee - half spread). Importantly, the block proposer does not need to use any information from transactions or the gossip network; they simply need to censor all other transactions.

How to eliminate MEV that uses order information

TEEs can be used to eliminate MEV that uses order information on dYdX chain. This would entail validators running the chain from within a TEE, then giving users the option to encrypt their transactions against the validators' TEE, ensuring that an order can only be decrypted and matched from within the TEE. Under this paradigm, validators have the choice to become a TEE validator, and users have the choice to encrypt their transactions such that only TEE validators can build blocks with the transaction. We'll start with an example encrypted transaction lifetime, then show the modifications that would be made to the validator, then finally demonstrate how this eliminates order information-based MEV.

[

trusted-untrusted

1436×740 67.7 KB

](https://europe1.discourse-cdn.com/flex013/uploads/dydx/original/2X/9/92c3d41d1c0abbdf524c857f9adfe2bdae01f1e6.png)

Users can verify the dYdX Chain protocol client running from within the TEE. Other hardware and software components, such as the network interface controller or the host operating system, are not within the TEE and thus are not trusted.

Lifetime of an Encrypted Transaction

1. User encrypts their transaction against a TEE public key. Users broadcasts the transaction into the network.
2. The TEE validator begins block construction from within the TEE.
3. The iterates through each mempool transaction; for any given encrypted transaction, it will attempt to decrypt it using the TEE private key. Once this process is completed, the TEE validator has a set of decrypted transactions that can be used to construct a block.
4. The TEE validator constructs a block using the existing block construction logic: first-come first-serve order cancellation and matching. This results in a block that it could propose to the network.
5. The TEE validator then signs the block's DataHash and the block's Height, using the TEE private key. This signature proves that the block was constructed from within the TEE.
6. The TEE validator broadcasts the block, other validators verify the block contents and the TEE signature. If the block's transactions are valid, and the block's TEE signature matches the block height and transactions, then the validators

consider the block valid.

Validator Modifications

TEE validators would initially register their validator nodes like any other validator node in the network. If the validator decided to run their validator client from within a TEE, then the validator client would generate a TEE-specific private key that is only accessible to software running in the TEE. That is, the entity running the validator would not have access to the TEE-specific private key, but the dYdX chain node running in the TEE could access to the TEE-specific private key.

At any point, the entity running the dYdX chain node in a TEE could create an InitializeTeeValidator

transaction, signed by the validator signing key. This transaction would contain sufficient evidence that the TEE is running the given validator node. This can be done by the validator node signing the full node's consensus node_id, an attestation of the software running in the TEE, and a good_til_block that specifies an expiration for the signature.

// Example data in an InitializeTeeValidator transaction { "validator_address": "4548C96CA12DF7BAEA97A4F2AB23E428116BB990", "tee_public_key": { "type": "tendermint/PubKeyEd25519", "value": "6TrVN9yyKAymZqw0kkO8e4ZvUhnP+zVnTbJWoOHRnAM=" }, "tee_attestation_hash": "", "good_til_block": 1000, "tee_signature": "iShEhZR8uyI+Jj1z8aZdRb9fVkxQ0PIzmxn9wjsCJxggcJtkHVGuSRwiVpzM1qn0vwL0dIzO0u5bMwhnBfsQCw==" }

All validators (both TEE validators and non-TEE validators) would receive and verify the InitializeTeeValidator

transaction. Assuming the TEE signature was valid, then all validators will update a chain state variable to associate the tee_public_key

with the given validator_address

.

Following a successful InitializeTeeValidator

transaction on block N

, all blocks proposed by the TEE validator after block N

must include an accompanying TEE signature in block metadata. This TEE signature would be generated by the TEE private key signing hash([DataHash, Height])

(ref), which would guarantee that the block was constructed from within the TEE, and hence the entity running the TEE validator did not have any of the encrypted transactions' information before signing the block's data.

Example of eliminated MEV: front-running

Suppose that a TEE validator wanted to conduct front-run a user's order to give the user their limit price. In order to do so, the validator would need to know the limit price on the user's transaction. If the user encrypts their transaction against the TEE validator's TEE private key, then that limit price is not available until the TEE private key decrypts it. The validator could attempt to get the user's limit price in two ways.

1. Exploit the TEE to extract the private key, then use the private key to get the limit price.
2. This is feasible, but very difficult.
3. This is feasible, but very difficult.
4. Decrypt the transaction from within the TEE, let the TEE construct the block, view the block transactions in plaintext, and refuse to broadcast the block.
5. While this is possible, this would lead to a missed block proposal, which the network can easily monitor and deter via in-protocol slashing.
6. While this is possible, this would lead to a missed block proposal, which the network can easily monitor and deter via in-protocol slashing.

Since the validator can only retrieve a user order's limit price from these two methods, it is unlikely that users' encrypted orders will have a leaked limit price. This means that Uniswap-style front-running on dYdX — where validators use the order's information to front-run or sandwich a user's order — would be eliminated for encrypted transactions.

Do TEEs eliminate MEV that doesn't use order information?

We have reviewed how dYdX chain can use TEEs to eliminate MEV that uses order information; but can TEEs eliminate

MEV that doesn’t use order information? In short, the answer is a clear No

.

Fundamentally, TEEs provide the guarantee that a server will run a client-verified piece of software. However, TEEs cannot guarantee what data that software observes. A malicious entity can run a proxy between the network and the TEE validator client without being detected.

Once this undetectable proxy is in place, it can censor and reorder transactions. And while it cannot decrypt transactions that were encrypted against TEE validator — and thus cannot garner access to encrypted transactions’ order data — it can still position a TEE validator operator to generate revenue via censorship MEV.

For example, consider the case we considered before, in which the fair value of BTC perps jumps by 10bps during block construction time. Suppose that the TEE validator operator has a proxy in place, and that proxy only broadcasts transactions to the TEE validator near the end of block construction time. Furthermore, suppose there is logic in the proxy that states “if BTC price has changed by ≥ 10bps since the beginning of the block construction time, then censor all transactions other than our own”. Under this logic, the TEE validator would only be exposed to the TEE validator operator’s transactions, which themselves would generate revenue for the TEE validator operator at the expense of market makers and other traders on dYdX chain.

Even simpler, a validator could give themselves latency advantages with zero code changes whatsoever by configuring a full node in New Zealand, then have their validator run in Tokyo and only peer with the full node, then make an RPC connection from their trading system directly to the validator. In this case, their trading system’s orders would get at least 30ms latency advantage over the competitors’ orders that are coming from New Zealand.

[

proxy-problem

1440×718 64.4 KB

](https://europe1.discourse-cdn.com/flex013/uploads/dydx/original/2X/5/50f19ce4b6809662ae7474a02da3ffeebe5a4cb7.png)

Example of censorship-based MEV that requires no code changes. This is just as much of an issue if the proposer is a TEE or non-TEE validator.

Since proxies can deter or entirely censor orders without the proposer client being able to tell the difference, this form of MEV cannot be solved with TEEs. Given that this problem is so meaningful, it is clear that TEEs are far from a complete solution.

Areas for future research

In this post, we look into a specific problem and solution space: how can we lower the threat of MEV on dYdX Chain via TEEs. This leads us to a partial solution, which can eliminate front-running orders, but which does not strip block proposers from their extracting value. But TEE-enabled validators is not the only way to approach MEV on dYdX Chain. There are a number of areas where future research may help dYdX make progress toward eliminating the specter of MEV.

One future research area is to introduce a private, centralized sequencer that produces private blocks. This sequencer would run from within a TEE, so users could generally be confident that the sequencer will not be able to front-run their transactions. Of course, this approach still does not solve the censorship-resistance problems that plague our other solution, but a censorship resistance argument is more compelling for a single centralized sequencer than for a group of validators whose reputation is less important. Nevertheless, moving to a centralized sequencer model would be meaningfully more centralized than the system that exists today, and such a centralizing change does not seem worth it.

Yet another area of research is to modify the application’s matching itself to make MEV more difficult. For instance, the dYdX Trading team’s frequent batch auction article details how to modify the matching engine to guarantee a uniform price for all trades in a single block. This would make sandwiching-style attacks impossible, but it would still not solve for the censorship case we detail above. Further research in this area may lead to practical MEV mitigations.

Conclusion

TEEs can greatly enhance dYdX’s user order privacy guarantees. Sandwich-like attacks would become impractically difficult in a TEE world, and this would provide a notable improvement over the current dYdX MEV solutions. However, TEEs are not a complete solution to MEV on the dYdX Chain. Most notably, they do not prevent censorship attacks, which would remain a problem for as long as dYdX exists. And it doesn’t help that TEEs have other issues, such as vulnerabilities to sophisticated, malicious validators, and increased development and integration effort for bringing TEEs to dYdX Chain.

With this in mind, I do not recommend that dYdX Chain incorporate TEEs into their validator clients at the moment. However, this technology does offer substantial benefits, so we should keep this solution in mind over time as the chain evolves.

Acknowledgments

My gracious thanks to the following individuals for their help with this research: Marko Baricevic, Matthew Downey, Frieder Erdmann, and Elijah Fox.