Many thanks to the PSE team & the zk residency group who gave me the reviews!

TL;DR

- Schnorr Sequencer is an optimistic approach for faster commitment in Ethereum scaling solutions
- Allows transaction-level commitments and instant finality
- Sequencer and sender complete a Schnorr signature for specific transaction index
- Involves sharing nonces, partial signatures, and challenge values between user and sequencer
- Suitable for layer 2 sequencers and transaction bundling in Account Abstraction
- Offers versatile solution for transaction ordering and commitment in Ethereum
- Economical solution for low-throughput systems such as app-chain
- To mitigate MEV attacks, timelock cryptography can be used
- Potential extends to rollups with multiple nodes connected through peer-to-peer network will help synchronization of the state instantly

Introduction

The zero-knowledge proof and the optimistic approach are both crucial in scaling solutions for Ethereum. ZKP, in particular, can provide trustless correctness of execution results, making it ideal for achieving finality of L2s. However, computing ZKP for a general-purpose rollup can be time-consuming. To address this issue, we can combine ZKP with the optimistic approach and leverage game theory for a faster commitment scheme.

The Schnorr Sequencer model is a commitment scheme for transaction ordering that enables the transaction-level commitment model rather than commiting a batch of transactions. This model allows the sequencer and the sender to complete a Schnorr signature for a specific transaction index. With the completed Schnorr signature, the sequencer commits to including the transaction for that index.

In detail, the sender opens a communication channel with the sequencer, and the sequencer returns an index for the transaction. The sender creates a partial signature, and the sequencer completes the Schnorr signature and publishes the information. Later, the sequencer submits a batch or bundle of transactions. If the transaction is not included in the batch with the given index, the sequencer gets slashed. This means that we can achieve transaction-level instant finality if the sequencer provides the signature data.

How the Schnorr Signature works

We'll start with the simple schnorr signature scheme. Basically, schnorr signature is multiplying a private key and a message hash value and verifying them using the homomorphic hidings and public keys. And to prevent it's get reversed, we add a random nonce value there. Let's see how it works in more detail.

1. Private key of Alice: k

, her public key: P

1. The message to sign: m
2. Alice chooses a random nonce r

, and its public value R

1. Challenge value e = H(R || P || m)
2. Creates a signature s = r + ke

and share (s,R)

1. Verification:

sG == R + e(kG) = R + eP

The advantageous of the Schnorr signature is that we can aggregate them using its linearity but the risk is the key-cancellation attack. So the MuSig[^1] scheme introduces us multiplying weight factors to the each private keys to prevent the key-cancellation attack. Let's see how the MuSig scheme works to aggregate signatures for two parties.

Alice

Shared

Bob

privKey

k_a

k_b

pubKey

P_a = k_a G

P_b = k_b G

random nonce

r_a

r_b

public nonce

R_a = r_a G

R_b = r_b G

aggretaged public nonce

R = R_a + R_b

hash of the public key set

l = H(P_a \| P_b)

weight factor

w_a = H(l\|P_a)

w_b = H(l\|P_b)

aggregated pub key

X = w_a P_a + w_b P_b

challenge e

e=H(R\|X\|m)

signature

s_a = r_a + k_a w_a e

s_b = r_b + k_b w_b e

aggregated signature

(s,R), s = s_a+s_b

And the verification should be:

sG = (s_a + s_b)G = (r_a + r_b)G + (w_a k_aG + w_b k_bG)e = R + eX

How the Schnorr Sequencer works

For the next step, let's try to make the sequencer and the user sign a schnorr signature together to put that transaction into a given index.

1. Sequencer registers the public key P_s

and stake some amount of ETH.

Sequencer’s private key: k_s

Sequencer’s public key: P_s = k_s G

\text{sequencer} \xrightarrow{P_s, \text{proof of stake}} \text{smart contract}

1. User chooses a random nonce and builds a transaction. Then sends the hash value of the transaction m

, user’s public key P_u

and the public nonce R_u = r_u G

to the sequencer.

Transaction hash: m = H(\text{tx})

User’s random nonce: r_u

User’s public nonce: R_u = r_u G

User’s private key: k_u

User’s public key: P_u = k_u G

\text{user}\xrightarrow{R_u, P_u, m}\text{sequencer}

1. And the sequencer secures an index to put the transaction and pick a random nonce r_s

to share the challenge value e

which is

e=H(R\|X\|m\|i)

where R = R_s + R_u

and X

is the weighted public key by the MuSig scheme.

Finally returns the index i

and the public nonce R_s

to ther user

\text{user}\xleftarrow{R_s, P_s, i}\text{sequencer}

1. Now the user and the sequencer share the challenge value e

and user starts to create the partial signature

s_u = r_u + k_u w_u e

and send it to the sequencer with the tx data together

\text{user}\xrightarrow{(s_u, \text{tx})}\text{sequencer}

1. Sequencer verify the signature and the hash of the transaction by checking

s_u G == r_uG + w_uk_uGe = R_u + w_uP_ue

m == H(\text{tx})

if the signature valid, then returns the partial signature data s_s

\text{user}\xleftarrow{(s_s)}\text{sequencer}

1. Now both can complete the full signature (s,R)

by adding both (s_s+s_u, R_s+R_u)

and publish this on the data layer. Or both of them can execute this transaction on-chain.

Validation

- Validator, a.k.a the smart contract, should have a mapping of the user's address and the public key
- Verification steps
- Compute X

using the registered public key of the tx.origin user and the sequencers's public key.

1. Compute the challenge e = H(R\|X\|m\|i)

using R

, X

, transaction hash m

, and the transaction index i

.

1. Verify sG == R + eX
2. Compute X

using the registered public key of the tx.origin user and the sequencers's public key.

- Compute the challenge e = H(R\|X\|m\|i)

using R

, X

, transaction hash m

, and the transaction index i

.

- Verify sG == R + eX

Challenge Steps

In the Schnorr Sequencer model, the challenge arises when the sequencer does not include the committed transaction in the batch with the given index. In this case, the user can provide proof of the sequencer's misbehavior to the smart contract. The smart contract will then verify the proof and slash the sequencer's stake.

1. User observes that the transaction is not included in the submitted batch with the given index.
2. User provides the user's public key P_u

, full signature (s, R)

, user's partial signature (s_u, R_u)

, index i

, and the transaction data to the smart contract.

1. The smart contract retrieves the sequencer's public key.
2. The smart contract computes the aggregated public key X

using the user's public key and the sequencer's public key.

1. The smart contract computes the challenge e = H(R \| X \| m \| i)

, where R = R_s + R_u

.

1. The smart contract verifies the sequencer's partial signature (s_s, R_s)

by checking if

(s - s_u)G == (R - R_u) + e(X - w_u P_u)

which equals to

s_s G == R_s + e(w_s P_s)

1. If the verification is successful, the smart contract slashes the sequencer's stake and compensates the user.

AA & zkSNARK

Validating Schnorr signatures on-chain is most cost-effective using zk-SNARK and account abstraction together. We can map user elliptic curve point to the contract account address using the CREATE2

algorithm and recursively prove the schnorr signatures inside of SNARK since they're just the combinations of elliptic curve points. If we choose a zk-friendly hash function to create the challenge e

, we can prove the validity without a huge cost. Therefore, it is worth noting that this model can be used not only for the layer 2 sequencer but also for transaction bundling in Account Abstraction, making it a versatile solution for transaction ordering and commitment in Ethereum.

MEV

However, there still exists a possibility of MEV by emptying some slots for front-running or cancelling the signature completion after getting the transaction information. To mitigate the MEV attacks, we can use the timelock cryptography to let the user share the pre-image when only it confirms the previous slots are all signed with. This approach might not be suitable for a system that needs a very high frequency transactions or need more DOS attack resistency.

Conclusion

Schnorr sequencer, or a mpc based sequencer, can give an economical instant finality for low-throughput systems. This approach can be used for application chains who wants to run their own rollup but doesn't have enough throughput to submit the batch to the layer 1. If we go further, we can consider a scenario that a rollup has multiple nodes which are connected through a peer-to-peer network and broadcast the signatures and then update the state tree instantly for each transaction. This will allow the system afford the higher throughput.