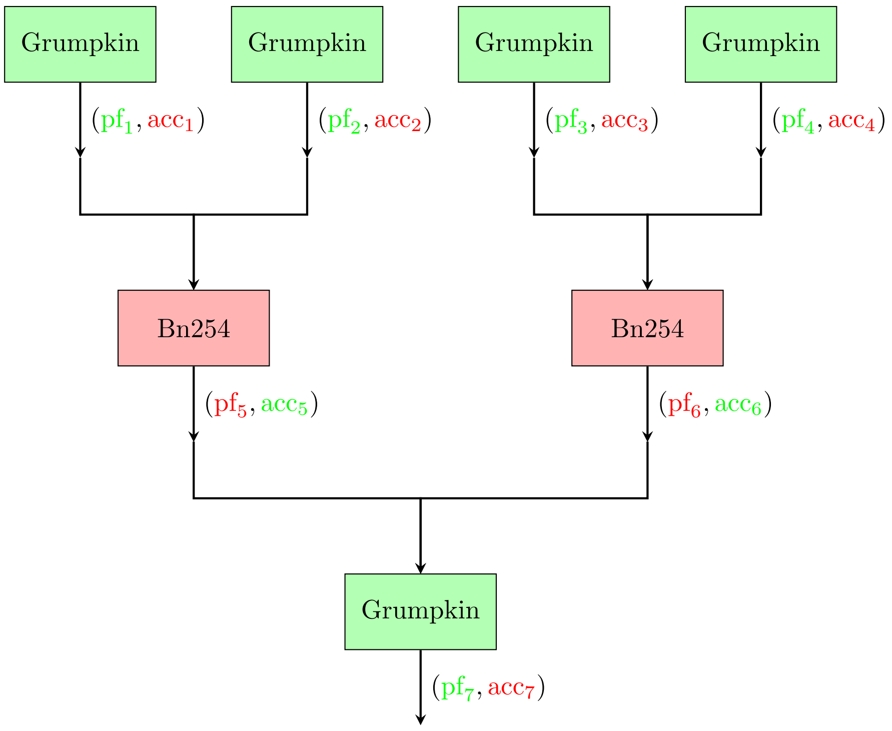
Recursive proving in nightfish

Basic model

Throughout this description of the basic model, the roles of Grumpkin and Bn254 can essentially be interchanged. The following diagram is to be thought of as a generic point in the middle of the recursive binary tree. The boxes all denote circuits, either Grumpkin or Bn254.


Grumpkin
Grumpkin
Grumpkin
Grumpkin
$(\mathrm{pf}_1, \mathrm{acc}_1)$
$(\mathrm{pf}_2, \mathrm{acc}_2)$
$(\mathrm{pf}_3, \mathrm{acc}_3)$
$(\mathrm{pf}_4, \mathrm{acc}_4)$
Bn254
Bn254
$(\mathrm{pf}_5, \mathrm{acc}_5)$
$(\mathrm{pf}_6, \mathrm{acc}_6)$
Grumpkin
$(\mathrm{pf}_7, \mathrm{acc}_7)$

$\mathrm{acc}_1, \dots, \mathrm{acc}_4$ denote Bn254 accumulators. This means each one is a commitment, point, evaluation and proof associated to a polynomial defined over the scalar field of Bn254. pf_5 and pf_6 denote Bn254 proofs. Again, these are proofs about statements made over the scalar field of Bn254.

The bottom Grumpkin circuit in the diagram partially verifies pf_5 and pf_6 and then accumulates them with $\mathrm{acc}_1, \dots, \mathrm{acc}_4$ into acc_7 and provides a proof of this accumulation pf_7. This means the veracity of $\mathrm{acc}_1, \dots, \mathrm{acc}_4, \mathrm{pf}_5$ and pf_6 is condensed into the veracity of a single accumulator acc_7. pf_7 is a proof that this accumulation was done correctly.

Until we reach the final layer and the decider circuit, none of the proofs are actually fully verified. We simply accumulate proofs into accumulators, provide proofs said accumulation was done correctly and then further accumulate these new proofs.

Details

- Base (client) circuits are all Bn254 proofs.
- Each client circuit has 4 commitments as input. The spend commitment, the change commitment, the spend fee commitment and the change fee commitment.

- At each layer a proof is a type of Plonk SNARK proof.
- For a Bn254 circuit, the underlying polynomial commitment scheme is KZG over the Bn254 curve.
- For a Grumpkin circuit, the actual proving system is HyperPlonk where the underlying polynomial commitment scheme is Zeromorph using IPA over the Grumpkin curve.
- We cannot use ordinary Plonk on the Grumpkin layers, as the relevant field does not have a large multiplicative 2-subgroup.

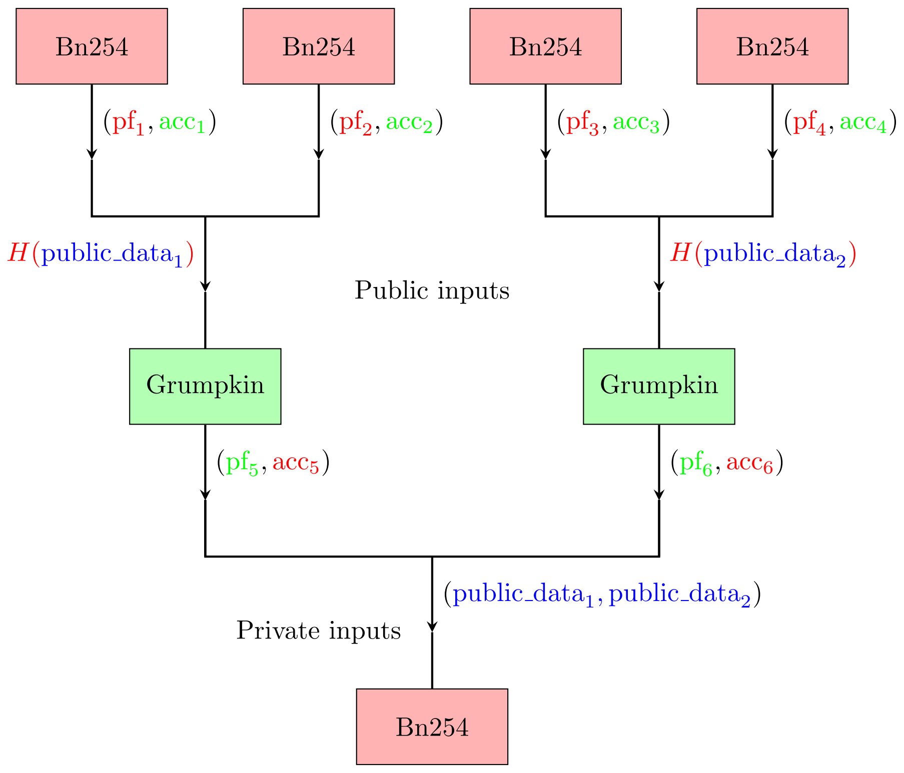
Atomic vs split accumulation

The Bn254 accumulators are called *atomic accumulators*. These carry around with them a proof of the claimed commitment, point and evaluation. However, the Grumpkin accumulators are called *split accumulators*. Instead of storing an opening proof, a split accumulator stores the polynomial itself. (This is also true for the Grumpkin proofs throughout the binary tree.) However, the verification of the accumulation does not require the polynomial. Therefore, only the proving of the accumulation requires access to the polynomial, until the final layer when the accumulation polynomial must be shown to open at the relevant point. This helps us to keep the verification costs sublinear in the degree of the polynomial. Without this, our circuits would increase in size as we moved through the recursive tree.

Public inputs

To avoid the number of public inputs growing at each layer, we hash together all the public data at each level and have a single public input to each circuit. All these unhashed preimages are then always private inputs to the next circuit. These preimages are then hashed together using a rescue hash defined over the scalar field in this next circuit.

The way in which equality of the hash of these private inputs and the public inputs to the previous circuits is enforced is that the public input is an input to the transcript. In the later circuit, the private inputs are used to reconstruct this hash and input to the transcript thus enforcing the desired equality.



Bn254
Bn254
Bn254
Bn254
$(\mathrm{pf}_1, \mathrm{acc}_1)$
$(\mathrm{pf}_2, \mathrm{acc}_2)$
$(\mathrm{pf}_3, \mathrm{acc}_3)$
$(\mathrm{pf}_4, \mathrm{acc}_4)$
$H(\mathrm{public\_data}_1)$
Public inputs
$H(\mathrm{public\_data}_2)$
Grumpkin
Grumpkin
$(\mathrm{pf}_5, \mathrm{acc}_5)$
$(\mathrm{pf}_6, \mathrm{acc}_6)$
$(\mathrm{public\_data}_1, \mathrm{public\_data}_2)$
Private inputs
Bn254

In the above diagram public_data_1 and public_data_2 are vectors of public data from the previous circuits. (We go into more detail below about exactly what they consist of.) Note that public_data_1 and public_data_2 are private inputs to the circuit at the next layer.

Accumulators

Continuing the notation of the above diagram, contained within public_data_1 is acc_1, acc_2 and acc_5. Similarly, contained within public_data_2 is acc_3, acc_4 and acc_6.

Scalars threaded forward

During the proving of an accumulation, scalar arithmetic is performed in the wrong (base) field. Mainly this happens in the partial proof verifications but there is also a wrong field challenge scalar generated during accumulation. We calculate these values outside of the circuit and simply use them within the current circuit. They are then verified in the next layer, where they are considered to be in the right field. Again, this is achieved by them being a part of the hashed public data.

For example, in the above diagram, the wrong field scalar arithmetic required to partially verify pf_1 and pf_2, and accumulate into acc_5, is not performed in the subsequent Grumpkin circuit. Instead, the relevant scalars are included in public_data_1. They are then passed in as private inputs to the next Bn254 circuit, where the calculation is verified.

Additionally, transcript verification is only done in Bn254 circuits. So we need to thread the appropriate challenges through Grumpkin circuits.

Calculation of new root

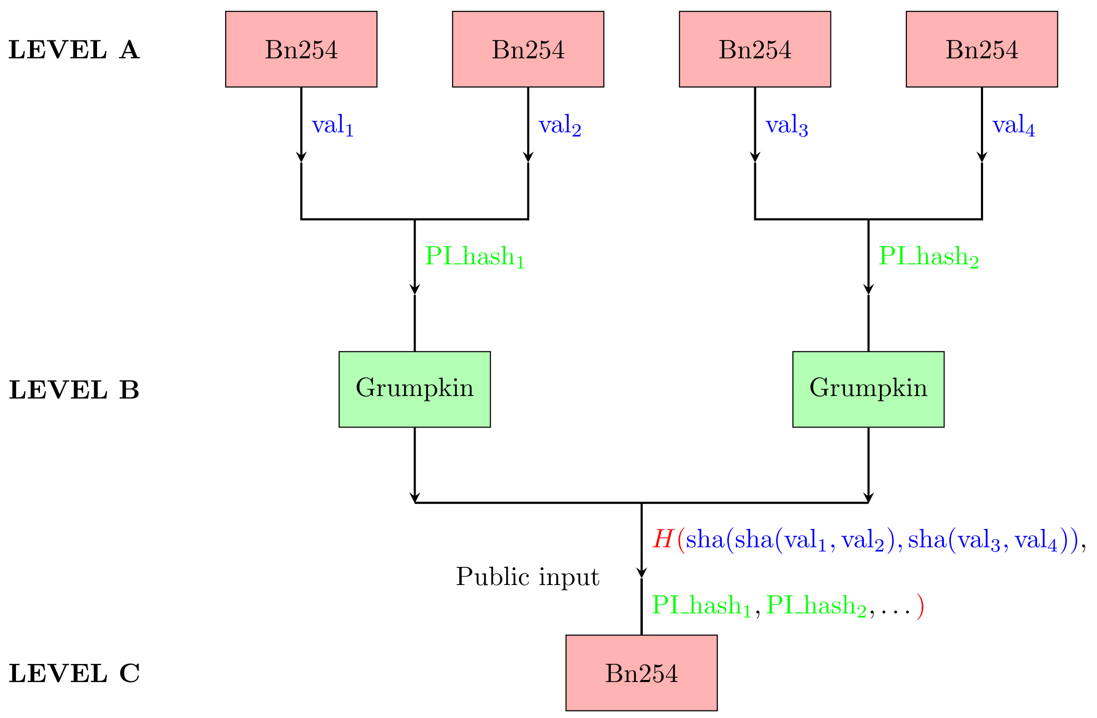
As mentioned in the first section, each base (client) circuit has 4 commitments as input. These need to be added to the Merkle tree and a new root calculated. We perform the relevant Merkle path calculations in the next layer of Bn254 circuits. (In other words, we skip the first Grumpkin layer, as we want to perform these expensive calculations in a Bn254 circuit). So, for each Bn254 circuit in this layer, we perform a Merkle root caculation with $4 \times 4 = 16$ commitments.

The circuits in all subsequent layers check that the new root of the input circuit on the left is equal to the old root of the input circuit on the right. Then it takes its old root to be the old root of the input circuit on the left and its new root to be the new root of the input circuit on the right. Again, these roots are passed through the recursive tree using the hashed public data.

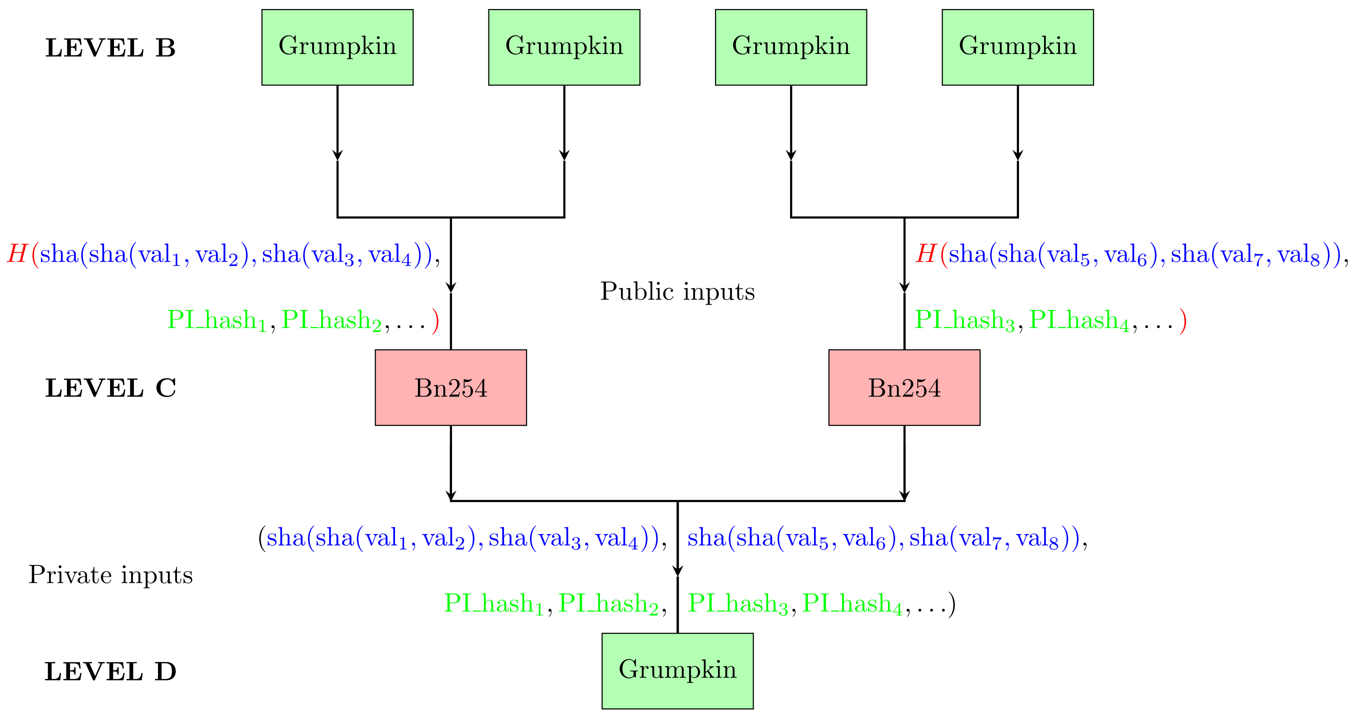
Hashed transaction data

We hash all the transaction data from the base layer using the sha256 hash recursively through the binary tree. We will refer to the current state of the recursive hash at a particular node. By this we mean all the transaction data associated with leaves above this particular node hashed together in this binary fashion.

At each Bn254 circuit the current state of the recursive hash is included in the hashed public data. At each Grumpkin circuit the current state of the recursive hash at this node is not a part of the public input nor the private input to the output Bn254 circuit. In other words, this recursive hashing only happens at every other layer and here we hash together 4 values from the layer two previous:

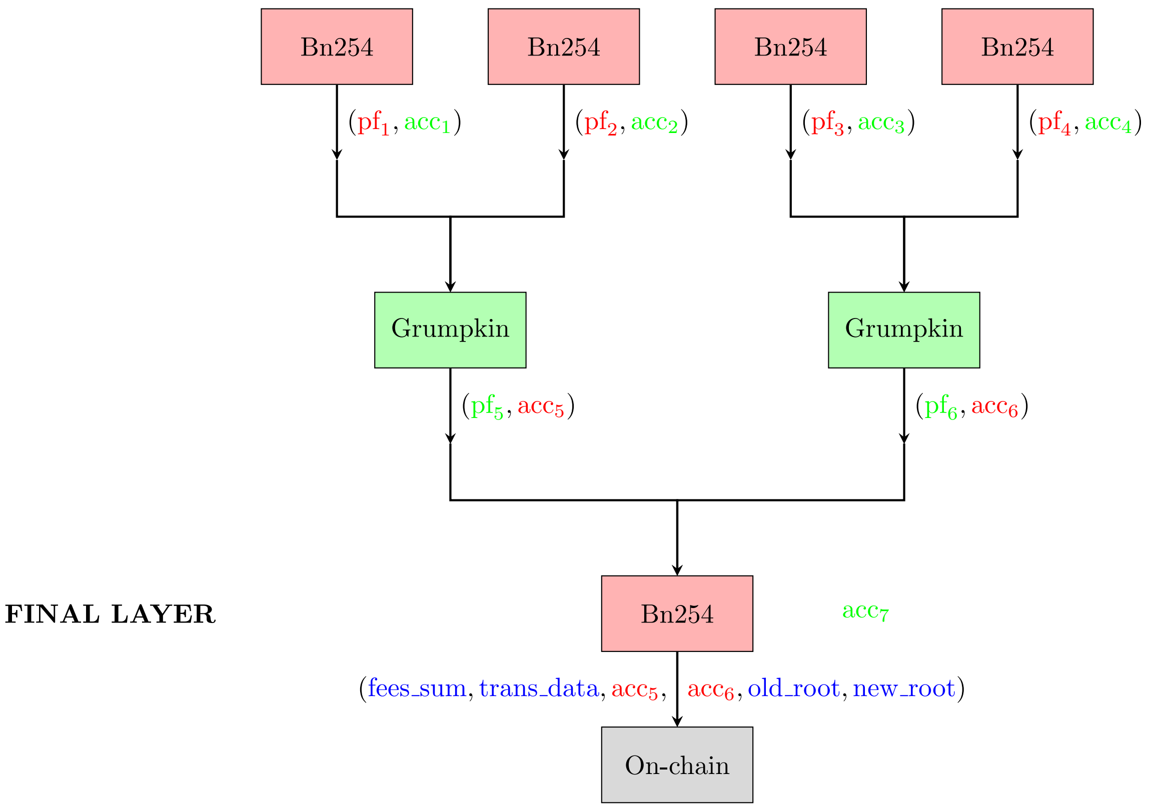


And now we have the same diagram but one layer further down:


LEVEL B
Grumpkin
Grumpkin
Grumpkin
Grumpkin
$H(\text{sha}(\text{sha}(\text{val}_1, \text{val}_2), \text{sha}(\text{val}_3, \text{val}_4)),$
$\text{PI\_hash}_1, \text{PI\_hash}_2, \ldots)$
Public inputs
$H(\text{sha}(\text{sha}(\text{val}_5, \text{val}_6), \text{sha}(\text{val}_7, \text{val}_8)),$
$\text{PI\_hash}_3, \text{PI\_hash}_4, \ldots)$
LEVEL C
Bn254
Bn254
Private inputs
$(\text{sha}(\text{sha}(\text{val}_1, \text{val}_2), \text{sha}(\text{val}_3, \text{val}_4)),$
$\text{PI\_hash}_1, \text{PI\_hash}_2,$
$\text{sha}(\text{sha}(\text{val}_5, \text{val}_6), \text{sha}(\text{val}_7, \text{val}_8)),$
$\text{PI\_hash}_3, \text{PI\_hash}_4, \ldots)$
LEVEL D
Grumpkin

The above diagrams show how the hashed transaction data $(\text{val}_1, \text{val}_2, \ldots)$ propagates through the binary tree. They also show the public input into a circuit being included in the hashed public data to the next. (See PI_hash_1, PI_hash_2, PI_hash_3 and PI_hash_4.)

Submitted on-chain


Bn254
Bn254
Bn254
Bn254
$(\mathrm{pf}_1, \mathrm{acc}_1)$
$(\mathrm{pf}_2, \mathrm{acc}_2)$
$(\mathrm{pf}_3, \mathrm{acc}_3)$
$(\mathrm{pf}_4, \mathrm{acc}_4)$
Grumpkin
Grumpkin
$(\mathrm{pf}_5, \mathrm{acc}_5)$
$(\mathrm{pf}_6, \mathrm{acc}_6)$
FINAL LAYER
Bn254
acc_7
$(\mathrm{fees\_sum}, \mathrm{trans\_data}, \mathrm{acc}_5, \mathrm{acc}_6, \mathrm{old\_root}, \mathrm{new\_root})$
On-chain

The $\mathrm{acc}_1, \ldots, \mathrm{acc}_4, \mathrm{pf}_5$ and pf_6 are accumulated into acc_7 and then fully verified in the final Bn254 circuit. acc_5 and acc_6 are submitted directly on-chain for verification. The fees_sum is calculated throughout the binary tree and also submitted on-chain. The fees must be public as the proposer must know how much they are being compensated.