Abstract

Current stable zkSNARK solutions are working well in a specific environment for at least a little bit heavy circuits. That's why we built Fawkes-Crypto in rust.

This is a lightweight framework for building circuits in bellman, using the groth16 proving system and BN254 curve.

The framework is targeted to use best practices for circuit building from circom and sapling-crypto.

Source code and a technical description

Available at https://github.com/zeropoolnetwork/fawkes-crypto.

Also, you may check the rollup example.

Benchmarks

Circuit

Constraints

Per bit

poseidon hash (4, 8, 54)

255

0.33

jubjub oncurve+subgroup check

19

ecmul_const 254 bits

513

2.02

ecmul 254 bits

2296

9.04

poseidon merkle proof 32

7328

poseidon eddsa

3860

rollup 1024 txs, 2^32 set

35695616

At i9-9900K rollup is proved for 628 seconds.

To reach these results we are using some improvements in implementation and circuit design.

The source code of the rollup is available athttps://github.com/snjax/fawkes-rollup.

Improvements

Elliptic curve point multipliers

Arbitrary point multipliers are implemented only for subgroup nonzero points.

To multiply point P

with bitset b_i

the first we compute Montgomery representation M = toMont(P)

and compute all power of two degrees of the point:

M,\ 2M,\ 4M,\ 8M\ ... \ 2^n M

To compute ecmul we select elements for nonzero bits and perform the addition.

For addition we can use cheap montAdd

function, but it has special cases for O

point and doubling. To prevent such cases we initialize the adder with (0,0)

point. This point is not in the subgroup, we will never reach doubling or O

point during the addition. After the addition, we convert adder into Edwards (x,y)

point. As the result we got (-x, -y) = bP

.

Addition for ecmul_const

is the same.

Jubjub oncurve+subgroup check

The method is described here.

For point P

we precompute point Q

, such that P=8Q

(during witness computation). After that, we prove this simple equation at the circuit and perform also on_curve check.

Constant comparisons

The method is very close to circomcompconstant. The related PR into circomlib is here.

To compare signal bit set a_i

with a constant bit set b_i

we split signal bit sets into 2-bit chunks. Each chunk could be compared with a corresponding constant chunk for 1 constraint and as output, we have {-1, 0, 1} (for less, equal, and more).

We can add these results to 1 << 127 - 1

with corresponding shift left (0 for lowest 2 bits, 1 for next 2 bits, etc). After that, we bitify the adder to 128 bits and check the highest one. a > b

if and if then the bit is equal 1.

Signal abstraction

In Fawkes-Crypto we are using Signal

instead of using allocated variables directly.

Signal

is a sparse linear combination of inputs, based on ordered linked list, so we perform arithmetics with Signal

with O(N)

complexity. With Signal

bellman will allocate additional inputs only when you really need it (for example, in the case when you multiply two

nonconstant Signal

). If you perform multiplication with constant or zero Signal

, no additional inputs will be allocated.

Wrapped math for field elements

Final fields and circuit math are wrapped and operators are implemented, so, in most cases if you want to type a+b

, you can do it.

Disclaimer

Fawkes-Crypto has not been audited and is provided as is, use at your own risk.