Hunting for the SNARK - February

A learning group for ZK and SNARK application development

Daniel Szego
In: https://www.linkedin.com/in/daniel-szego/

Logistics: Hunting for the SNARK

Every month, thirds thursday in 2025, from 18 (CET)

One hour, presentation + short discussion

Different topics on zero knowledge proof,

- mostly from programmer and application developers perspective
- with some theory

Coordination:

- Discord channel: LF Decentralized Trust https://discord.com/channels/905194001349627914/1329201532628898036
- Meetup.com: https://www.meetup.com/lfdt-hungary/events/305634614/
- Repo with all the contents:https://github.com/LF-Decentralized-Trust-labs/

Quizzes and small programming challenges, LFDT merchs at the end

Logistics: Hunting for the SNARK

February - Introduction, Theory : Definitions and building blocks

March - Theory : Polynomial commitments

April - Theory : Interactive oracle proofs

May - Programming : Circom

June - Programming : Circom

July - Programming : Noir

August - Programming : Noir

September : Applications : Off-chain transaction

October : Applications : Proving solvency

November : Applications : Rollup

December : Wrap up, Applications

Subject to change based on community discussion ….

Agenda

- *Basic ideas of zero knowledge proofs*
- *Computational models*
- *Arithmetic circuits and R1CS*
- *Definitions and properties of*
  - *NARK,*
  - *SNARK*
  - *zk-SNARK*
- *Comparisons of algorithms*
- *High level constriction*
- *Q&A*

Prover - verifier model

"Proof" of a statement, e.g. I know a preimage of a hash function

It's not a "classic" mathematical proof, it's stochastic, I know with high probability

I know some kind of secret information, I "prove" that I know without saying it

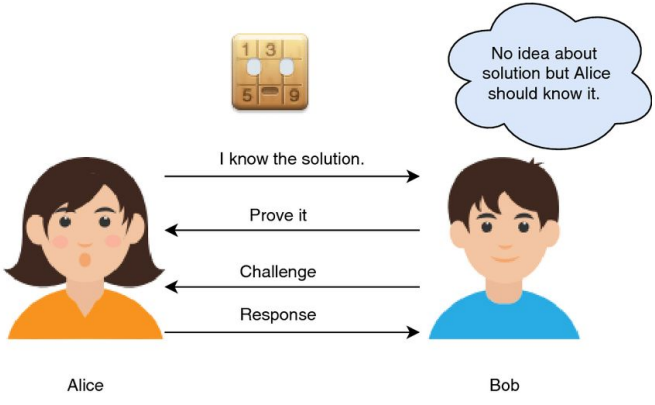
Roles:

- Prover: prover

- Verifier: verifier, validator

Interactive / non-interactive

Proof size, prover / verifier time



Example - Cave of Ali Baba

Interactive probabilistic proof

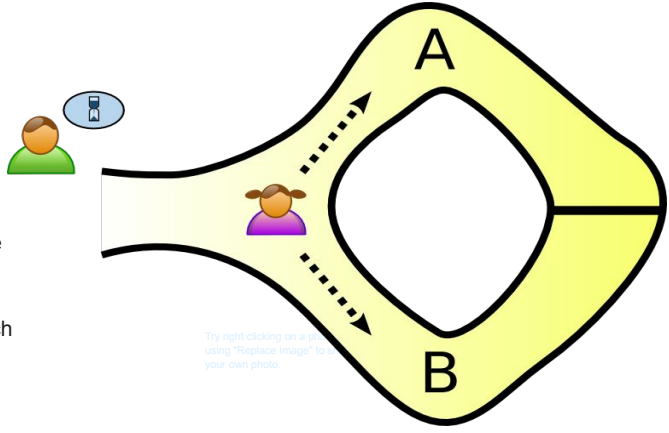
Alice and Bob

Alibaba's cave, which is closed in the middle with a door that opens with a "magic word".

Alice enters the cave and chooses a side to enter.

Bob goes into the cave and decides which side Alice should come back from.

He can cheat 50% of the time -> when repeated many times, he approaches 0%



Computational model

Calculating a function as computation or business logic

Evaluating a computation as a true / false predicate

High level language for abstraction (domain specific)

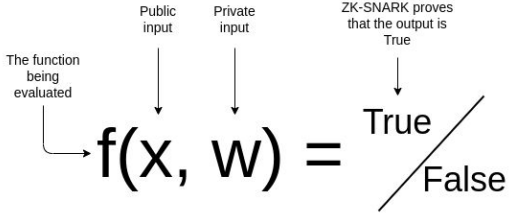
Compilation to lower level

Lower level mathematical representation:

- Arithmetic circuits
- R1CS, rank one constraints
- Arithmetization with lookup tables (e.g. Plonk)

Mathematical constructs, e.g.Polynomials on discrete fields and hash functions

Different properties of polynomials, e.g. roots, equations



Arithmetic circuit

Defined over an *F* finite field of a *p* prime

DAG: directed acyclic graphs

Inputs, gates, constants, connections (wires)

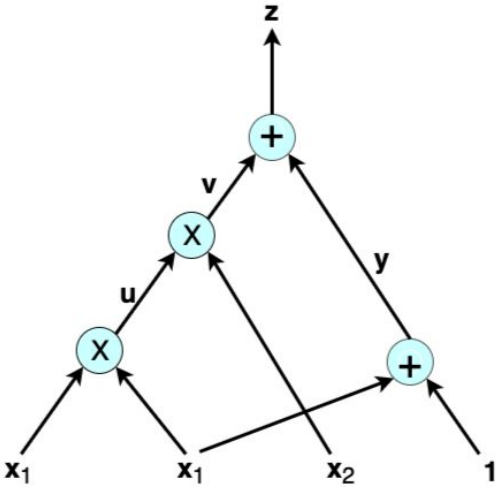
Nodes are labelled with "+" addition and "*x*" multiplication

Inputs are *1, x1, x2, ….. xN*.

Public and private inputs

Defined an n variate polynomial with evaluation recipe

E.g. *C* arithmetic circuit for proving the preimage of a hash function: $C_{SHA}(h, m)$: outputs 0 if $SHA_{256}(m) = h$, and $\neq 0$ otherwise



R1CS - Rank one constraints

x: field elements of x_1 …. x_l

w: w_1, ….w_{m-l-1}

Equations in the form of:

$a \times b = g$

a,b and g are affine combination of variables

Examples:

$w_1 \times (w_2 - w_3 + 1) = x_1$

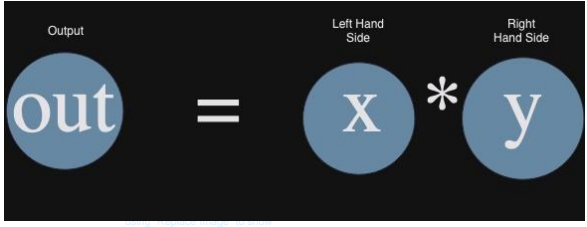
$w_2 \times w_2 = w_2$

wrong: $w_2 \times w_2 \times w_2 = x_1$

multiply equations : constraint system

convertible to arithmetic circuits

matrix representation



ZK engineering flow

Engineering flow:
- High level language
- Low level abstraction
- Polynomials

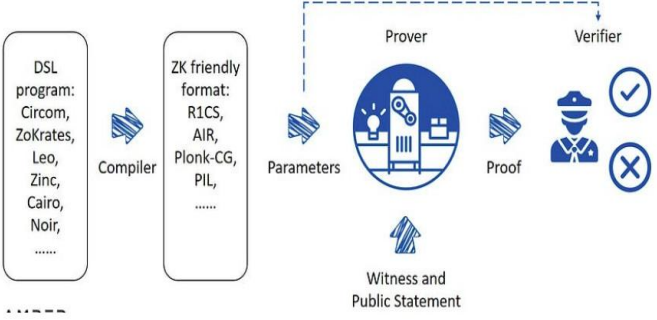
Prove and verify

Complex development frameworks, compile, test, prover, verifier module integrations.

Imperative / description / circuit languag

Different base programming language

Different programming language and framework integration modules



NARK - Non-interactive ARgument of Knowledge

Preprocessing the computational model

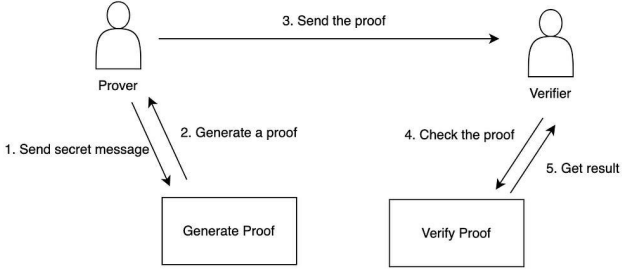
Arithmetic circuit (computation): *C(x, w) -> F*
- *x* public input
- *w* private input, witness

Algorithms

Preprocessing *S* algorithm the circuit for producing *pp* public and *vp* private parameters *S(C) -> pp* and pv parameters

Prover algorithm: *P (pp, x, ww) -> proof*

Verifier algorithm: *V (vp, x, proof) -> accept / reject*

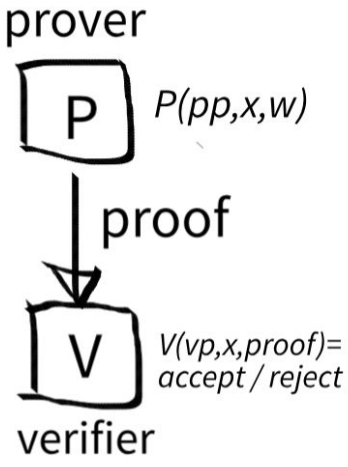


NARK Properties

Complete:
For each input pairs and circuit, if *C (x, w) = 0*, then the generated proof is accepted by the prover with very high probability: *Pr [V(vp, x, P (pp, , x, w)) = accept] = 1*

Knowledge sound:
If the verifier accepts, then the prover “knows” a *w* witness (secret input), such that *C (x, w) = 0*
(Extractor model)

Zero knowledge (optional):
The *C* circuit, the *x* public input, the *vp* and *pp* public parameters and the proof reveal nothing about the *w* witness (private input)



(zk)SNARK

SNARK = S and NARK

Succinct:
- Proof size is small, sublinear of the w witness
- Proving time is short, sublinear of the size of *C circuit*

Strongly succinct:
- Proof size is smaller, logarithmic in the size of C circuit
- Proving time is faster, logarithmic in the size of C circuit

(The verifier has no time to read the circuit)

zkSNARK, a SNARK that is also zero knowledge

Core algorithm consideration

Under very active development

Proof size

Verification time

Proving time matters as well

Setup :
- per circuit
- universal
- transparent

Post quantum readiness

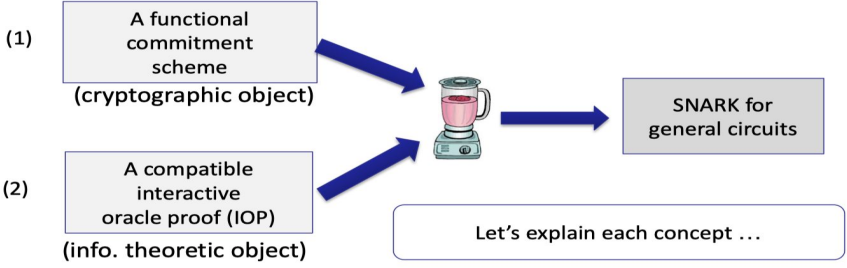
	size of proof π	verifier time	setup	post-quantum?
Groth'16	$\approx$ 200 Bytes $O_\lambda(1)$	$\approx$ 1.5 ms $O_\lambda(1)$	trusted per circuit	no
Plonk / Marlin	$\approx$ 400 Bytes $O_\lambda(1)$	$\approx$ 3 ms $O_\lambda(1)$	universal trusted setup	no
Bulletproofs	$\approx$ 1.5 KB $O_\lambda(\log \lvert C \rvert)$	$\approx$ 3 sec $O_\lambda(\lvert C \rvert)$	transparent	no
STARK	$\approx$ 100 KB $O_\lambda(\log^2 \lvert C \rvert)$	$\approx$ 10 ms $O_\lambda(\log^2 \lvert C \rvert)$	transparent	yes

Elements of building a SNARK

Polynomial commitment scheme: get succinct interactive argument: Bind and commit a polynomial.

Interactive Oracle Proof (IOP): R1CS or circuit satisfiability. Evaluating (committed) polynomials in one or multiply points in multiply rounds, extend polynomial commitment to general satisfiability.

Fiat-Shamir: transforming a interactive proofs to non-interactive:



Zoo of SNARK

Construction

Several polynomial IOP

Several polynomial commitments

Mixing up components:
- time (verification, proof)
- proof size
- setup assumptions
- post quantum

Polynomial IOP

Interactive proof:
- vSQL, Libra

Multi-prover interactive proofs:
- Spartan

constant -round polynomial IOPs:
- Marlin, Plonk

.

Polynomial commitment

Pairing:
- KZG10

Discrete logarithm:
- Bulletproof

Hashing:
- FRI

Groups of unknown order:
- DARK

Challenge

The Alibaba cave example is an interactive probabilistic proof.

Can you make it non-interactive or at least less interactive in a way that it can not be hacked / mocked by the prover ?

Conditions of the game can be lightly fine-tuned.

Links, Resources, Literature

Zero-Knowledge Proofs: A Beginner's Guide
https://www.dock.io/post/zero-knowledge-proofs

Introduction to zero knowledge proofs
https://www.youtube.com/watch?v=6uGimDYZPMw

Zero Knowledge What? An Introduction to Zero Knowledge
https://codethechange.stanford.edu/guides/guide_zk.html

Zero knowledge proof explained
https://medium.com/coinmonks/zero-knowledge-proof-explained-1595600ff1cf

zk-SNARKs: A Gentle Introduction
https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf

Happy Hunting for the SNARK :)

Q & A

Daniel Szego
In: https://www.linkedin.com/in/daniel-szego/