CONSENSYS

research.consensys.net

Optimizing cryptographic algorithms in gnark

Devcon Bogota 2022

Youssef El Housni

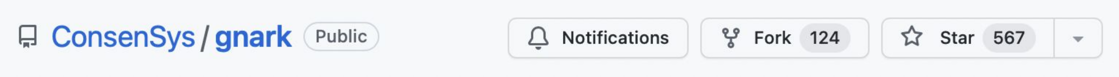
Team

Who?

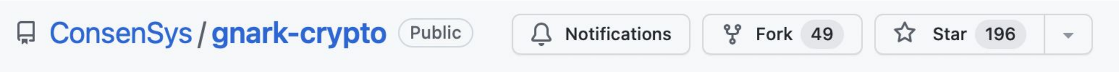
- Arya Pourtabatabaie
- Ivo Kubjas
- *Youssef El Housni*
- Thomas Piellard
- Gautam Botrel

What?

We're building gnark, a fast and easy to use open source zkSNARK library, in Go.



and gnark-crypto, a fast cryptographic library, in Go.



gnark under the hood

Frontend
(write a "circuit")

Backend
(proof generation & verification)

- Groth16, PLONK w/ KZG or FRI
- stdlib: MiMC, E(d/C)DSA, pairing, BLS sig., KZG…
- Native and non-native field arithmetic

Pairing and elliptic curve cryptography

gnark-crypto

- BN254, BLS12-381, BLS12-377/BW6-761, BLS24…
- Fast MSM, fast pairings
- KZG, FRI, Plookup…
- Sumcheck (GKR)

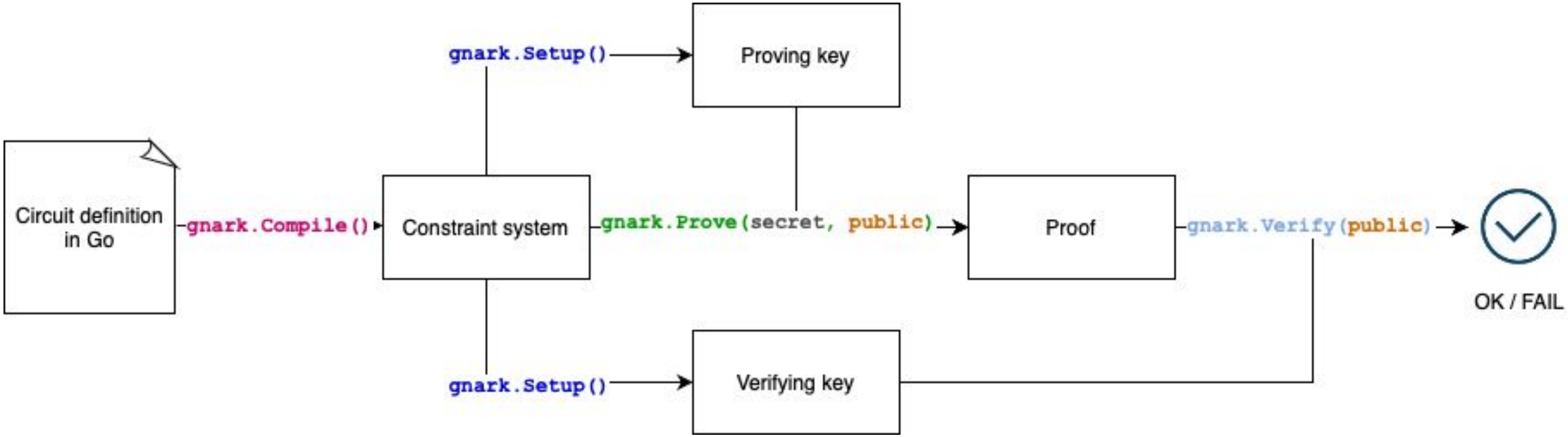
Field arithmetic (~big integer library)

gnark-crypto

- 768-bit, 384-bit, 256-bit, goldilocks… on multi-targets
- FFT, Pornin's optimized inverse…

gnark workflow

```
pk, vk, err := groth16.Setup(ccs)
proof, err := groth16.Prove(ccs, pk, witness)
err := groth16.Verify(proof, vk, publicWitness)
```




```
ccs, err = frontend.Compile(ecc.BN254.ScalarField(), r1cs.NewBuilder, &c)
ccs, err = frontend.Compile(ecc.BLS12_381.ScalarField(), scs.NewBuilder, &c)
```

gnark circuits: *play.gnark.io*

The gnark playground

Groth16 PlonK Run Share Examples

```
// gnark is a zk-SNARK library written in Go. Circuits are regular structs.
// The inputs must be of type frontend.Variable and make up the witness.
// The witness has a
//       * secret part --> known to the prover only
//       * public part --> known to the prover and the verifier
type Circuit struct {
    Secret frontend.Variable // pre-image of the hash secret known to the prover only
    Hash   frontend.Variable `gnark:",public"` // hash of the secret known to all
}

// Define declares the circuit logic. The compiler then produces a list of constraints
// which must be satisfied (valid witness) in order to create a valid zk-SNARK
// This circuit proves knowledge of a pre-image such that hash(secret) == hash
func (circuit *Circuit) Define(api frontend.API) error {
    // hash function
    mimc, _ := mimc.NewMiMC(api)

    // hash the secret
    mimc.Write(circuit.Secret)

    // ensure hashes match
    api.AssertIsEqual(circuit.Hash, mimc.Sum())

    return nil
}
-- witness.json --
{
```

▶ Proof is valid ✓

▼ 274 constraints

L·R == 0

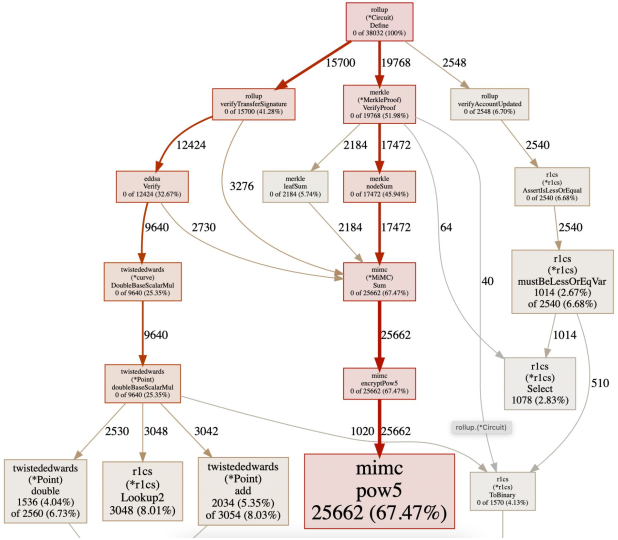
#	L	R
0	227063593160049201514509818732644766896230235191445544141110657236065169432·1 + Secret	22706359316004920151450 Secret
1	v0	v0

gnark circuit compiler: specialized circuit, ecosystem agnostic

[gnark]

- No DSL, plain Go, **no dependencies**
- Compiles large circuit (seconds)
- Playground, constraints profiler, …
- Write circuit once, use different curves and backends
- 2-chains, best-in-class 1-layer of recursion
- Several packages audited (Algorand) and fuzz-tested for months (geth)
- One code base which performs well on:
    - Server (CPU)
    - Mobile (70% faster than zprize)
    - WASM (30% faster than zprize)

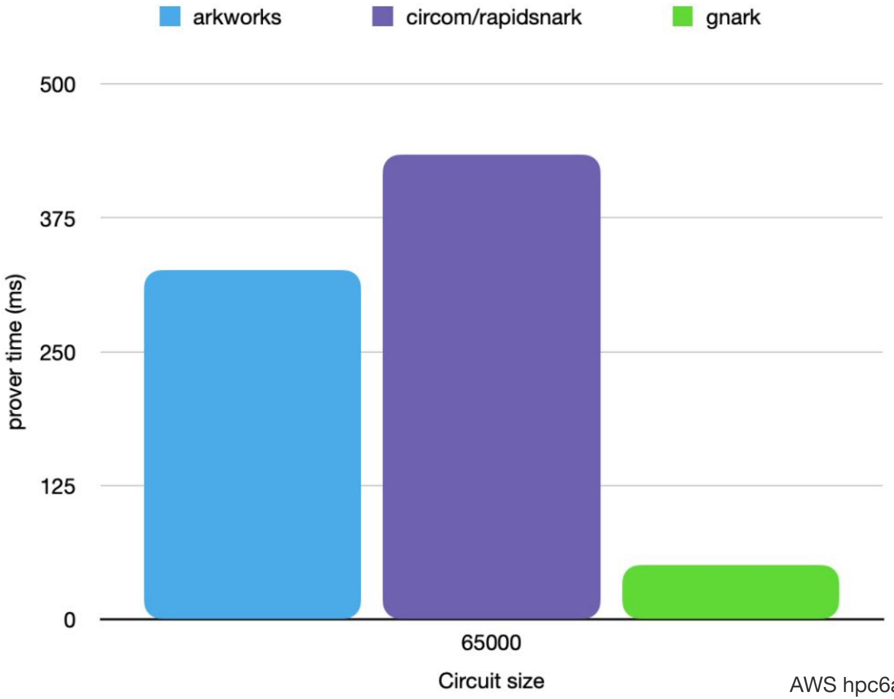
```go
func (circuit *Circuit) Define(api frontend.API) error {
    // compute x**3 and store it in the local variable x3.
    x3 := api.Mul(circuit.X, circuit.X, circuit.X)

    // compute x**3 + x + 5 and store it in the local variable res
    res := api.Add(x3, circuit.X, 5)

    // assert that the statement x**3 + x + 5 == y is true.
    api.AssertIsEqual(circuit.Y, res)
```

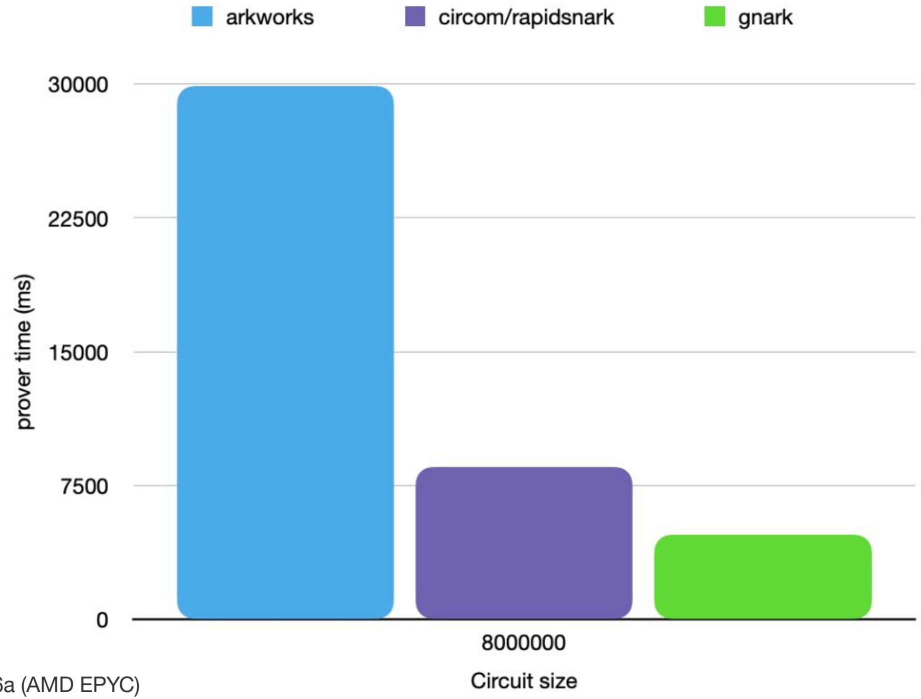


Constraints profiler

gnark is very fast

Groth16 SNARK prover on BN254: MSM, FFT, parallelism

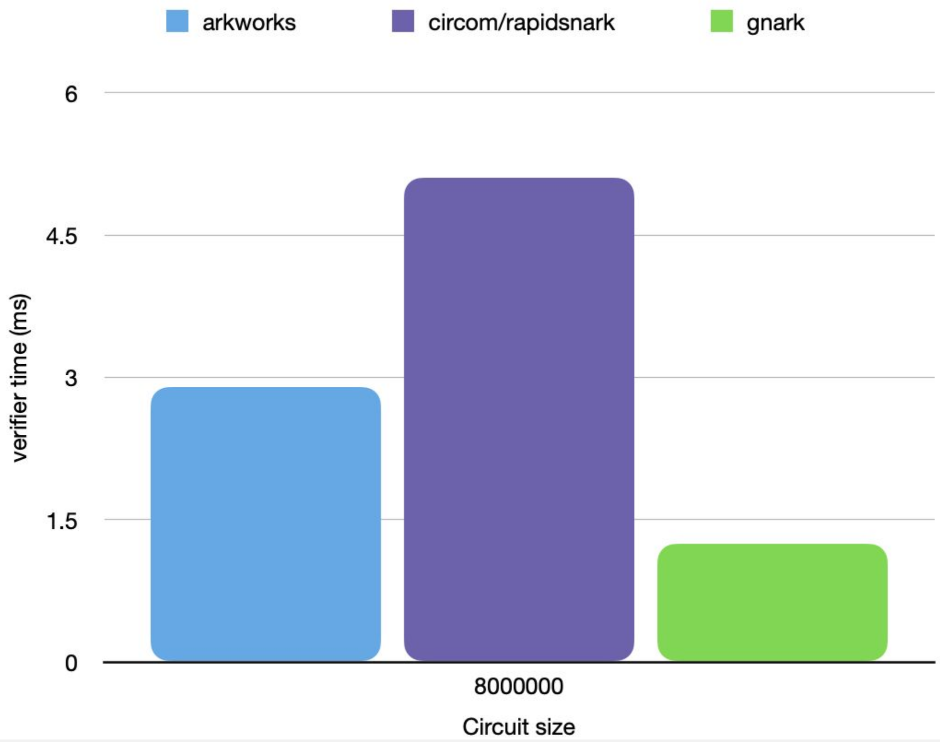




AWS hpc6a (AMD EPYC)

gnark is very fast

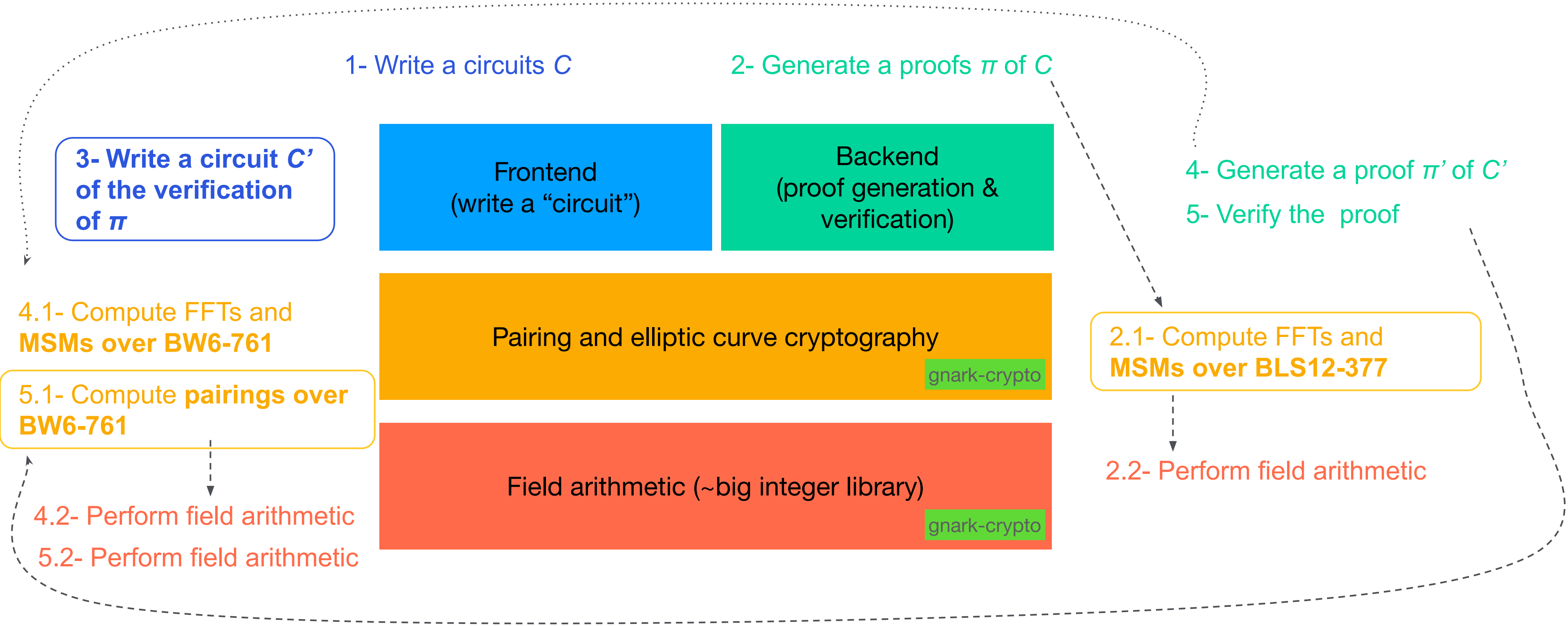
Groth16 SNARK verifier: Pairing on BN254



AWS hpc6a (AMD EPYC)

Why is gnark that fast?

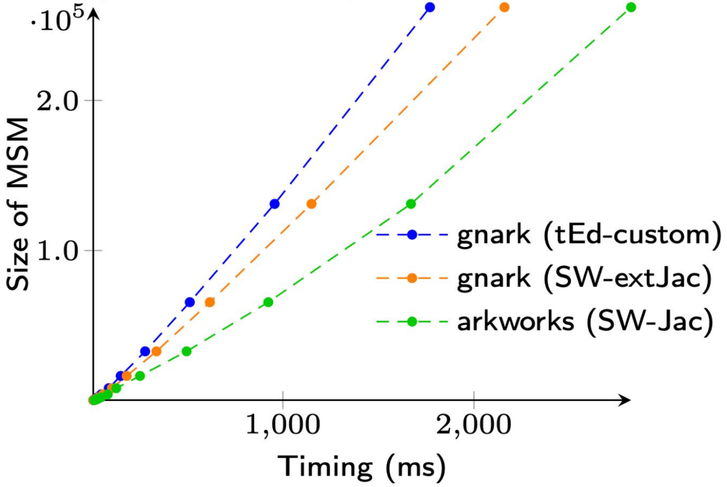
Example: 1-layer recursive Groth16 proof



Why is gnark that fast?

2.1- Compute FFTs and **MSMs over BLS12-377**

Fig. 2. MSM (BLS12-377 $\mathbb{G}_1$) comparison: gnark vs. arkworks



Samsung Galaxy A13 5G (SoC MediaTek Dimensity 700 (MT6833)).

- Speedup 40-47% (tEd-custom)
- Speedup 20-35% (SW-extJac)

Implementation	Timing	Curve form and coordinates system	Parallelism?	Precomputation?	2-NAF buckets?
arkworks	924 ms	SW Jacobian (X, Y, Z)	✓	✗	✗
gnark	512 ms	tEd $(a = -1)$ Custom (X, Y, T)	✓	✗	✓

https://github.com/gbotrel/zprize-mobile-harness/blob/main/msm.pdf

Why is gnark that fast?

b-bit MSM: $a_1 * G_1 + \cdots + a_n * G_n$

- Step 1: reduce the b-bit MSM to several c-bit MSMs for some chosen fixed c ≤ b
- Step 2: solve each c-bit MSM efficiently
- Step 3: combine the c-bit MSMs into the final b-bit MSM

→ Overall cost is: **b/c*(n + 2^{c-1}) + (b − c − b/c − 1)**

➢ Mixed re-additions: to accumulate G_i in the c-bit MSM buckets with cost **b/c* (n − 2^{c-1} + 1)**

➢ Additions: to combine the bucket sums with cost **b/c*(2^c − 3)**

➢ Additions and doublings: to combine the c-bit MSMs into the b-bit MSM with cost **b−c+b/c−1**

- ○ **b/c − 1** additions and
- ○ **b − c** doublings

twisted Edwards	extended $(XYZT)$ $x = X/Z, y = Y/Z, x \cdot y = T/Z$	$-x^2 + y^2 = 1 + dx^2y^2$ $(a = -1)$	**7m** (dedicated) **8m** (unified)

+All inner BLS:

$-x^2 + y^2 = 1 + (7+4\sqrt{3}) * x^2 y^2$

+Custom tEd extended coordinates

(X,Y,T)=(y-x, y+x, 2d*x*y)

+Parallelism, 2-NAF buckets…

Why is gnark that fast?

3- Write a circuit *C'* of the verification of π

Implementation open-sourced (MIT/Apache-2.0) at
https://github.com/ConsenSys/gnark
e.g. For BLS12-377,

	Constraints
Pairing	**11535**
Groth16 verifier	19378
BLS sig. verifier	14888
KZG verifier	20679

https://eprint.iacr.org/2022/1162.pdf

Miller loop:

+ Affine coordinates → ≈ 19k (arkworks)
+ Division in extension fields
+ Double-and-Add in affine
+ lines evaluations (1/y, x/y)
+ Loop with short addition chains
+ Torus-based arithmetic

Final Exponentiation:

+ Karabina cyclotomic square
+ Torus-based arithmetic
+ Exp. with short addition chains

Why is gnark that fast?

5.1- Compute **pairings over BW6-761**

$$e(P,Q) = m(P,Q)^{(q^6-1)/r}$$

integer	bitsize	Binary HW	2-NAF HW
u+1	64	7	7
u^3-u^2-u	190	136	**31**
(u-1)^2	127	15	**12**

arkworks: $m(P,Q) = f_{u+1,Q}(P) \cdot f^q_{u^3-u^2-u,Q}(P)$

gnark: $m(P,Q) = f_{u+1,Q}(P) \cdot (f_{u+1})^q_{(u-1)^2,[u+1]Q}(P) \cdot l^q_{[(u+1)(u-1)^2]Q,-Q}(P)$

1 pairing over BW6-761 AWS z1d.large (3.4 GHz Intel Xeon)	arkworks	1.71 ms
	gnark	1.22 ms

https://eprint.iacr.org/2021/1359
https://hackmd.io/@gnark/BW6-761-changes

research.consensys.net

Questions?

gnark@consensys.net
@gnark_team

play.gnark.io
github.com/ConsenSys/gnark
github.com/ConsenSys/gnark-crypto