Zero knowledge proof and SNARK programming workshop

Daniel Szego

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

Tartalom

- Zero knowledge proofs
- Example - Waldo, cave of Ali Baba
- SNARK / zkSNARK
- Applications
- Conclusions
- ZK programming
-

Zero knowledge proof

"Proof" of a statement

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

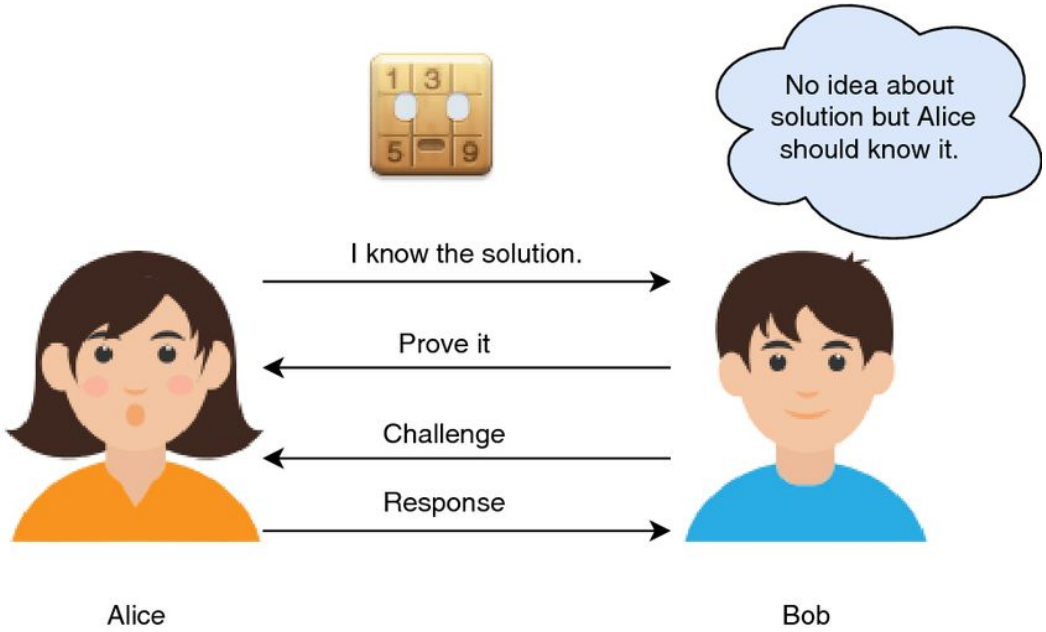
I know some kind of secret information, I "prove" that I know without saying it

Roles:

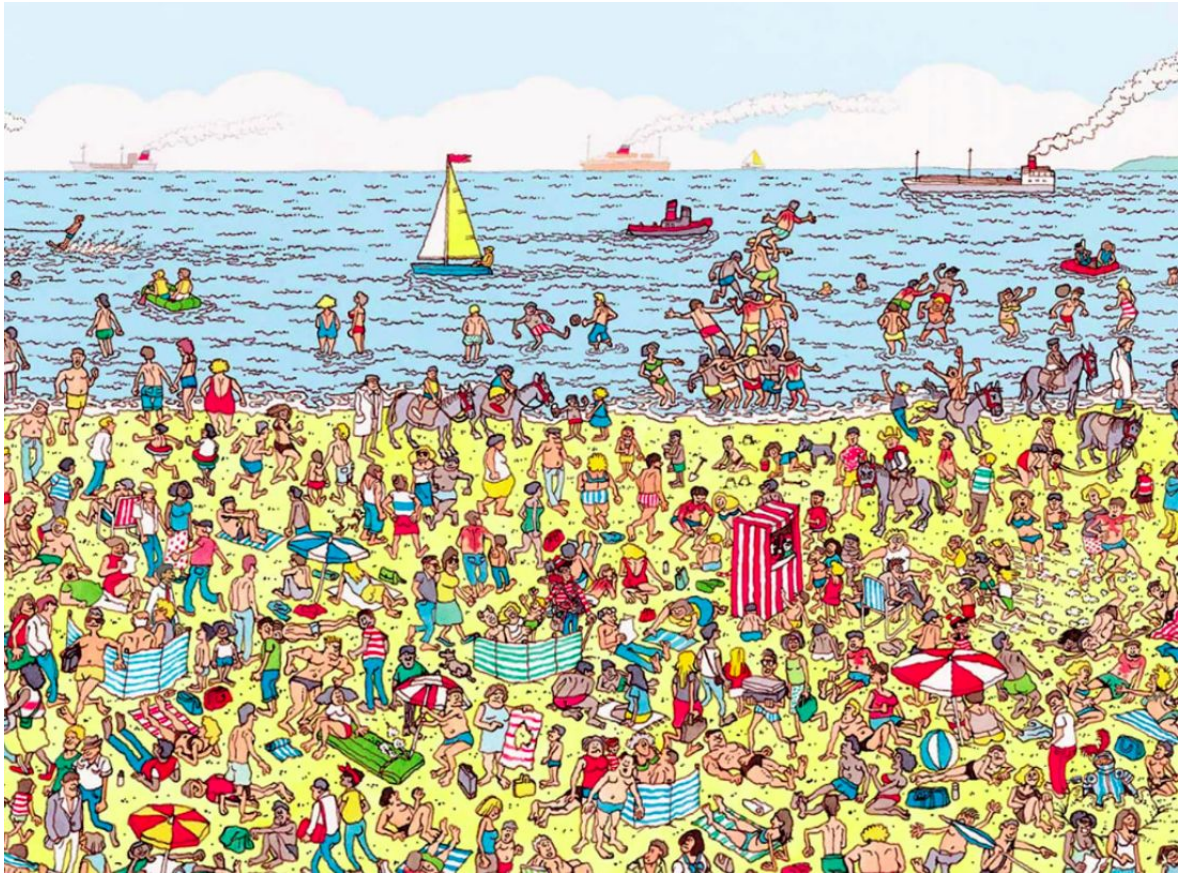
- Prover: prover

- Verifier: verifier, validator

Interactive / non-interactive



Example - Waldo



Example - Cave of Ali Baba

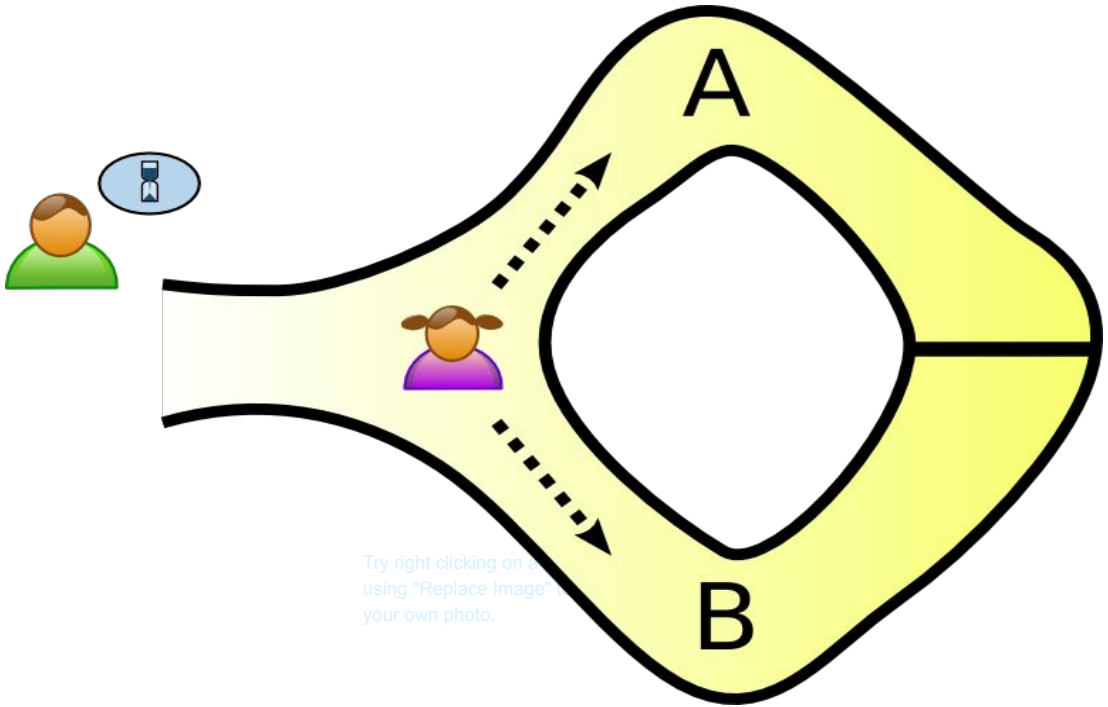
Alice and Bob

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%



SNARK / zkSNARK

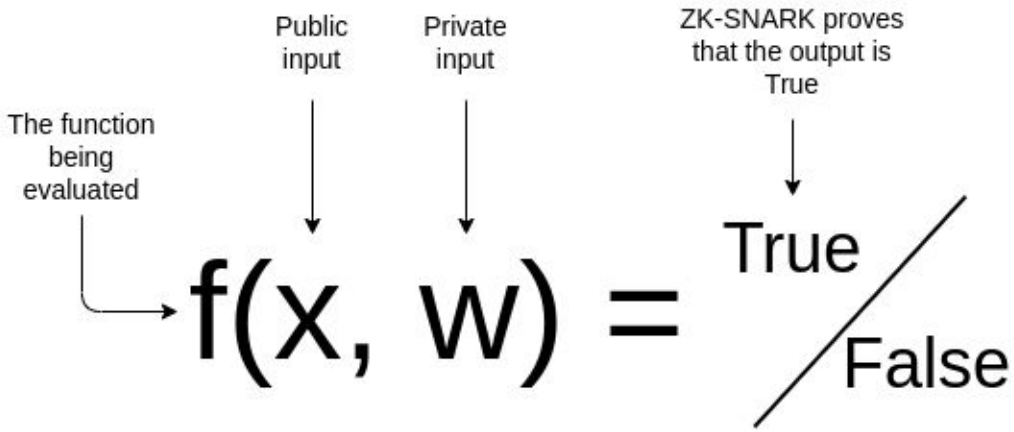
(zk) SNARK

Succinct: short, concise proof

Non-Interactive: there is no interaction, the prover produces it and sends it to the verifier.

Argument of Knowledge: Some information that the prover knows.

Zero-Knowledge: None of the private information reaches the validator.



Application - Authenticated but edited photos

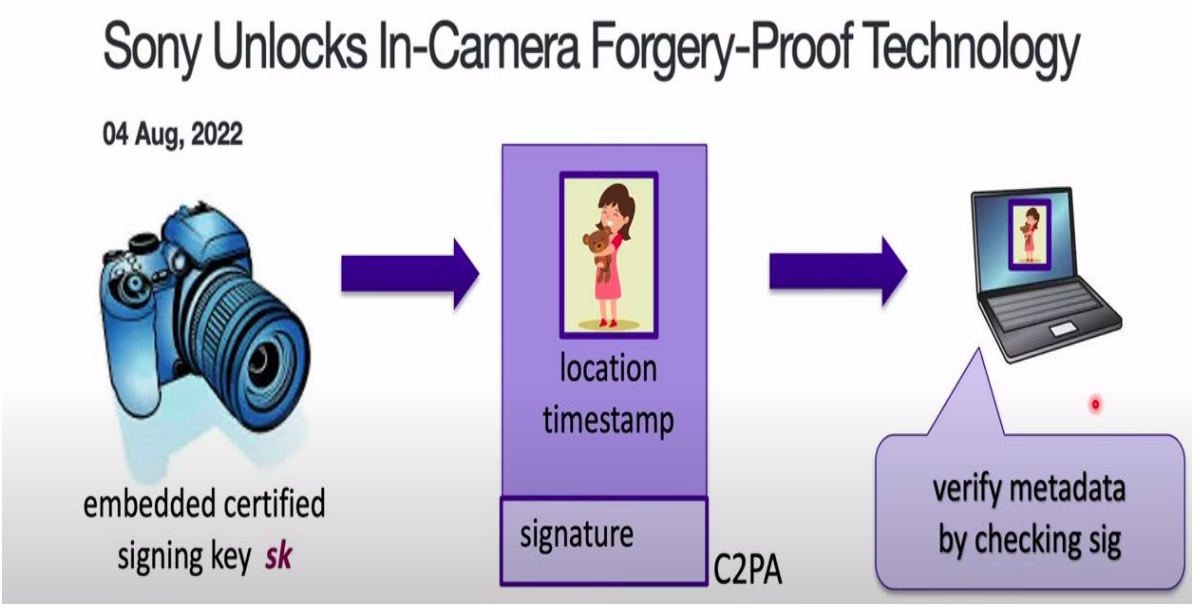
Fake pictures, articles, information online.

Camera with digital signature, location and time information.

Image transformation, resizing, cropping, sharpening, changing the color palette, etc.

The digital signature is not valid after transformation

Digital signature + transformation + SNARK



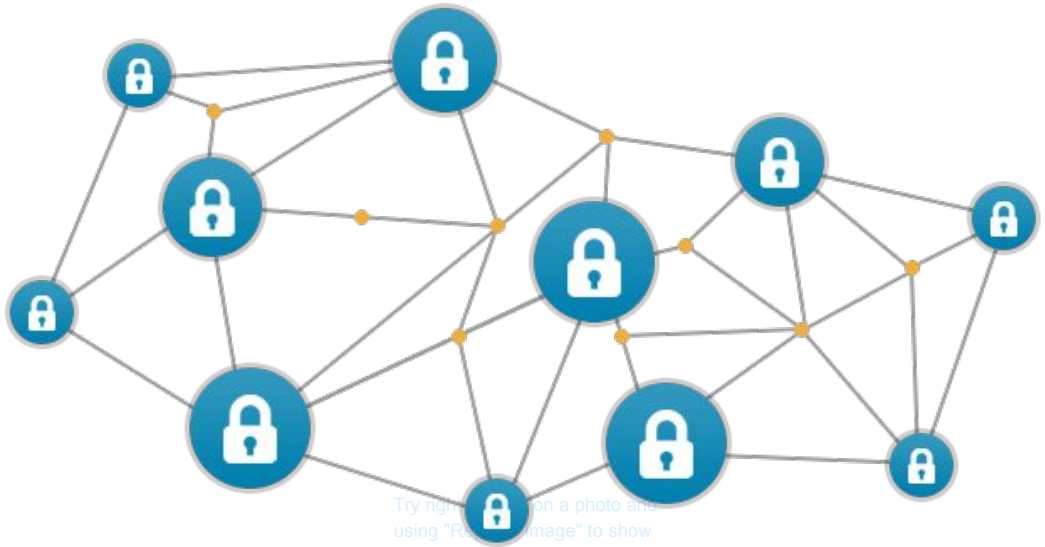
Application - Private blockchain transaction

Blockchain: independent nodes that authenticate transactions.

To authenticate, the node must see the transaction.

Privacy issues.

Instead of full transactions, transactions with secret input and zkSNARK proof.



Application - Rollup

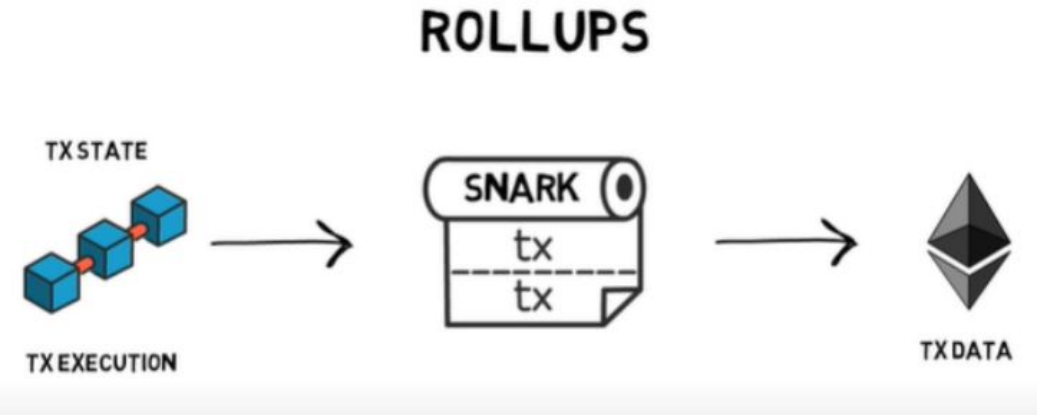
Scalability problem: Most blockchains can perform few transactions and that too slowly.

Reason: independent validators.

Batch execution of transactions outside the blockchain..

The problem is reliability, correct execution, exclusion of fraud.

SNARK / zkSNARK proof of correct execution



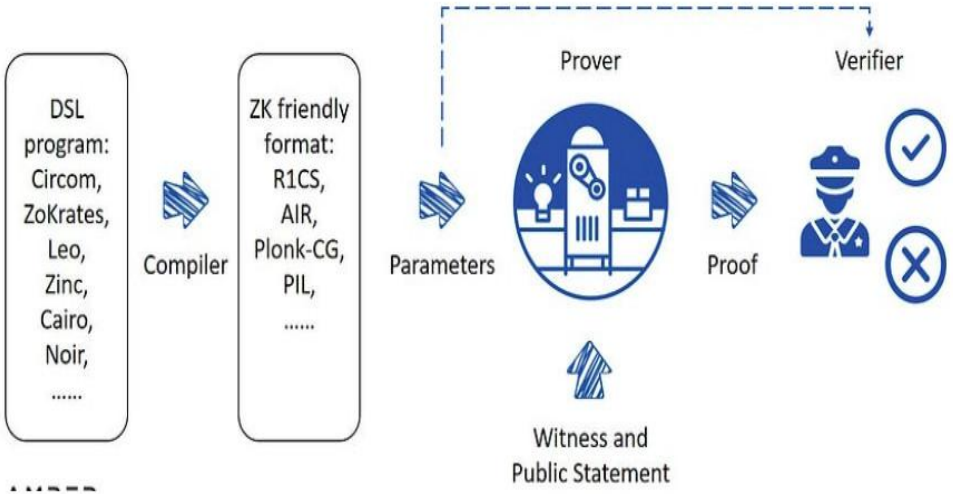
ZKP programming considerations

It is an area that requires serious theoretical preparation.

Especially mathematicians.

DSL (domain specific language) simplified, domain specific languages, simpler (!) for zero knowledge and SNARK programming.

However, simplified programming and cryptographic basics are still necessary.



Conclusions

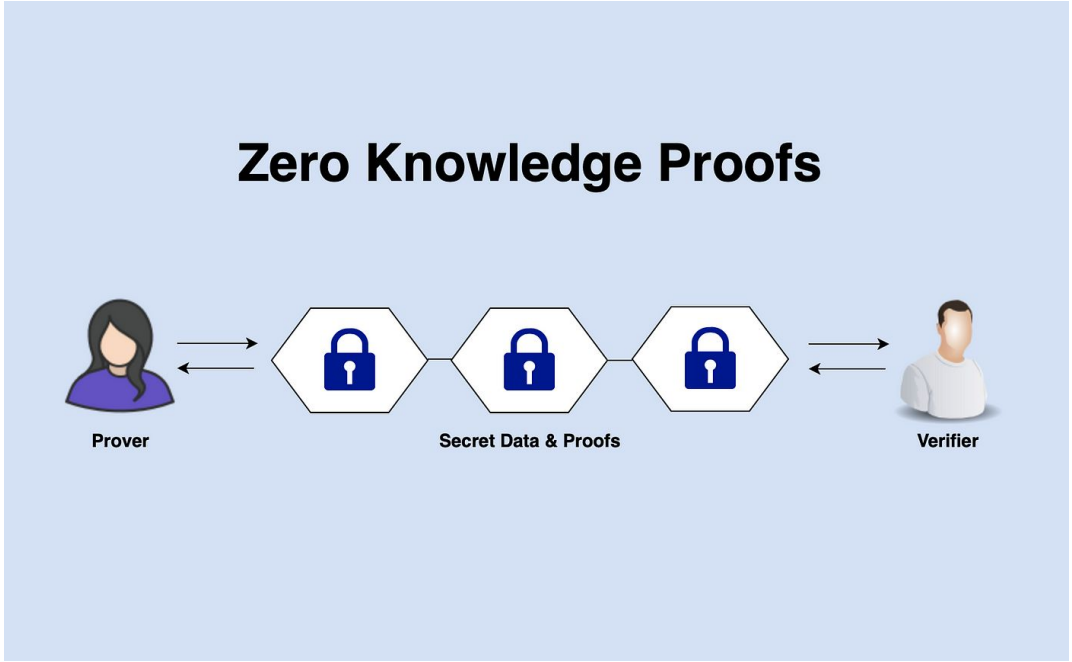
An exciting emerging field,

Dynamic development both on the research - theoretical side and on the application side.

Min. 15 years of active development.

Tools and frameworks are also available on the engineering / programming side.

The area can be at least as important as the digital signature.




Zero knowledge
programming

ZK programming - engineering flow

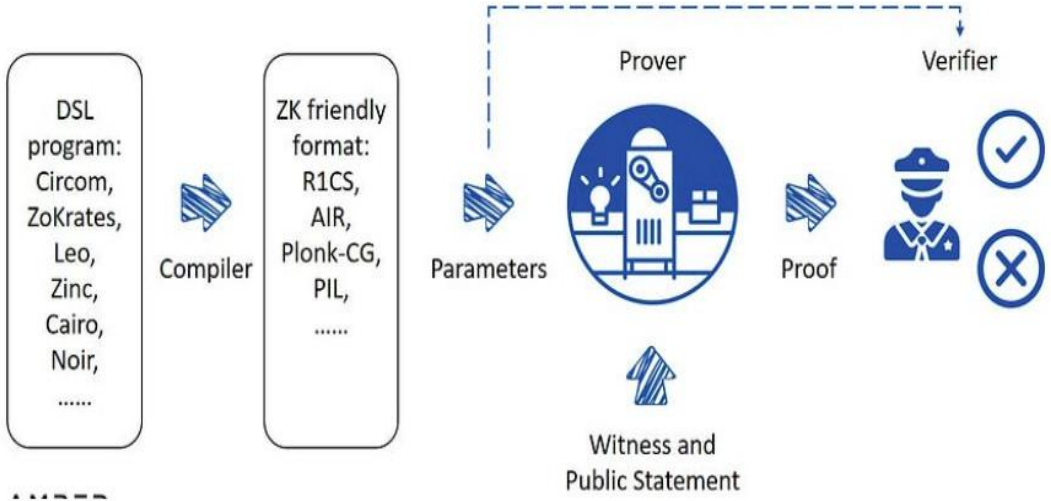
Domain specific languages for SNARK or *zk*SNARK programming

Abstracting away some of the mathematical and theoretical complexity

ZK and SNARK programming without cryptographic knowledge ? Not yet :)

Compilation to mathematical representation, R1CS

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



Core algorithm consideration

Under very active development

Proof size

Verification time

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

DSL language and tool selection

Core algorithm consideration

Imperative / description / circuit languages

Different base programming language

Different programming language and framework integration modules

Technological life cycle: all are early stage, but:

- **Successful productive usage**
- **Stable releases**

Language	Team
Noir	Aztec
SnarkyJS	O(1) Labs
Leo	Aleo
Circom	iden3
Cairo	STARKWARE
Lurk	Lurk Lab

Circom

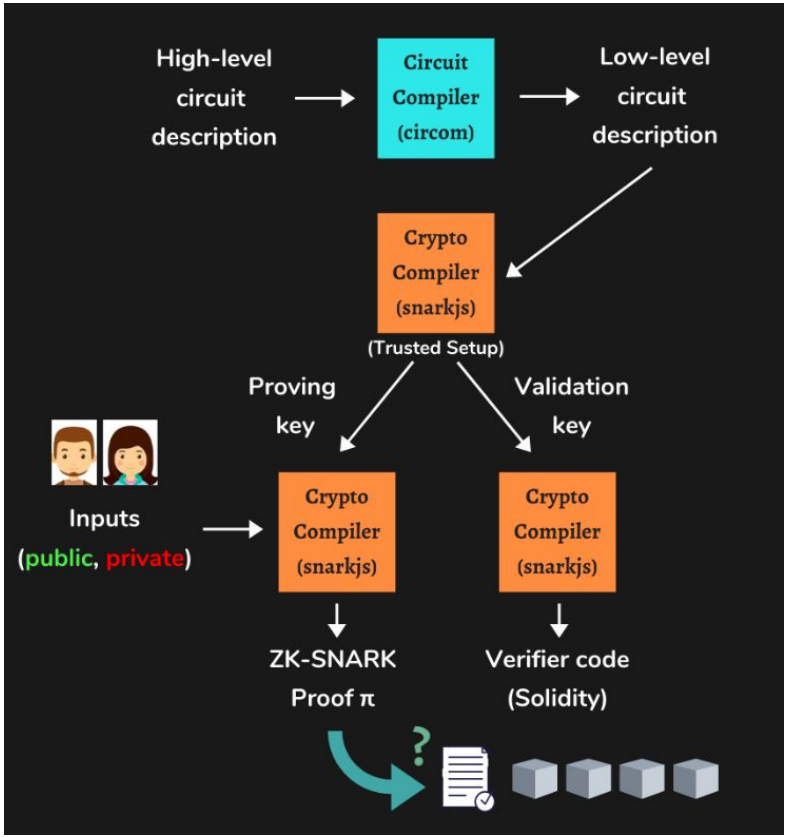
DLS / circuit programming language and development environment for arithmetic circuits and constraints

Used e.g. in tornado cash

Supports:
- **Groth16**
- **Plonk**

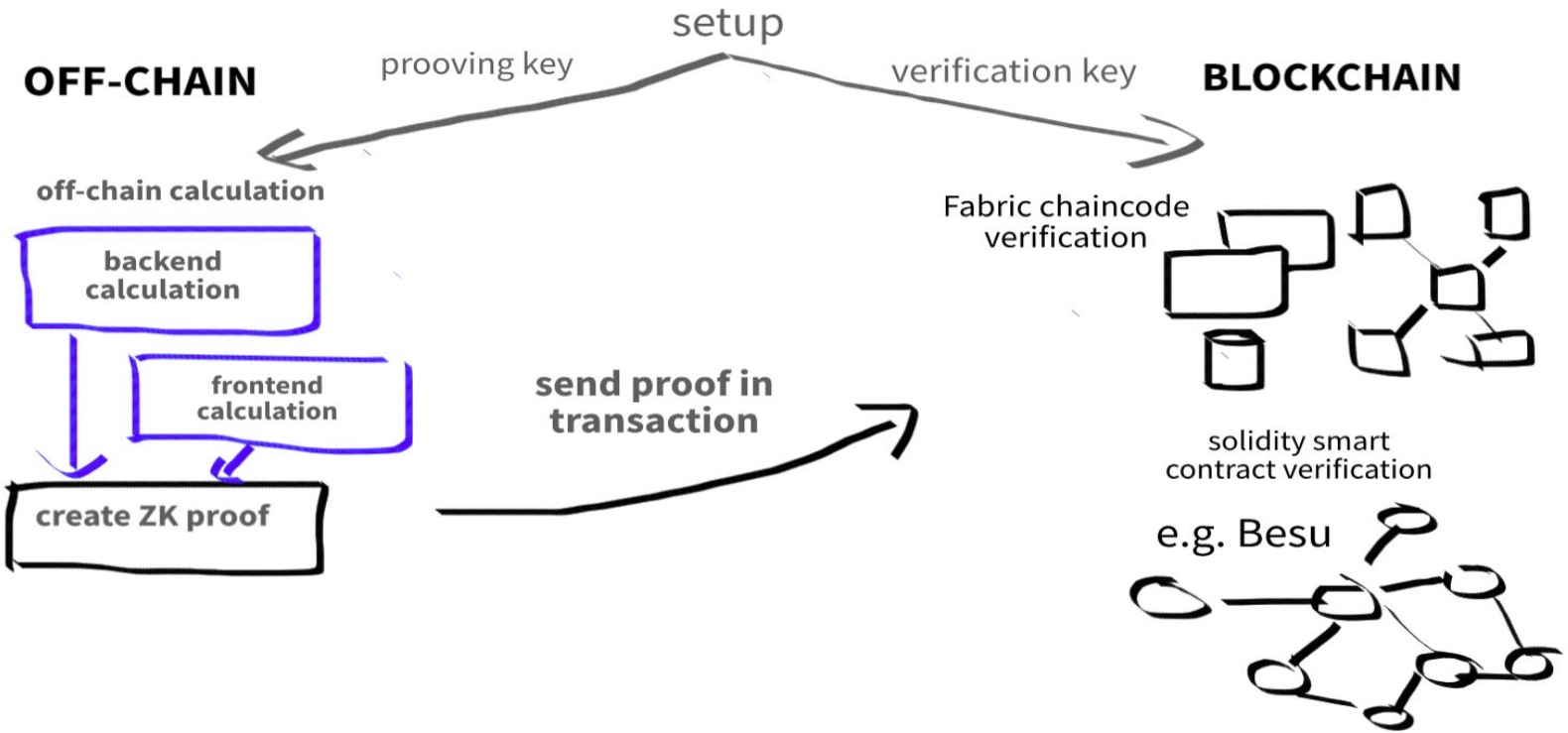
Well established (exist 3 years :)

Supported integration:
- **javascript (snarkjs)**
- **cpp**
- **solidity verifier**



https://docs.circom.io/

Architectures - demo setups



Power of Tau

Try right clicking on a photo and
using "Replace Image" to show
your own photo.

Circom language

Rank 1 constraint / circuit programming language:

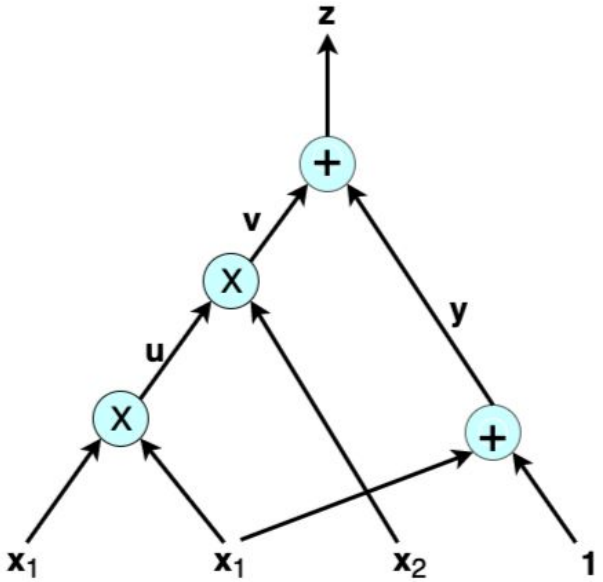
- arithmetic circuits and rank 1 constraints
- alpha x beta = gamma : affine combination of variables

Circuit / Hardware description language (HDL):

- Wires
- Signals
- Gates
- Sub-circuits
- Connecting, wiring circuits, gates

Templates and template programming for subcircuits:

- Programming elements for creating modules, signals
- Variables, control structure over templates and signals
- Templates for higher abstraction
- The output must be always a fix number of circuits, modules



Circom language

Rank 1 constraint / circuit programming language:

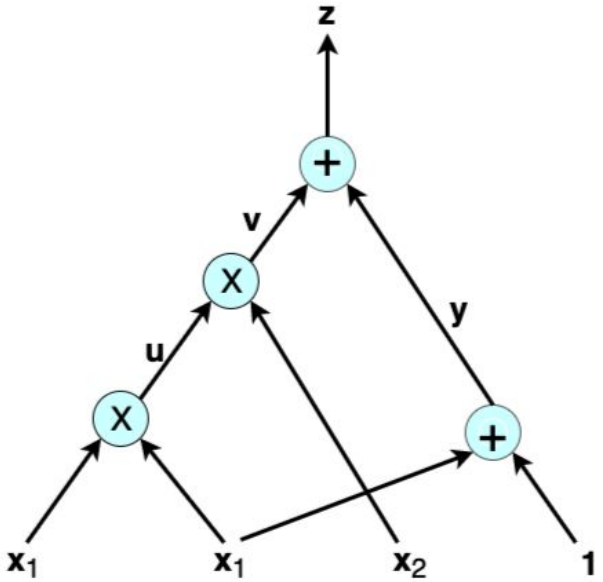
- **z <-- x * y : assigning variable to a signal**
- **Assignment is more general - any expression**
- **z === x * y : rank 1 constraint among the signals**
- **z <== x * y : both assignments and constraints**

Circuit / Hardware description language (HDL):

- **component**
- **component main {} : public vs private signals**

Metaprogramming:

- **Template argument : fix at compiler time !**
- **Array of signals (fix)**
- **For loops over array of signals**
- **fix iteration evaluated at compiler time**
- **variables : used onty at compilation to generate the circuit**
- **Embedding into another circuit**



Q & A

Daniel Szego

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