This proposal is inspired by earlier posts from@vbuterin (link) and @barryWhiteHat (link)

Overview

In the post, we introduce a SNARK based side chain for an ERC20-like token. This is useful for organizations that wants to issue their platform tokens (e.g. games, streaming platforms, and more). The token is governed by an on-chain smart contract, which maintains the token balance for each user. However, different from a standard ERC-20 token, in our case a special role called operator regularly submits blocks to the smart contract. Each block contains a list of token transfer transactions, along with a SNARK proof which only proves the validity of the user signatures. The smart contract verifies the SNARK proof and then executes the transactions to update the token balance for the users.

Compared to barryWhiteHat's proposal, this scheme might require much less computational overhead for SNARK generation, and yet the runtime complexity of the on-chain smart contract is the same. Below are the details.

System Spec

There are three roles. The users, a plasma operator, and an on-chain smart contract.

User Responsibility

: Users submit token transfer transactions to the operator. A transaction tx

is the concatenation of five strings:

tx

:= f

|| t

|| n

|| v

|| sig

Here f

is the address of the "from" account, t

is the "to" account, n

is the nonce of the from account, v

is the amount of tokens to be transferred, and sig

is the signature of the from account.

Plasma Operator Responsibility

: As the plasma operator accumulates the transactions, he packs them into blocks. A block is the concatenation of multiple transactions (e.g. thousands of transactions):

blk

:= tx

1

|| tx

2

|| … || txn

The operator also calculates the hash of the block blk

hb

:= HASH(blk

)

Then, the plasma operator generates a SNARK proof π, which proves the validity of the signatures of the transactions. To be more specific, the SNARK π proves that for input hb

, there exists a witness string blk

:= tx

1

|| tx

2

|| … || txn

, such that

(hb

== HASH(blk

)) AND (VERIFY(txi

, sigi

) == true for i

= 1, 2, …, n

)

The plasma operator then submits a tuple (blk

, hb

, π) to the smart contract.

Smart Contract Responsibility

: The smart contract stores the balance of each users on-chain. It has two responsibilities: 1) Penalize the plasma operator if he behaves maliciously, and 2) process the blocks submitted by the plasma operator.

To achieve the first goal, the plasma operator need to deposit a certain amount of Ethers into the smart contract as the collateral. The smart contract can detect malicious operator behaviors. Such behaviors include but not limit to: the blocks submitted contains invalid transactions (e.g. trying to send more tokens than the “from” account’s balance), or the SNARK proof is invalid, or the operator has not submit new blocks for a certain period of time. If any malicious behavior is detected, the operator will lose his collateral.

To process the blocks submitted, the smart contract needs to carry out the following tasks:

1. Firstly, it verifies the SNARK proof π with input hb

.

1. Secondly, it validates hb

== HASH(blk

). This is necessary since hb

is the input to the SNARK verification. If the SNARK verification passes, it means the submitted block blk

indeed contains valid signatures from the senders.

1. Finally, the smart contract execute each transaction in the block and update the account balances.

Smart Contract Runtime Analysis

This proposed scheme can reduce processing time per transaction significantly compared to a fully on-chain ERC20 smart contract.

Firstly, the verification of the signatures for each transaction in the block is replaced by one single

SNARK proof verification. The SNARK verification can be executed in constant amount of time (typically a couple milliseconds) since its input hb

is simply a hash, whose length is a constant.

Secondly, let us look at the transaction execution time. At the first glance, it might appear that in our proposal does not have advantage compared to a fully on-chain smart contract. However, we note that the plasma operator can pack many transactions (e.g. thousands) into one block. It could happen that many of these transactions have the same sender and/or receiver. Processing these transactions in batch reduces the time spent on account lookup and balance update. Let us look at an example. Assume a block contains multiple transactions between two accounts A and B:

tx

1

: A sends 60 tokens to B

tx

2

: A sends 20 tokens to B

tx

3

: B sends 50 tokens to A

tx

4

: A sends 30 tokens to B

...

When the smart contract processes these transactions together, it only needs to lookup the account of A and B once. It then calculates the final balance of A and B, and then update the accounts just once

. In comparison, with a fully on-chain smart contract, each of these transactions is processed individually. Each time the accounts of A and B need to be retrieved and updated. Thus the runtime complexity for our framework is O

(m

log N

- k

), Here m

is number of distinct accounts accessed by the transactions in the block. N is the total number of token holders, and k is the number of transactions in the block. On the other hand, with a fully on-chain smart contract, the runtime complexity is O

(k

log N

). Since m

could be much smaller than k

, considerable speedup can be achieved.

Finally we note that the smart contract needs to validate hb

== HASH(blk

). Even with thousands of transactions, the size of a block is still in the range of sub-hundred kilobytes (with some compaction). So validating the hash should only take at most a couple milliseconds.

With the above runtime estimations, the transaction throughput should reach at least hundreds of TPS, maybe even 1000+ TPS for the token.

SNARK Generation Complexity

One significantly difference of the proposed framework compared to barryWhiteHat's proposal is that our SNARK proof π only proves the validity of the sender signatures. It does NOT prove the validity of state transfer (i.e. the validity of the account balance update). This is intended to significantly reduce the SNARK generation time. Compared to barryWhiteHat's approach which calculates the account balances off-chain, in our proposal, the balances are calculated by the smart contract. However, the overhead is minimal, since calculating the account balance only involves simple addition/subtraction. The account lookup/update complexity are the same for the two approaches.

This trade-off leads to substantial reduction in terms of SNARK generation complexity. The runtime complexity of the the SNARK generation is a quasi linear function of the total size of the input and witness string (e.g. O

(s

log s

) where s is the total size). If we want to prove both the state transfer and the sender signatures are valid, this size is in the order of O

(m

log N

- k

). Here the notations are as above, m

is number of distinct accounts accessed by the transactions in the block, N

is the total number of token holders, and k

is the number of transactions in the block. On the other hand, if we only generate the proof for the signature, the size of the witness string is only O

(k

).

To get a more concrete idea, in barryWhiteHat's post, it is estimated that with optimization, for each transaction, proving the account is part of the state tree takes 29k * 2 = 58k constraints, while confirming one signature could only requires as little as 2k constraints. Thus, this could lead to more than 30X speedup for SNARK proof generation. With 1000 transactions per block, the total number of constraint for signature verification is only 2k * 1000 = 2 million transactions. Potentially the SNARK generation can be done on a single machine rather than using a high-end server cluster.

Remarks

SNARK as Mining

: An interesting variant of the proposed framework is to let users generate the SNARK proof instead of the plasma operator. Each time a new block is submitted to the smart contract by the operator, users can compete to generate the SNARK proof for the block. The first user submit the proof wins some newly minted tokens similar to mining reward. This way users can play the role of "SNARK Miners" to enhance the security of the system. If instead a user finds out that a block contains invalid signatures, he can inform the smart contract. If this is indeed the case, the smart contract can slash the Ether collateral of the plasma operator and award a portion of the collateral to the user.

Block and State Pruning

: We also note that after the smart contract validated a block and updated the account balances, the block is no longer needed. It can be deleted from the smart contract storage to save space. Thus, our proposal does not incur much extra space overhead besides the user account storage.