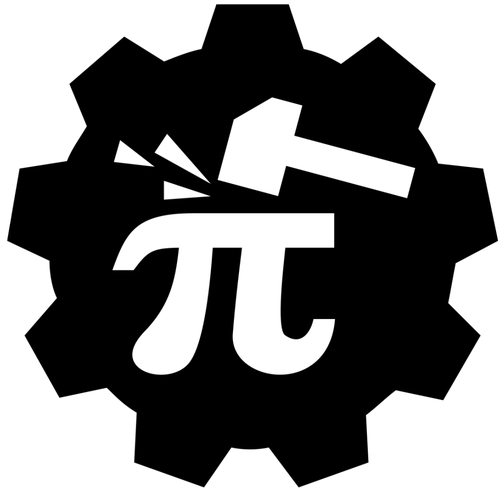


arkworks

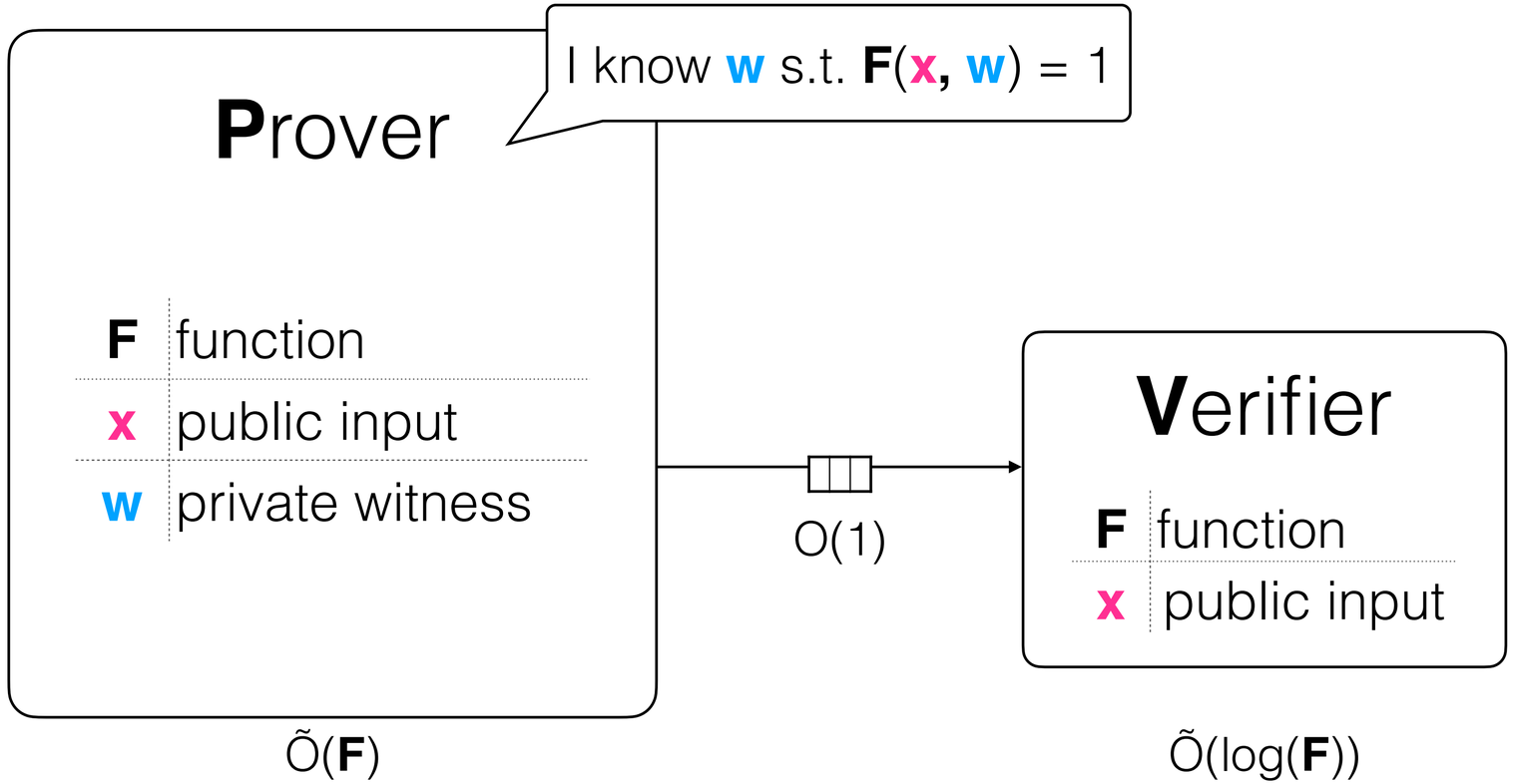
Rust Ecosystem for zkSNARKs

Sarah Bordage, **Giacomo Fenzi**,
Ziyi Guan, Pratyush Mishra,
Hossein Moghaddas, Ngoc Khanh Nguyen

arkworks.rs

Succinct Non-Interactive Arguments (SNARKs)

[Mic94, Groth10, GGPR13, Groth16…
…, GWC19, CHMMVW20, …]

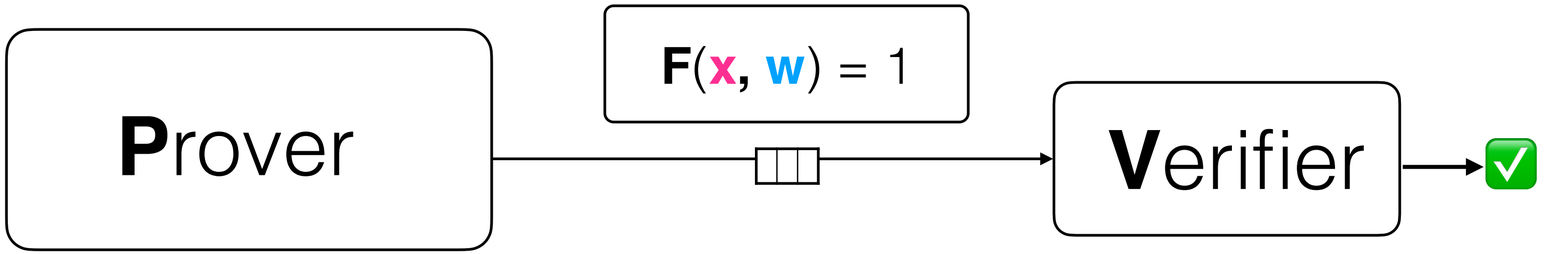


Zero Knowledge: V learns nothing about **w** except that **F**(**x**, **w**) = 1

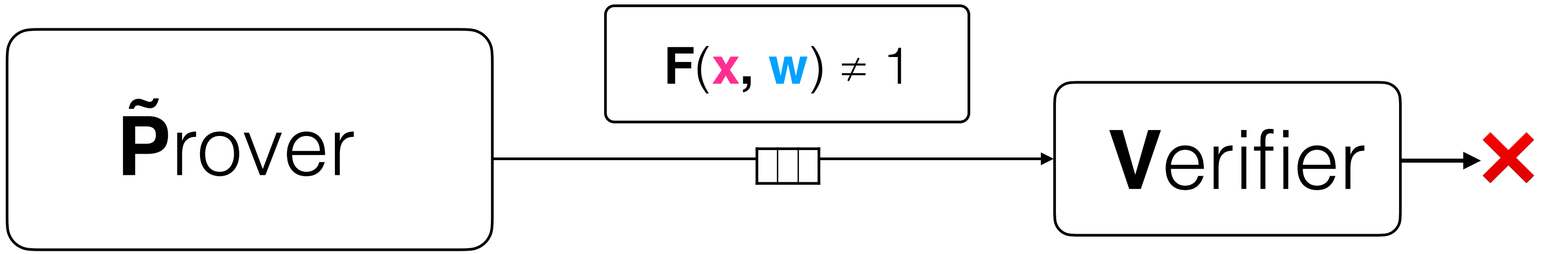
Succinctness: V runs in time much less than |**F**|

SNARKs are Secure

- **Complete**: If the prover's claim is true, then they can certainly convince the verifier.

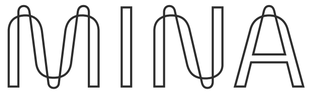


- **Sound**: If the claim is untrue, no prover can convince the verifier.



Many applications!

Private transactions



…

Scalable and/or Private Smart Contracts

Decentralized multiplayer games

- **Anonymous credentials** [DFKP16]
- **Prove existence of security vulnerability** [DARPA Sieve, OBW22]
- **Coercion-resistant voting** [MACI]
- …

What does it take to develop a zkSNARK application?

Application

Express your computation in the zkSNARK language

Many languages and frameworks!

Leo, ZoKrates, Noir, Circom, snarky, libsnark gadgetlib, bellman, Cairo, …

Choose a zkSNARK for your use case

Many different kinds of SNARKs!

Groth16, PLONK [GWC19 Marlin [CHMMVW20], STAR [BBHR18] …

Choose your algebraic object

Many different choices of and elliptic curves!

BLS12-381, BLS12-37 Ristretto, Pasta, BN25 secp256k1, …

Implement and optimize your algebraic object

Many different things to optimize!

Fast pairing formulae, specialized modular reductions, endomorphism-based multiplcation, …

How are SNARK apps developed today?

Everybody reimplements each portion, from scratch!

circom

gnark

libsnark

bellman

Why?

- Optimization to overcome overhead of SNARKs
- New algorithms and techniques every few months
- Need to interface well with existing codebase

Problems with this approach

- Tons of wasted reimplementation effort
- Optimizations are scattered across libraries
- Upgrading becomes difficult
- Auditing and bug-fixing efforts are split

How do we overcome these issues?

By making it easy and low-cost to re-use a library!

Modular

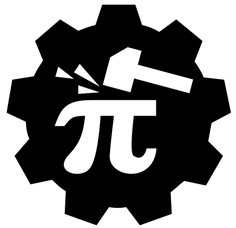
It should be easy to mix-and-match components, and swap out outdated techniques for newer ones

Ergonomic

It should be easy to write secure protocols by default, and then integrate them into existing codebases

Efficient

It should be easy to get performance matching best specialized implementations



arkworks

Modular, Ergonomic, Efficient Rust Ecosystem
for zkSNARK programming

Modular

Leverages the Rust **type system** to abstract away details and offer **reusable interfaces** at each level of the zkSNARK stack

Ergonomic

Encapsulates implementation details to offer **secure defaults**. Leverages **Rust cross-compilation and FFI** features to enable easy interfacing with other languages

Efficient

Offers state-of-the-art **assembly/intrinsic** implementations of **field** and **curve** arithmetic, and allows curve-specific optimizations

R1CS

- Recall **F**, **x** were public parameters
- The prover want to prove “There exists **w** s.t. **F**(**x**, **w**) = 1”
- Example of a **NP** statement
- What should the format for **F**, **x** be?
- Some representations have more efficient zkSNARKs

R1CS

- Let $\mathbb{F}$ be finite field
- The **R**ank **1** **C**onstraint **S**atisfaction relation is

$$((\mathbf{A}, \mathbf{B}, \mathbf{C}, \vec{x}), \vec{w}) \in \text{R1CS}(\mathbb{F})$$

If and only if

$$\mathbf{A}\vec{z} \odot \mathbf{B}\vec{z} = \mathbf{C}\vec{z}, \text{ with } \vec{z} = (1, \vec{x}, \vec{w})$$

$$\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times n}, \vec{x} \in \mathbb{F}^{n_{in}}, \vec{w} \in \mathbb{F}^{n - n_{in} - 1}$$

- Maps to **F** = (A, B, C), **x** = $\vec{x}$, **w** = $\vec{w}$

Constraint System Programming

- Directly programming algebraic constraints is **painful**.
- A number of high-level languages that compile down to constraints, but balancing performance and ergonomics is tricky

Care about	**Don't care about**
• Good performance • Ergonomic constraint writing	• Details of SNARKs, curves

Ergonomics

Karatsuba Multiplication

Native code

```
let v0 = self.c0 * &other.c0;
let v1 = self.c1 * &other.c1;

result.c1 += &self.c0;
result.c1 *= other.c0 + &other.c1;
result.c1 -= &v0;
result.c1 -= &v1;
result.c0 = v0 + v1 * P::NONRESIDUE;
```

Constraint code
in **prior libraries**

```
template<typename Fp2T>
void Fp2_mul_gadget<Fp2T>::generate_r1cs_constraints()
{
    this->pb.add_r1cs_constraint(r1cs_constraint<FieldT>(A.c1, B.c1, v1),
                                 FMT(this->annotation_prefix, " v1"));
    this->pb.add_r1cs_constraint(
      r1cs_constraint<FieldT>(A.c0, B.c0, result.c0 + v1 * (-Fp2T::non_residue)),
      FMT(this->annotation_prefix, " result.c0"));
    this->pb.add_r1cs_constraint(
      r1cs_constraint<FieldT>(A.c0 + A.c1, B.c0 + B.c1,
      result.c1 + result.c0 + v1 * (FieldT::one() - Fp2T::non_residue)),
      FMT(this->annotation_prefix, " result.c1"));
}

template<typename Fp2T>
void Fp2_mul_gadget<Fp2T>::generate_r1cs_witness()
{
    const FieldT aA = this->pb.lc_val(A.c0) * this->pb.lc_val(B.c0);
    this->pb.val(v1) = this->pb.lc_val(A.c1) * this->pb.lc_val(B.c1);
    this->pb.lc_val(result.c0) = aA + Fp2T::non_residue * this->pb.val(v1);
    this->pb.lc_val(result.c1) = (this->pb.lc_val(A.c0) + this->pb.lc_val(A.c1)) *
                                 (this->pb.lc_val(B.c0) + this->pb.lc_val(B.c1)) -
                                 aA - this->pb.lc_val(v1);
}
```

Ergonomics

Karatsuba Multiplication

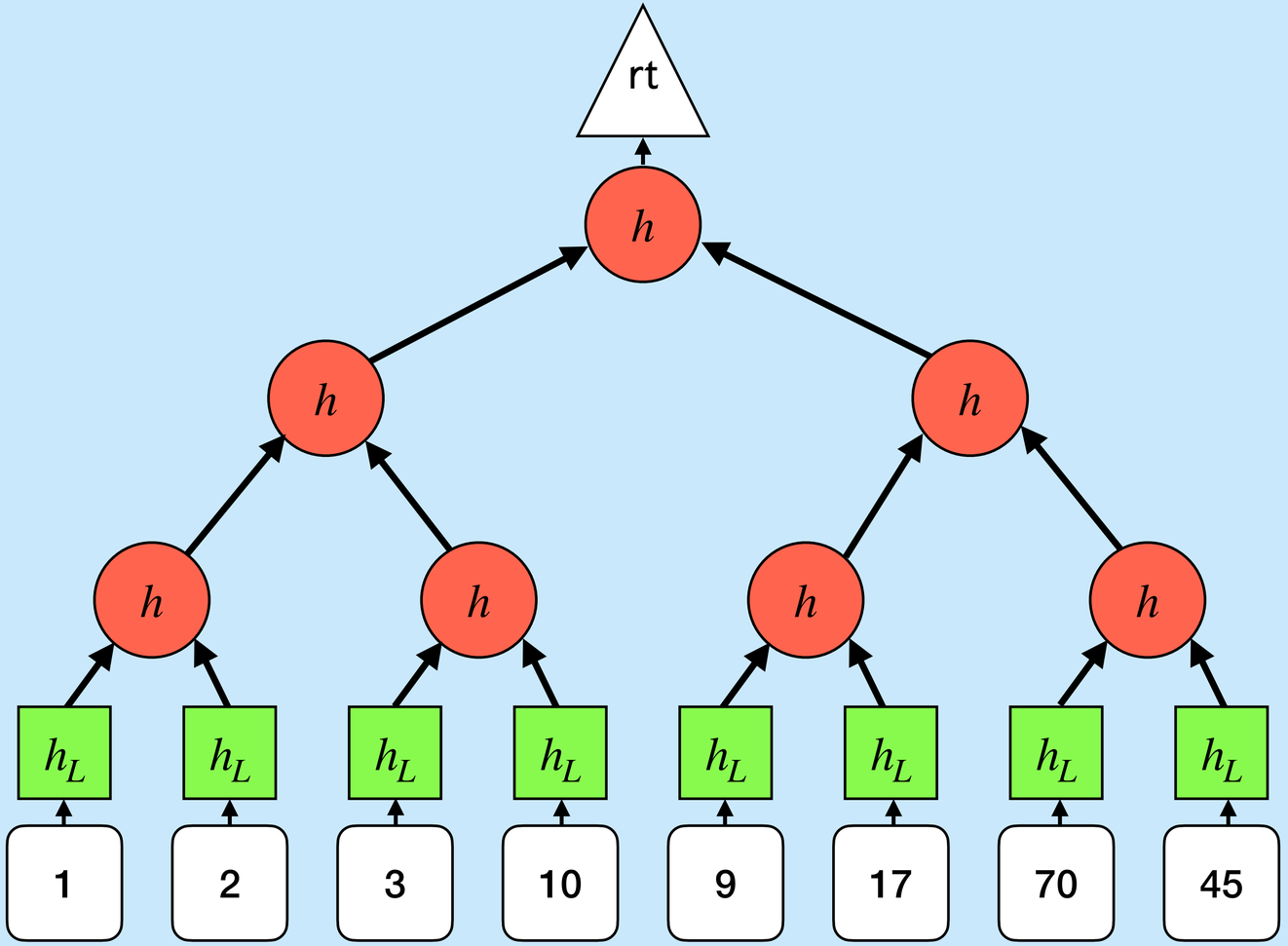
Native code

```
let v0 = self.c0 * &other.c0;
let v1 = self.c1 * &other.c1;

result.c1 += &self.c0;
result.c1 *= other.c0 + &other.c1;
result.c1 -= &v0;
result.c1 -= &v1;
result.c0 = v0 + v1 * P::NONRESIDUE;
```

Constraint code
in `arkworks`

```
let v0 = &self.c0 * &other.c0;
let v1 = &self.c1 * &other.c1;

result.c1 += &self.c0;
result.c1 *= &other.c0 + &other.c1;
result.c1 -= &v0;
result.c1 -= &v1;
result.c0 = &v0 + &v1 * P::NONRESIDUE;
```

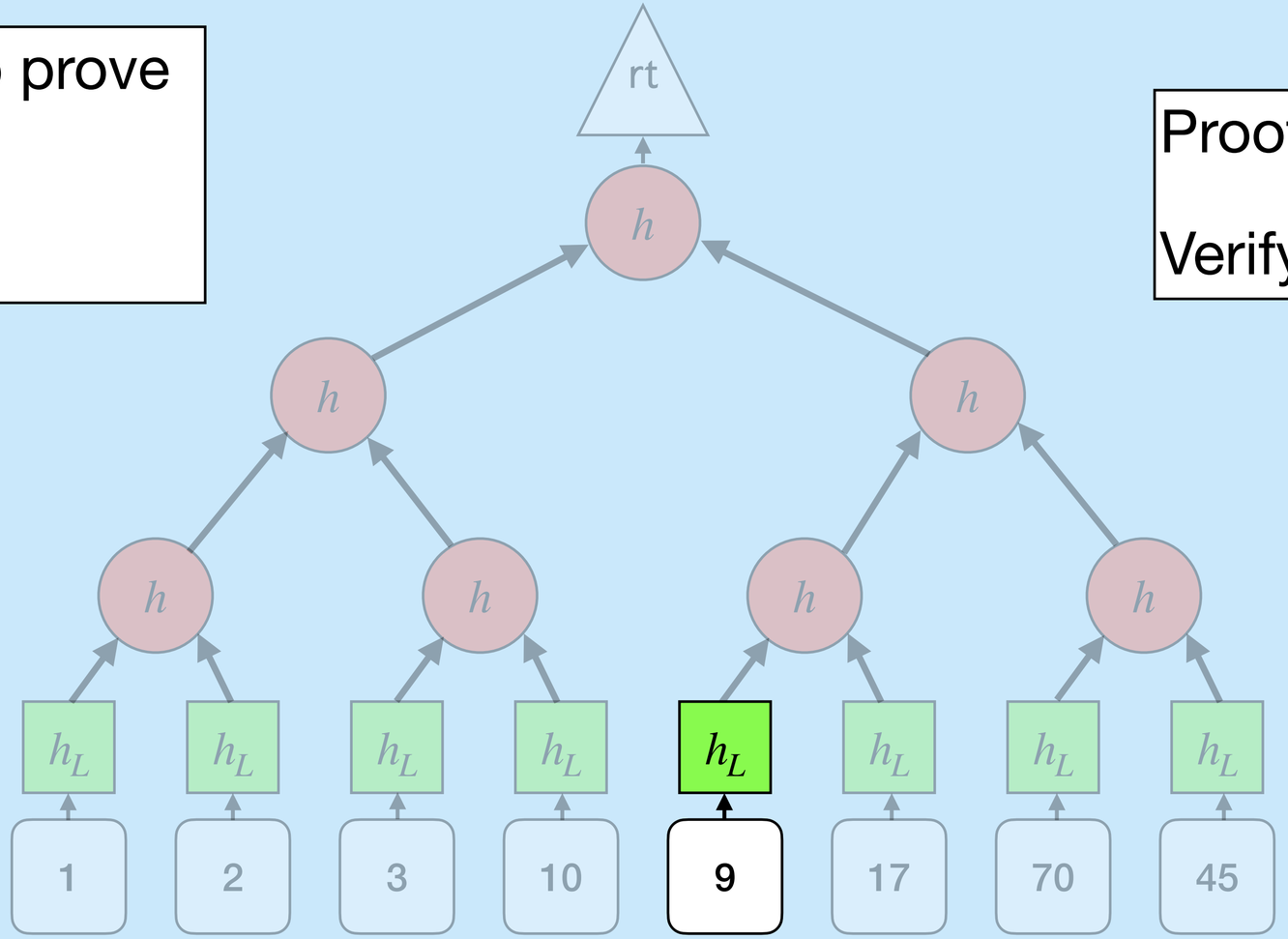
Merkle Tree



Merkle Tree

Prover wants to prove
4-th leaf is 9

Verifier has rt



Proof: h_L h h

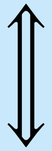
Verify: rt $=_?$ rt′

Merkle Tree in Arkworks

```
let leaf_crh_params = <LeafHash as CRH>::setup(&mut rng).unwrap();
let two_to_one_crh_params = <TwoToOneHash as TwoToOneCRH>
                            ::setup(&mut rng).unwrap();

let tree = SimpleMerkleTree::new(
           &leaf_crh_params, &two_to_one_crh_params,
           &[1u8, 2u8, 3u8, 10u8, 9u8, 17u8, 70u8, 45u8]
           ).unwrap();

let proof = tree.generate_proof(4).unwrap();
let root = tree.root();
let result = proof.verify(
             &leaf_crh_params, &two_to_one_crh_params,
             &root, &[9u8]
             ).unwrap();
```

Merkle Tree Constraint

F(**x**, **w**) = 1



x = (**rt**, **leaf**) and **w** is a proof that attests that the 4th leaf of Merkle tree rooted at **rt** has value equal to **leaf**

Merkle Tree Constraint

```
let root = RootVar::new_input(ark_relations::ns!(cs, "root_var"), || Ok(&self.root))?;
let leaf = UInt8::new_input(ark_relations::ns!(cs, "leaf_var"), || Ok(&self.leaf))?;

let leaf_crh_params = LeafHashParamsVar::new_constant(cs.clone(),&self.leaf_crh_params)?;
let two_to_one_crh_params = TwoToOneHashParamsVar::new_constant(
                            cs.clone(), &self.two_to_one_crh_params)?;

let path = SimplePathVar::new_witness(ark_relations::ns!(cs, "path_var"),
           || { Ok(self.authentication_path.as_ref().unwrap()) })?;

let leaf_bytes = vec![leaf; 1];
let is_member =
        path.verify_membership(
        &leaf_crh_params,
        &two_to_one_crh_params,
        &root,
        &leaf_bytes.as_slice(),
    )?;
is_member.enforce_equal(&Boolean::TRUE)?;
```

SNARKs

- Now we have **F**, **x**, **w**
- Do we have to implement **P**, **V**?
- `arkworks` has many built-in

On the Size of Pairing-based Non-interactive Arguments*

Jens Groth**

University College London, UK
j.groth@ucl.ac.uk

**Snarky Signatures:
Minimal Signatures of Knowledge from
Simulation-Extractable SNARKs**

Jens Groth* and Mary Maller**

University College London
{j.groth, mary.maller.15}@ucl.ac.uk

MARLIN:
Preprocessing zkSNARKs
with Universal and Updatable SRS

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Yuncong Hu
yuncong_hu@berkeley.edu
UC Berkeley

Mary Maller
mary.maller.15@ucl.ac.uk
UCL

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Psi Vesely
psi@ucsd.edu
UCL

Nicholas Ward
npward@berkeley.edu
UC Berkeley

May 27, 2020

Proofs for Inner Pairing Products and Applications

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Mary Maller
mary.maller@ethereum.org
Ethereum Foundation

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nirvan Tyagi
tyagi@cs.cornell.edu
Cornell University

Psi Vesely
psi@berkeley.edu
UC Berkeley

Gemini: Elastic SNARKs for Diverse Environments

Jonathan Bootle
jbt@zurich.ibm.com
IBM Research

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Yuncong Hu
yuncong_hu@berkeley.edu
UC Berkeley

Michele Orrù
michele.orru@berkeley.edu
UC Berkeley

Proving our Merkle Tree

```
use ark_bls12_381::Bls12_381;
use ark_groth16::Groth16;
use ark_snark::SNARK;

let mut rng = ark_std::test_rng();

let (pk, vk) = Groth16::<Bls12_381>::circuit_specific_setup(
                        defining_merkle_tree_circuit, &mut rng
                        ).unwrap();

let proof = Groth16::prove(&pk, merkle_tree_circuit, &mut rng).unwrap();

let public_input = [ root, leaf ];


let valid_proof = Groth16::verify(&vk, &public_input, &proof).unwrap();

assert!(valid_proof);
```

Modularity: switch between proof systems easily

Our generic SNARK interfaces make
switching between proof systems easy

Switch between proof systems

```
Groth16::prove(pk, circuit, rng);
```

Switch between polynomial commitments

```
Marlin::<KZG10>::prove(pk, circuit, rng);
```

Algebra

- All SNARKs require efficient **finite field arithmetic** (think $\mathbf{Z}_p$), as well as efficient **polynomial arithmetic** over these fields.
- Many SNARKs also require implementations of **extension fields** and **elliptic curves** over these fields.

Want

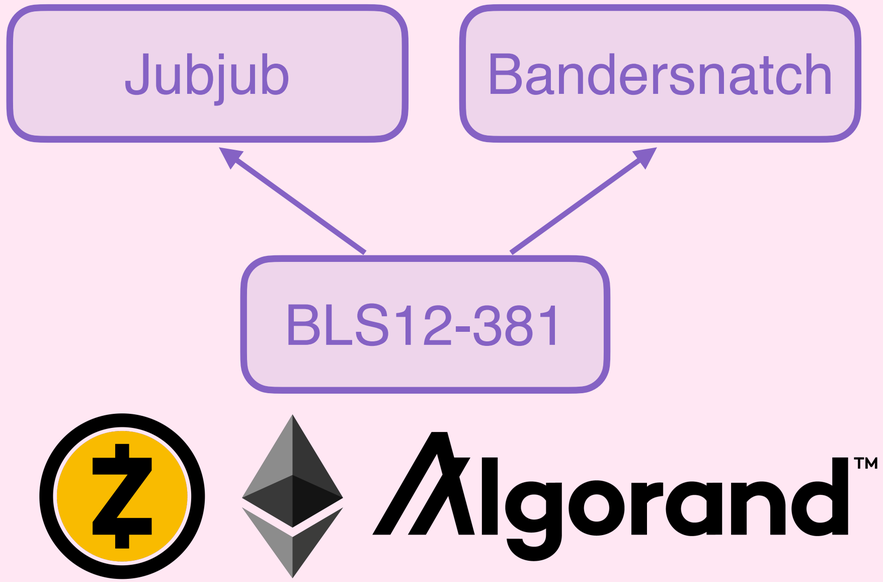
- Ease of implementing new fields/curves
- Performance like hand-optimized impl

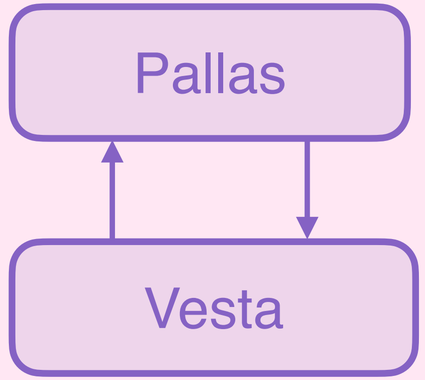
Don't want

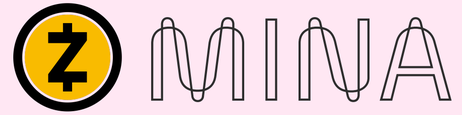
- Redundancy/duplication of code and effort

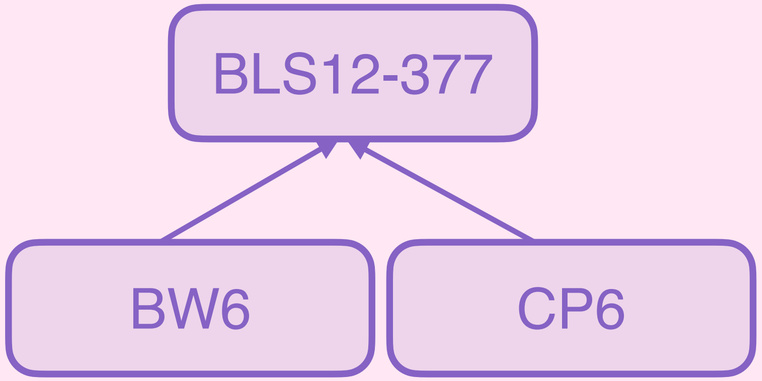
Ergonomics

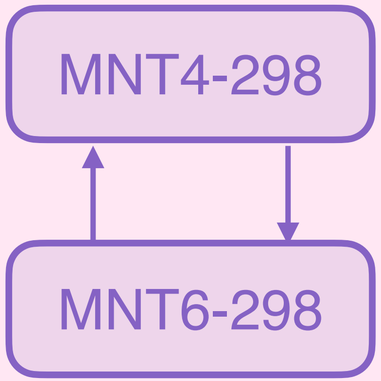
Many curves available and easy to implement new ones!

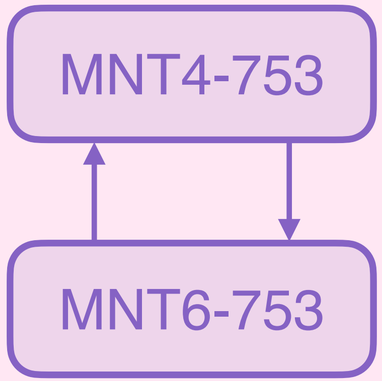












Modularity

Our generic curve interfaces allow easily switching between different curves

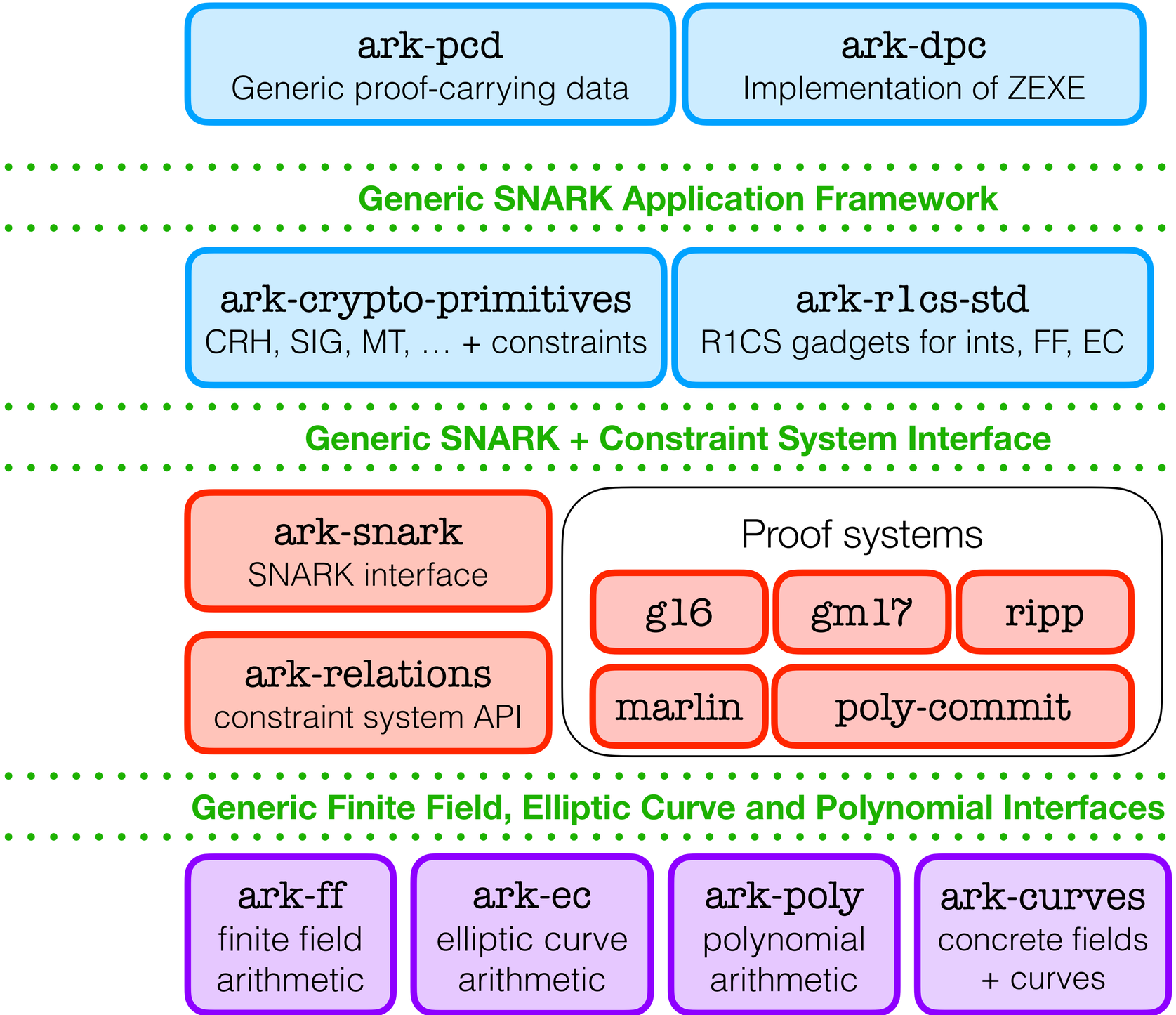
```
Groth16::<Bn254>::prove(pk, circuit, rng)
```

```
Schnorr::<Baby_Jubjub>::sign(sk, message)
```

Performance

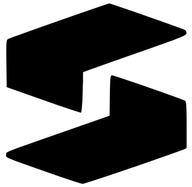
Our generic implementations match or outperform specialized and hand-coded impls

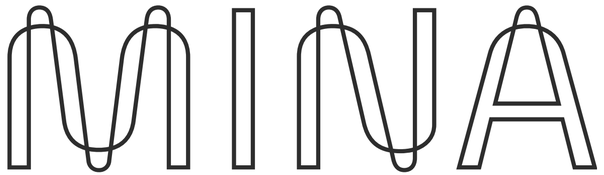
BLS12-381	**Bellman**	`arkworks`	**MCL (dcg)**
F_q Mul	56 ns/op	31 ns/op	44 ns/op
G_1 Add	963 ns/op	692 ns/op	504 ns/op
G_1 Dbl	483 ns/op	381 ns/op	360 ns/op
Pairing	2 ms/op	1.7 ms/op	0.7 ms/op

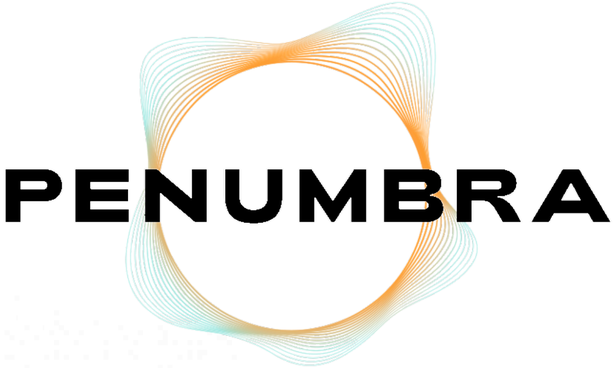

ark-pcd
Generic proof-carrying data
ark-dpc
Implementation of ZEXE
Generic SNARK Application Framework
ark-crypto-primitives
CRH, SIG, MT, … + constraints
ark-r1cs-std
R1CS gadgets for ints, FF, EC
Generic SNARK + Constraint System Interface
ark-snark
SNARK interface
ark-relations
constraint system API
Proof systems
g16
gm17
ripp
marlin
poly-commit
Generic Finite Field, Elliptic Curve and Polynomial Interfaces
ark-ff
finite field arithmetic
ark-ec
elliptic curve arithmetic
ark-poly
polynomial arithmetic
ark-curves
concrete fields + curves

Has any of this been useful?

Impact







Webb Tools

web3 foundation

And 15+ academic papers!

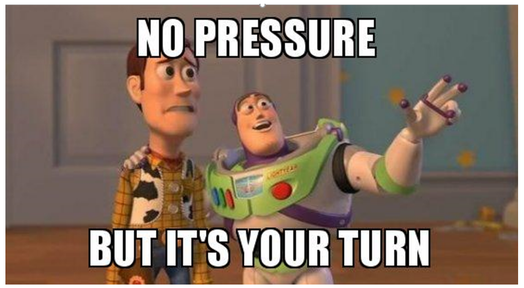
Key part of academic papers

2019
Aurora
[BCR+19]

2020
Marlin
[CHM+20]

2021
Incrementally Verifiable Computation
[BCM+20, BCL+21]

2022
Gemini
[BCH+22]



Thanks to all the contributors!

@lightyear15
@popog
@hrodr
@adria0
@3for
@clearloop
@drskalman
Pascal Berrang
Sean Bowe
Jeff Burdges
Weikeng Chen
Xu Cheng
Alessandro Chiesa
Shumo Chu
Jon Chuang
Sunhua Chuang
Emma Dauterman
Bruno França
Nicolas Gailly
François Garillot
Ralph Giles
George Gkitsas
Brandon H. Gomes
Marcin Gorny
Kobi Gurkan
Daira Hopwood
Youssef El Housni
Yuncong Hu
Solomon Joseph
Georgios Konstantopoulos
Marek Kotewicz
Ryan Kung
Ying Tong Lai
Hyemin Lee
Ryan Lehmkuhl
Chenxing Li
William Lin
Victor Lopes
Simon Masson
Pratyush Mishra
Dev Ojha
Michele Orrù
Alex Ozdemir
Horace Pan
Carlos Pérez
Antoine Rondolet
Michael Rosenberg
Thibaut Schaeffer
Tom Shen
Sam Steffen
Drew Stone
Nirvan Tyagi
Henry de Valence
Ashutosh Varma
Sergey Vasilyev
Psi Vesely
Nicholas Ward
Kevaundray Wedderburn
David Wong
Alexander Wu
Howard Wu
Alex Xiong
Zhenfei Zhang

Come Join Us

- Did you like what you saw?
- We have open positions
  - Cryptography Engineer
  - Semester Projects & Master Thesis

Come talk to me or drop me an email!

Thanks for your attention!

Any non-pizza related questions?

*To get involved with **arkworks**:*

arkworks-rs

arkworks-rs

discord.gg/D9GjcxPt4R