nomadic labs

Zero-knowledge rollups : trading computing power for scalability

---

Miguel Ambrona, **Marc Beunardeau**, Victor Dumitrescu, Antonio Locascio, Marina Polubelova, Anne-Laure Schmitt, Marco Stronati, Raphael Toledo, Danny Willems

Journée scientifique Inria - Nomadic Labs
Paris, June 1st, 2022

What are zk-rollups

Scalability in blockchain

- A blockchain is a replicated system, every node recomputes everything $\Rightarrow$ more computing power $\neq$ more performances
- One person could compute for everyone, but the system could not be trustless
- Cryptography can help with this trust issue

Verifiable computations

given a program f, an input x

- A powerful party computes $y = f(x)$
- Also computes a proof π
- One can efficiently verify the correctness of the computation thanks to the proof

This seems adapted to our problem, but we will use something slightly different

Zero-knowledge proof

given a boolean program f (called statement) and two inputs x (public) and,w (private)

- $\mathrm{Prove}(x, w) = \pi$, $|\pi| = \mathcal{O}(1)$ runs in $\mathcal{O}(|f| \log |f|)$ (with a big constant)
- $\mathrm{Verify}(x, \pi) =$ bool, running time $\mathcal{O}(|x|)$, constant in $|f|$
- Verify that the prover knows some w such that $f(x, w)$ holds
- The advantage compared to verifiable computation is that we can hide part of the input

Note 1 : Zero-knowledge means that π gives no information on w, but we do not use this property
Note 2 : f is basically a circuit that does addition and multiplication modulo a big prime, more on that by Danny Willems

Naive scalability from zkp

- $\mathcal{T}(\mathcal{S}, t) = \mathcal{S}'$
- $\mathcal{S}_0 \xrightarrow{t_0} \mathcal{S}_1$
- Prove $\mathcal{T}(\mathcal{S}_0, t_0) == \mathcal{S}_1$
- Send the new state along with a proof
- The blockchain does not need to execute t_0

Problems:

1. The blockchain still needs to updates the whole state
2. The public inputs are big
3. Verifying a zkp is usually harder than executing a transaction

Solving problems 1 and 2

We will leverage private inputs :

- Hash the state $c_i = H(\mathcal{S}_i)$, c_i is constant size
- Prove $\mathcal{T}(\mathcal{S}_0, t_0) == \mathcal{S}_1$ and $c_{0,1} = H(\mathcal{S}_{0,1})$
- The blockchain stores and update only the hash of the state
- only the c_i are public inputs

Solving problem 3

We will leverage the constant time verification

- $\mathcal{S}_0 \xrightarrow{t_0} \mathcal{S}_1 \cdots \mathcal{S}_n \xrightarrow{t_n} \mathcal{S}_{n+1}$
- Prove for all i that $\mathcal{T}(\mathcal{S}_i, t_i) == \mathcal{S}_{i+1}$ and $c_{0,n+1} = H(\mathcal{S}_{0,n+1})$
- Only c_0 and c_{n+1} are public inputs

$\Rightarrow$ We achieved infinite scaling by building a zk-rollup !

Challenges of creating proofs

Prover performance

- Zkp incur a big overhead
- Proving 100 transactions is doable in $\approx$ 3 min with one core
- We want to prove one to two orders of magnitude more

We should use more than one core but the proving algorithm does not parallelise well

Aggegration

We developed an aggregation protocol to overcome this

- $\mathrm{Aggregate}(\pi_1, \cdots, \pi_n) = \Pi$ produces a proof that $\pi_1, \cdots \pi_n$ are valid proofs w.r.t public inputs $x_1, \cdots x_n$
- We parallelise the production of $\pi_1, \cdots \pi_n$, aggregate and send Π to the L1
- $\mathrm{Aggregate}$ has reasonable running time for $n = 100$

Problem: Π is verified in $\mathcal{O}(|x_1| + \cdots + |x_n|)$, our running time is still linear and this problem seems inherent

Handling public inputs in aggregation

idea: intermediate public inputs are intermediate state's hash $\Rightarrow$ L1 does not care about them

- $x_i = (c_i^{in}, c_i^{out})$
- $\mathrm{Aggregate}$ also proves $c_i^{out} = c_{i+1}^{in}$ for i from 0 to $n-1$
- only c_0^{in} and c_n^{out} need to be public

Now we really have infinite scaling : we add computing power to scale the blockchain

Implementation consideration

Implementation consideration

- Since our purpose is to scale we obviously want a fast implementation
- We want to be modular as we have a big stack of protocols which we want to be able to replace one by one based on new research

$\Rightarrow$ We use a mix of C and OCaml

C Implementation

- We bound *Blst*, a high performance library for our algebraic low level operations (field, elliptic curve, pairing)
- We developed a polynomial library with contiguous memory (for cache)
- Our polynomial have two representations, coefficient and evaluation with conversion between them (FFT)
- We expose an in place API to lower memory consumption

Note : we actually developed a generic contiguous C array library together with OCaml bindings

OCaml implementation

- We developed a functorised proving system based on the C libraries
- We used OCaml multicore for parallelisation (we will use Eio for distribution)
- We developed a monadic style DSL to write our statements with 2 different interpreters (good candidate for verification)

Questions ?