Error-correcting code to optimize ZK-rollup verifier

Thanks to @liochon and @OlivierBBB for reviews and discussion.

In a SNARK scheme, the verifier algorithm accepts a vector of public inputs. In Groth16 and, in fact, many other pairing-based schemes, the verifier performs a fixed base multi-exponentiation of the public inputs. In scalability applications of SNARK (for instance zk-rollups), a circuit is typically super-massive and takes a batch of transactions as public inputs. There are several ways to do it.

We propose an idea specifically tailored for rollups, which achieves a constant time overhead in computation at the expense of increased call-data utilization.

Classic techniques for passing many inputs

1/ No technic, just do it

All the public inputs are sent on-chain, and we run the verifier normally on the batch of inputs. The multi-exponentiation cost dominates and becomes impractical for moderately-large public inputs: ~6000 gas (+512 gas because of the call-data) per input so we are stuck at ~1000 public inputs.

2/ Using hashes

After this transformation, the public inputs are turned private and we use their hash as public input.

All the public data x = (x_1, x_2, .., x_n)

(which are now private inputs of the circuit) are sent in the block.

The verifier contract hashes the public inputs: h = Hash(x_1, x_2, x_3, ...)

.

The verifier contract calls the verifier algorithm using h.

The circuit check consistency between h

and its private input x

.

The tradeoff here (on EVM) is that:

- Either the hash function is efficient on the smart-contract (sha256, keccak256) and the gas cost is narrowed down to 20-30 gas per public input. Hence, the cost per public inputs become negligible compared to the call data costs. However, the cost for the prover jumps up to ~25k constraints.
- Or the hash function is efficient on SNARK (MIMC7, Poseidon, Pedersen Hash) and this yields no significant improvement on the verifier contract while still increasing a little the complexity of the circuit (~700 constraints/~8000 gas) for MIMC7. The tradeoff might be a lot more interesting if we have optimized support for MIMC in the EVM.

Seeing this, it could be tempting to directly pass the hash on-chain and pass it as public input:

The problem when doing this is that we can’t know what statement is being verified.

Even if the statement is passed along with the precomputed hash, we can’t be sure the public data is consistent with what is being proven.

Moving forward to a new construction

We propose an approach with the following performances:

- Logarithmic gas cost consistency check on the smart-contract
- A single public input
- Low-overhead on the circuit size
- Linear overhead on bandwidth (block size)

For this, we are going to use a linear error-correcting (not erasure) code L with an error-correction rate of r

. That means that if a portion rho

< r

of a code-words is wrong, we can recover the correct code-words. This is classic PCPP construction, also used for data-availability checks.

We pass l = L(x)

and a

, where a

is a Merkle Tree of all the values of L(x)

. Then we use a

as a source of randomness (following [Mic00] construction) to select several branches of the Merkle-tree a

until we are guaranteed with 128 bits of security that a

and l

agree on 1-r

of their values.

For n queries, we have soundness of (1 - r)^{-n}

. Morally, in the context of rollups, if one operator decides to lie on a few fields of l

, this won't prevent the other operators to recover the actual content of the transactions, hence no harm.

In the circuit, we pass a

as public input, and we extend the circuit in the following manner:

Open a

completely to L(x)

=> \frac{1400}{r}

constraints

Compute x

from L(x)

(linear operation => free)

Check L(x) is a proper codeword (check that \frac{|x|}{1-r}

linear combinations are zero)

In practice, for eth1 this method is impractical as it only becomes more efficient at ~10k inputs. The high gas cost of MIMC7 (compared to an EC exponentiation), the current cost of call-data make it impossible to get a significant improvement using this method. However, in a context where the call-data cost becomes negligible per input (against ~512 today) which is plausible with sharding, this approach becomes more interesting for batches with ~10000 public inputs (compared to approach 2).

Using multi-opening polynomial commitments

Following the idea of Vitalik and Justin to use polynomial commitments as a practical accumulator, we consider the application of multi-opening in our scenario. Using polynomial commitments to replace state roots. This results in a particularly efficient case of the previous construction.

Here, we represent the "actual" public inputs as coefficients of a polynomial P(X) = \sum_{i=0}x_ iX^i

. And publish the L(x) = (l_0, .., l_{\frac{n}{1 - r}}) = (P(0), .., P(\frac{n}{1 - r}))

evaluations of the polynomial. These will act the encoding of the actual n public inputs.

The prover compute (L(x), acc_x = g_1^{P(\alpha)}

) on-chain. Then, using Fiat-Shamir, he generates k challenge points j = (j_i)_{i \in [k]} \in [\frac{n}{1 - r}]^{k}

. If the prover has been honest, he is able to open the commitment and prove that: \forall i \in [k], P(j_i) = l_{j_i}

.

The on-chain verifier checks are the polynomial commitment verifier and he uses the acc_x

as the new public input.

Cost analysis:

In the smart-contract:

- Send \frac{n}{1-r}

values of P(x)

for x=0 … \frac{n}{1-r}

in the call-data

- Evaluate the coefficients of I(X)

and Z(X)

. Note that asymptotically k << n, and that the field has a good 2-adicity, this helps the FFT being reasonably efficient. In practice it is only going to need k=128 points for r=1/2.

- 2 Elliptic curves multiplication to get I

and Z

.

- 2 Pairing checks

In the circuits:

- Opens the polynomial commitment on its coefficients: n fixed base point exponentiation

Of course, the idea of using error-correcting code to relieve CPU a the expense of more call-data becomes pointless the EVM optimizes MIMC to a point where its cost becomes negligible compared to call-data and if the security of GMIMC becomes assessed enough to make it relevant. However, this method is applicable today.