Zero Knowledge Proofs

Quote from Edward Snowden

“Zero-knowledge proofs may be the future of private trade.”

Quote from Remco Bloemen remco@0x.org (mailto:remco@0x.org)

> ### Disclaimer: contains maths
>
> If you don’t understand something
>
> - Not your fault, this stuff is hard
> - Nobody understands it fully
>
> If you don’t understand anything
>
> - My fault, anything can be explained at some level
>
> If you do understand everything
>
> - Collect your Turing Award & Fields Medal

Introduction

It is difficult to find zero knowledge resources that avoid the extremes of either over simplyfying the subject, or presenting so much mathematical detail that the reader gets bogged down and loses interest.
The purpose of this tutorial then is to find an accessible but informative middle ground.

We start with some examples to show how zero knowledge proofs can proceed, and the situations where they could be used.

Actors in a Zero Knowledge Proof

- Creator - optional, maybe combined with the prover
- Prover - I will call her Peggy
- Verifier - I will call him Victor

Examples to give an Intuitive grasp of zero-knowledge proofs

1. Wheres Wally
   Based on the pictures of crowds where Wally is distinctivly dressed, the aim being to find him amoungst a sea of similar people.
   The proof procedes as follows :
   Imagine the Peggy has found Wally in the pricture and wants to prove this fact to Victor, however if she just shows him, Victor is liable to cheat and claim he also found Wally. In order to prove to Victor that she has indeed found Wally, without giving away his location in the picture

   1. Peggy cuts a hole in a (very) large sheet of paper, the hole should be the exact shape of Wally in the underlying picture.
   2. Peggy places the paper sheet over the original picture, so that the location of the picture beneath the paper is obscured.
   3. Victor can then see throught he hole that Wally has indeed been found, but since the alignment with the underlying picture cannot be seen, he doesn't gain any information about the location of Wally.

2. Colour blind verifier
   This is an interactive proof showing that the prover can distinguish between a red and a green billiard ball, whereas the verifier cannot distinguish them.

   • The prover wants to show the verifier that they have different colours but does not want him to learn which is red and which is green.

   • Step 1: The verifier takes the balls, each one in each hand, holds them in front of the prover and then hides them behind his back. Then, with probability 1/2 either swaps them (at most once) or keeps them as they are. Finally, he brings them out in front.

   • Step 2: The prover has to say the verifier switched them or not.

   • Step 3: Since they have different colours, the prover can always say whether they were switched or not.
   But, if they were identical (the verifier is inclined to believe that), the prover would be wrong with probability 1/2.

   • Finally, to convince the verifier with very high probability, the prover could repeat Step 1 to Step 3 k times to reduce the probability of the prover being successful by chance to a extremely small amount.

3. Sudoku
   An interactive proof can be created to prove the knowledge of a solution to a sudoku puzzle by placing cards in the sudoku grid. The process is descrived here Sudoku Proof (https://blog.computationalcomplexity.org/2006/08/zero-knowledge-sudoku.html)

zk-SNARKS

Now that we an intuitive grasp of zero knowledge proofs, lets go into the details by looking at a certain type of proof system, a zk-SNARK.

The process of creating and using a zk-SNARK can be summarised as

A zk-SNARK consists of three algorithms C, P, V defined as follows:

The Creator takes a secret parameter lambda and a program C, and generates two publicly available keys:

- a proving key pk
- a verification key vk

These keys are public parameters that only need to be generated once for a given program C.

The prover Peggy takes a proving key pk, a public input x and a private witness w.
Peggy generates a proof $pr = P(pk, x, w)$ that claims that Peggy knows a witness w and that the witness satisfies the program C.

The verifier Victor computes $V(vk, x, pr)$ which returns true if the proof is correct, and false otherwise.
Thus this function returns true if Peggy knows a witness w satisfying

$$C(x, w) = true$$

For example imagine I want to prove that I know the square root w of a large number x
The program C would take the parameters w (the secret) and x the public variable, square w and check if this is equal to x. If it is equal to return true, otherwise return false.

Trusted Setups and Toxic Waste

Note here the secret parameter lambda This parameter sometimes makes it tricky to use zk-SNARK in real-world applications. The reason for this is that anyone who knows this parameter can generate fake proofs. Specifically, given any program C and public input x a person who knows lambda can generate a proof $pr2$ such that $V(vk, x, pr2)$ evaluates to true **without** knowledge of the secret w.

From Proof Systems to ZKP Systems

Proving Systems

A statement is a proposition we want to prove. It depends on:

- Instance variables, which are public.
- Witness variables, which are private.

Given the instance variables, we can find a short proof that we know witness variables that make the statement true (possibly without revealing any other information).

What do we require of a proof ?

- Completeness: there exists an honest prover P that can convince the honest verifier V of any correct statement with very high probability.
- Soundness: even a dishonest prover P running in super-polynomial time cannot convince an honest verifier V of an incorrect statement. Note: P does not necessarily have to run in polynomial time, but V does.

To make our proof zero knowledge we also need ‘zero knowledginess’

To oversimplify: represented on a computer, a ZKP is nothing more than a sequence of numbers, carefully computed by Peggy, together with a bunch of boolean checks that Victor can run in order to verify the proof of correctness for the computation.

A zero-knowledge protocol is thus the mechanism used for deriving these numbers and defining the verification checks.

Interactive v Non Interactive Proofs

Non-interactivity is useful if we want to allow multiple independent verifiers to verify a given proof without each one having to individually query the prover.

In non-interactive zero knowledge protocols there is no repeated communication between the prover and the verifier. Instead, there is only a single “round”, which can be carried out asynchronously.
Using publicly available data, Peggy generates a proof, which she publishes in a place accessible to Victor (e.g. on a distributed ledger).
Following this, Victor can verify the proof at any point in time to complete the “round”. Note that even though Peggy produces only a single proof, as opposed to multiple ones in the interactive version, the verifier can still be certain that except for negligible probability, she does indeed know the secret she is claiming.

Succint v Non Succint

Succinctness is necessary only if the medium used for storing the proofs is very expensive and/or if we need very short verification times.

Proof v Proof of Knowledge

A proof of knowledge is stronger and more useful than just proving the statement is true. For instance, it allows me to prove that I know a secret key, rather than just that it exists.

Argument v Proof

In a proof, the soundness holds against a computationally unbounded prover and in an argument, the soundness only holds against a polynomially bounded prover.
Arguments are thus often called “computationally sound proofs”.

The Prover and the Verifier have to agree on what they’re proving. This means that both know the statement that is to be proven and what the inputs to this statement represent.

Alternatives to ZKSnarks and trusted setups

STARKS

(Scaleable Transparent Arguments of Knowledge)
zk-STARKs were created by Eli-Ben Sasson, a professor at the Technion-Israel Institute of Technology.
They generally, considered a more efficient variant of the technology - potentially faster, but the proof size is much greater.

2018 : Scalable and Transparent and post quantum secure computational integrity - Eli Ben Sasson et al.

E.g. Aurora , Hodor
Their underlying cryptographic assumptions (CRH) make them post quantum resistant

Interactive Oracle Proofs - Developed from PCPs
Although ZK Probabilistically Checkable Proofs have been known since the mid-1990’s, but “the proofs arising from the PCP theorem were so long and complicated that it would have taken thousands of years to generate and check them, and would have needed more storage bits than there are atoms in the universe.”

ZK-PCPs are transparent (or “public randomness”), which means that the randomness used by the verifier is public; in particular, setting up a ZK-PCP requires no external trusted setup phase,

A great deal of development in this area, see Starkware etc.

From the useful
https://github.com/gluk64/awesome-zero-knowledge-proofs (https://github.com/gluk64/awesome-zero-knowledge-proofs)

Comparison of the most popular zkp systems

	SNARKs	STARKs	Bulletproofs
Algorithmic complexity: prover	O(N * log(N))	O(N * poly-log(N))	O(N * log(N))
Algorithmic complexity: verifier	~O(1)	O(poly-log(N))	O(N)
Communication complexity (proof size)	~O(1)	O(poly-log(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Ethereum/EVM verification gas cost	~600k (Groth16)	~2.5M (estimate, no impl.)	N/A
Trusted setup required?	YES 😒	NO 😄	NO 😄
Post-quantum secure	NO 😒	YES 😄	NO 😒
Crypto assumptions	Strong 😒	Collision resistant hashes 😄	Discrete log 😒

Vitalik's Blog Articles

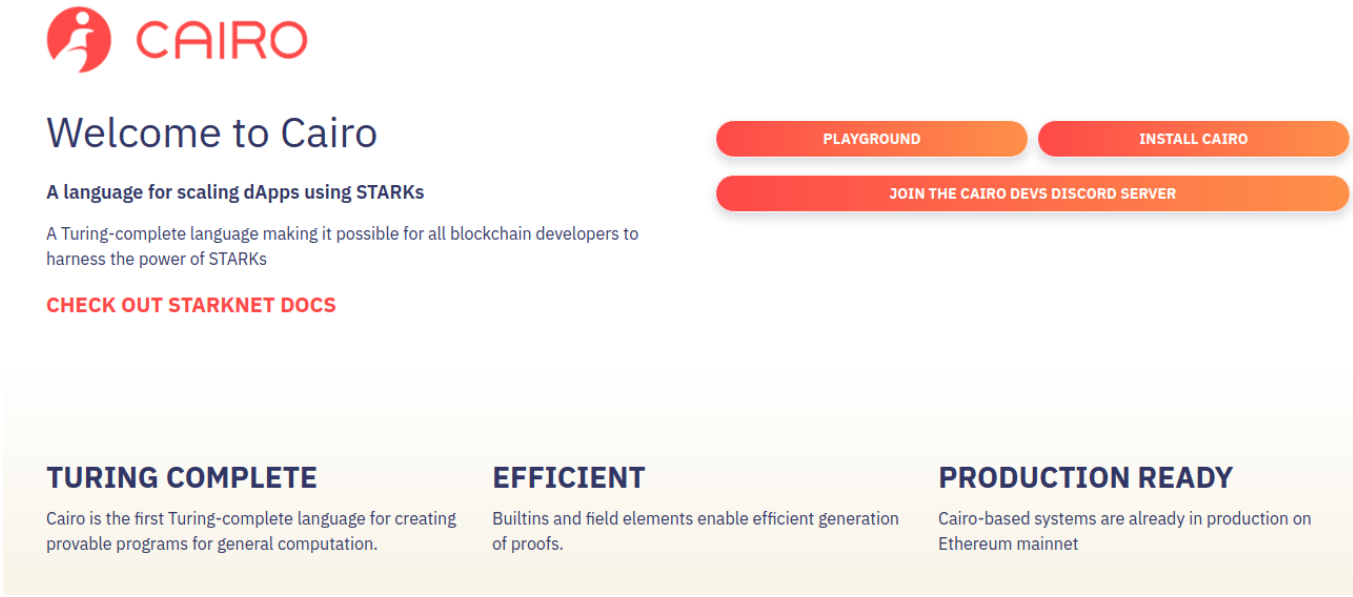
- Proofs with Polynomials (https://vitalik.ca/general/2017/11/09/starks_part_1.html)
- Thank Goodness It's FRI-day (https://vitalik.ca/general/2017/11/22/starks_part_2.html)
- Into the weeds (https://vitalik.ca/general/2018/07/21/starks_part_3.html)

The STARK paper (https://eprint.iacr.org/2018/046.pdf)

libSTARK (https://github.com/elibensasson/libSTARK)

More resources available at starkware.co (http://starkware.co)

Cairo



Zero Knowledge Proof Timeline

Changes have occurred because of

- Improvements to the cryptographic primitives (improved curves or hash functions for example)
- A fundamental change to the approach to zero knowledge
  See the excellent blog post from Starkware :
  The Cambrian Explosion (https://medium.com/starkware/the-cambrian-explosion-of-crypto-proofs-7ac080ac9aed)

1984 : Goldwasser, Micali and Rackoff - Probabilistic Encryption.
1989 : Goldwasser, Micali and Rackoff - The Knowledge Complexity of Interactive Proof Systems
1991 O Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. Preliminary version in 1986. (Graph colouring problem)

…

2006 Groth, Ostrovsky and Sahai introduced pairing-based NIZK proofs, yielding the first linear size proofs based on standard assumptions.

2010 Groth combined these techniques with ideas from interactive zero-knowledge arguments to give the first constant size NIZK arguments.

2016 : Jens Groth - On the Size of Pairing-based Non-interactive Arguments

From Matthew Green (https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/)

Prior to Goldwasser et al., most work in this area focused the soundness of the proof system. That is, it considered the case where a malicious Prover attempts to ‘trick’ a Verifier into believing a false statement. What Goldwasser, Micali and Rackoff did was to turn this problem on its head. Instead of worrying only about the Prover, they asked: what happens if you don’t trust the Verifier?

The process of creaing a ZKSnark

We can see this workflow in Zokrates, a project which allows you to create zx-SNARKS on the Ethereum blockchain.

A preview of the process flow in Zokrates

- The Creator writes and compiles a program in the Zokrates DSL
- The Creator / Prover generates a trusted setup for the compiled program
- The Prover computes a witness for the compiled program
- The Prover generates a proof - Using the proving key, she generates a proof for a computation of the compiled program
- The Creator / Prover exports a Verifier - Using the verifying key she generates a Solidity contract which contains the generated verification key and a public function to verify a solution to the compiled program

That gives us an overview of the process, lets now turn to the theoretical underpinnings, starting with a look at what it means to have a proof.

Transforming our Problem from High Level Language to a zero knowledge proof

Eli Ben Sasson explains the process as 2 parts

Arithmetization

From “I know the keys to spend a shielded cryptocurrency
to
I know four polynomials A(X), B(X), C(X), D(X), each of degree less than 1,000, such that this equality holds: $A(X)*B(X)-C(X) = (X^{1000}–1)*D(X)$

Low-degree compliance

This means using cryptography to ensure that the prover actually picks low-degree polynomials and evaluates those polynomials on randomly chosen points requested by the verifier.

Creating a SNARK in more detail

Lets look first at transforming the problem into a QAP, there are
3 steps :

- code flattening,
- conversion to a rank-1 constraint system (R1CS)
- formulation of the QAP.

Code Flattening

We are aiming to create arithmetic and / or boolean circuits from our code, so we change the high level language into a sequence of statements that are of two forms

x = y (where y can be a variable or a number)
and
x = y (op) z (where op can be +, -, *, / and y and z can be variables, numbers or themselves sub-expressions).

For example we go from

```
def qeval(x):
    y = x**3
    return x + y + 5
```

to

```
sym_1 = x * x
y = sym_1 * x
sym_2 = y + x
~out = sym_2 + 5
```

Rank 1 Constraint Systems

(From http://coders-errand.com/constraint-systems-for-zk-snarks/ (http://coders-errand.com/constraint-systems-for-zk-snarks/))

The important thing to understand is that a R1CS is not a computer program, you are not asking it to produce a value from certain inputs. Instead, a R1CS is a verifier, it shows that an already complete computation is correct .

The arithmetc circuit is a compositon of multplicatve sub-circuits (a single multiplcation gate and mutiple addition gates)

A rank 1 constraint system is a set of these sub-circuits expressed as constraints, each of the form:
$AXB = C$
where A, B, C are each linear combinations c1· v1+ c2· v2+ …
The c_i are constant field elements, and the v_i are instance or witness variables (or 1).

The inputs and outputs of a subcircuit are not predetermined.

- $AXB = C$ doesn't mean C is computed from A and B just that A, B, C are consistent.

More generally, an implementation of $x = f(a, b)$ doesn't mean that x is computed from a and b, just that x, a, and b are consistent.

Constraint languages can be viewed as a generalization of functional languages:

- everything is referentially transparent and side-effect free
- there is no ordering of constraints
- composing two R1CS programs just means that their constraints are simultaneously satisfied.

Thus our R1CS contains :

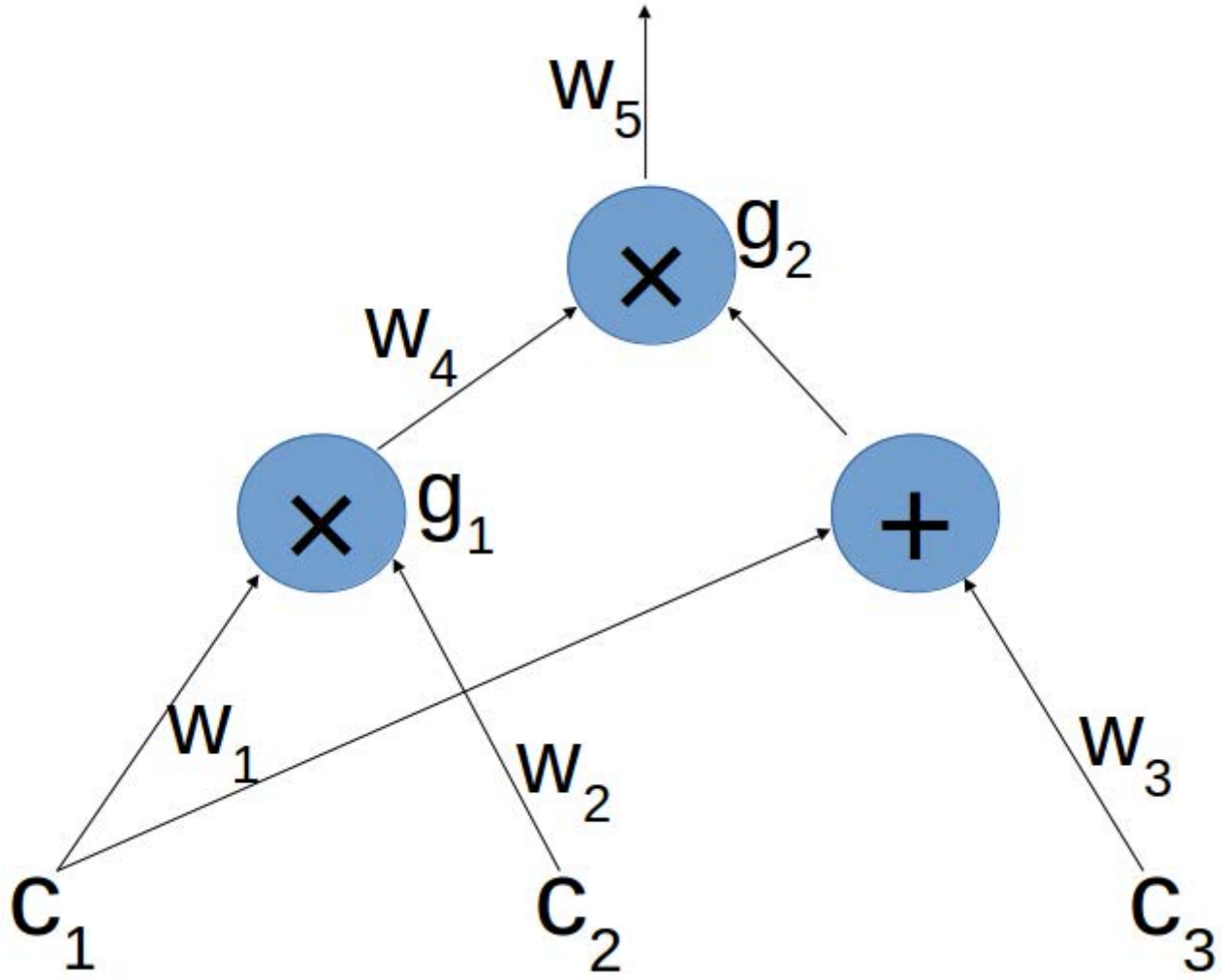
- the constant 1
- all public inputs
- outputs of the function
- private inputs
- auxilliary variables

The R1CS has

- one constraint per gate;
- one constraint per circuit output.

Example

Assume P wants to prove to V he knows
c1, c2, c3 such that (c1 · c2) · (c1 + c3) = 7
We transform the expression above into an arithmetic circuit as depicted below



Definition: A legal assignment for the circuit is of the form:

(c1, . . . , c5), where c4 = c1 · c2 and c5 = c4 · (c1 + c3).

From R1CS to QAP

Gennaro et al.showed that circuit satisfiability can be efficiently reduced to QAP satisfiability so we will use a Quadratic Arithmetic Program to verify circuit computation

Our goal is to devise a set of polynomials that simultaneously encode all of the constraints, so that we can verify the satisfiability thereof with a single check on the polynomials instead of a check over each constraint. The clever trick is to build the polynomials in a way that they can generate all of the constraints.
The R1CS is initially encoded as vectors and from these we define poynomials

We are aiming to create a left, right, and output polynomial of degree m for m constraints and a target polynomial of degree d

Given fixed values
$(c_1, \ldots, c_5)$
we use them as coefficients to define a left, right, and output polynomials.
We want to show that our created QAP has a legal assignment,

From our example above each multiplication gate X has an associated target point in the field $\mathbb{F}_p$. In our example, they will be 1 and 2.
• Each wire (in our example, there are 5) has an associated left, right and output polynomials: .
• Each polynomial evaluates to either 0 or 1 on the target points. If the polynomial is “related” to the multiplication gate associated with a target point, then it evaluates to 1. Otherwise to 0.

we define our left. right and output polynomials as

Our QAP Q of degree d and size m consists of polynomials

$L_1, \ldots, L_m, R_1, \ldots, R_m, O_1, \ldots, O_m$ and a target polynomial T of degree d.

An assignment (c1,...,cm) satisfies Q if,

defining

$$L := \sum_{i=1}^{m} ci \cdot Li, R := \sum_{i=1}^{m} ci \cdot Ri, O := \sum_{i=1}^{m} ci \cdot Oi$$

and

$$P := L \cdot R - O$$

and
we have that T divides P.

We moved from a situation where we had d groups of 3 * d vectors of m + 1 coefficients to one where we have 3 vectors each with m+1 polynomials of d-1 degree.

We converted a set of vectors into polynomials that generate them when evaluated at certain fixed points. We used these fixed points to generate a vanishing polynomial that divides any polynomial that evaluates to 0 at least on all those points. We created a new polynomial that summarizes all constraints and a particular assignment, and the consequence is that we can verify all constraints at once if we can divide that polynomial by the vanishing one without remainder. This division is complicated, but there are methods (the Fast Fourier Transform) that can perform if efficiently.

From http://coders-errand.com/how-to-build-a-quadratic-arithmetic-program/ (http://coders-errand.com/how-to-build-a-quadratic-arithmetic-program/)

Zero Knowledginess

Blind evaluation of a polynomial using Homomorphic Hiding

Suppose Alice has a polynomial P of degree d , and Bob has a point $s \in \mathbb{F}_p$ that he chose randomly.
Bob wishes to learn E(P(s)), i.e., the HH of the evaluation of P at s
Two simple ways to do this are:

1. Alice sends P to Bob, and he computes E(P(s))by himself.
2. Bob sends s to Alice; she computes E(P(s)) and sends it to Bob.

However, in the blind evaluation problem we want Bob to learn E(P(s)) without learning P which precludes the first option; and, most importantly,
we don't want Alice to learn s, which rules out the second

Using HH, we can perform blind evaluation as follows.

1. Bob sends to Alice the hidings $E(1), E(s), \ldots, E(s_d)$
2. Alice computes $E(P(s))$ from the elements sent in the first step, and sends $E(P(s))$ to Bob. (Alice can do this since E supports linear combinations, and $P(s)$is a linear combination of $1, s, \ldots, s_d$.)

Note that, as only hidings were sent, neither Alice learned s nor Bob learned P

The rough intuition is that the verifier has a "correct" polynomial in mind, and wishes to check the prover knows it. Making the prover blindly evaluate their polynomial at a random point not known to them, ensures the prover will give the wrong answer with high probability if their polynomial is not the correct one. This, in turn, relies on the Schwartz-Zippel Lemma stating that "different polynomials are different at most points".

but

the fact that Alice is able to compute E(P(s)) does not guarantee she will indeed send E(P(s)) to Bob, rather than some completely unrelated value.

Our process then becomes

1. Alice chooses polynomials L,R,O,H
2. Bob chooses a random point s ∈ $\mathbb{F}_p$, and computes E(T(s))
3. Alice sends Bob the hidings of all these polynomials evaluated at s, i.e. E(L(s)),E(R(s)),E(O(s)),E(H(s))
4. Bob checks if the desired equation holds at s. That is, he checks whether E(L(s)·R(s) −O(s))=E(T(s)·H(s)).

Furthemore we use

- Random values added to our s to conceal the s value

- The Knowledge of Coefficient Assumption to prove Alice can produce a linear combination of the polynomials.

If Alice does not have a satisfying assignment, she will end up using polynomials where the equation does not hold identically, and thus does not hold at most choices of s Therefore, Bob will reject with high probability over his choice of s.

We now need to make our proof non interactive, for this we produce the Common Reference String from Bob's first message

For details of these see : https://z.cash/blog/snark-explain6/ (https://z.cash/blog/snark-explain6/)

Other Projects

Bulletproofs

https://crypto.stanford.edu/bulletproofs/ (https://crypto.stanford.edu/bulletproofs/)
https://eprint.iacr.org/2017/1066.pdf (https://eprint.iacr.org/2017/1066.pdf)

Aim to prove that asecret committed value lies in a given interval.

Bulletproofs require no trusted setup. However, verifying a bulletproof is more time consuming than verifying a SNARK proof.
They rely only on the discrete logarithm assumption, and are made non-interactive using the Fiat-Shamir heuristic

a general drop-in replacement for Sigma-protocols.

Bulletproofs shrink the size of the cryptographic proof from over 10kB to less than 1kB. Moreover, bulletproofs support proof aggregation, so that proving that m transaction values are valid adds only O(log(m)) additional elements to the size of a single proof.

They concentrate on confidential transactions, but can be for general NP langs
The proof size islogarithmicin the numberof multiplication gates in the arithmetic circuit for verifying a witness. The proofs are much shorter than zk-SNARK and allow inputs to be Pedersen commitments to elements of the witness.

The high level idea of this proving system is that

1. The prover commits to a value(s) that he wants to prove knowledge of
2. The prover generates the proof by enforcing the constraints over the committed values and any additional public values. The constraints might require the prover to commit to some additional variables.

3. Prover sends the verifier all the commitments he made in step 1 and step 2 along with the proof from step 2.
4. The verifier now verifies the proof by enforcing the same constraints over the commitments plus any public values.

DIZK

Dizk Repo (https://github.com/scipr-lab/dizk)
Dizk Paper (https://eprint.iacr.org/2018/691)

DIZK provides Java-based implementations using Apache Spark [Apa17] for:

1. Proof systems
    - A serial and distributed preprocessing zkSNARK for R1CS (Rank-1 Constraint Systems), an NP-complete language that resembles arithmetic circuit satisfiability. The zkSNARK is the protocol in [Gro16].
    - A distributed Merlin-Arthur proof system for evaluating arithmetic circuits on batches of inputs; see [Wil16].
2. Scalable arithmetic
    - A serial and distributed radix-2 fast Fourier transform (FFT); see [Sze11].
    - A serial and distributed multi-scalar multiplication (MSM); see [BGMW93] [Pip76] [Pip80].
    - A serial and distributed Lagrange interpolation (Lag); see [BT04].
3. Applications using the above zkSNARK for
    - Authenticity of photos on three transformations (crop, rotation, blur); see [NT16].
    - Integrity of machine learning models with support for linear regression and covariance matrices; see [Bis06] [Can69] [LRF97] [vW97].

Mina (formerly CODA)

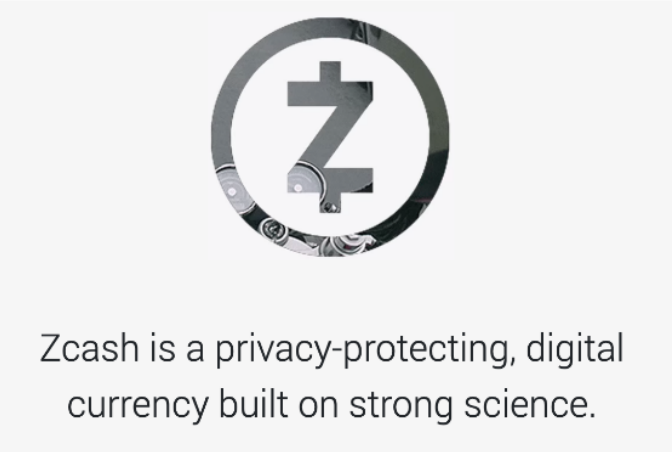
CODA - O(1) labs

From their white paper :
Coda is the first cryptocurrency protocol with a succinct blockchain.
With Coda however, no matter how much the usage grows, the blockchain always stays the same size - about ~20 kilobytes . Synchronizing the chain state with Coda requires receiving less than a megabyte of data. Concretely, a user of Coda requires only around 20 kilobytes and about 10 milliseconds to verify their balance.

Coda's succinct blockchain is enabled by recursive composition of zk-SNARKs. Recursive composition of SNARKs enable constant-sized proofs of arbitrary, incremental computations. This allows the Coda network to construct a proof of the validity of blockchain state that can be updated incrementally as more blocks are added.

ZKP Use Cases

Privacy preserving cryptocurrencies



Also Nightfall , ZKDai

Blockchain Scalability

For example
Rollups on Ethereum (https://github.com/barryWhiteHat/roll_up)

“The scalability of ZK rollup will increase by up to 4x, pushing theoretical max TPS of such systems well over 1000.” - Vitalik

A contract on the mainnet holds funds and a commitment to a sidechain state
Transactions on the sidechain are bundled together and committed as a proof on the mainnet.

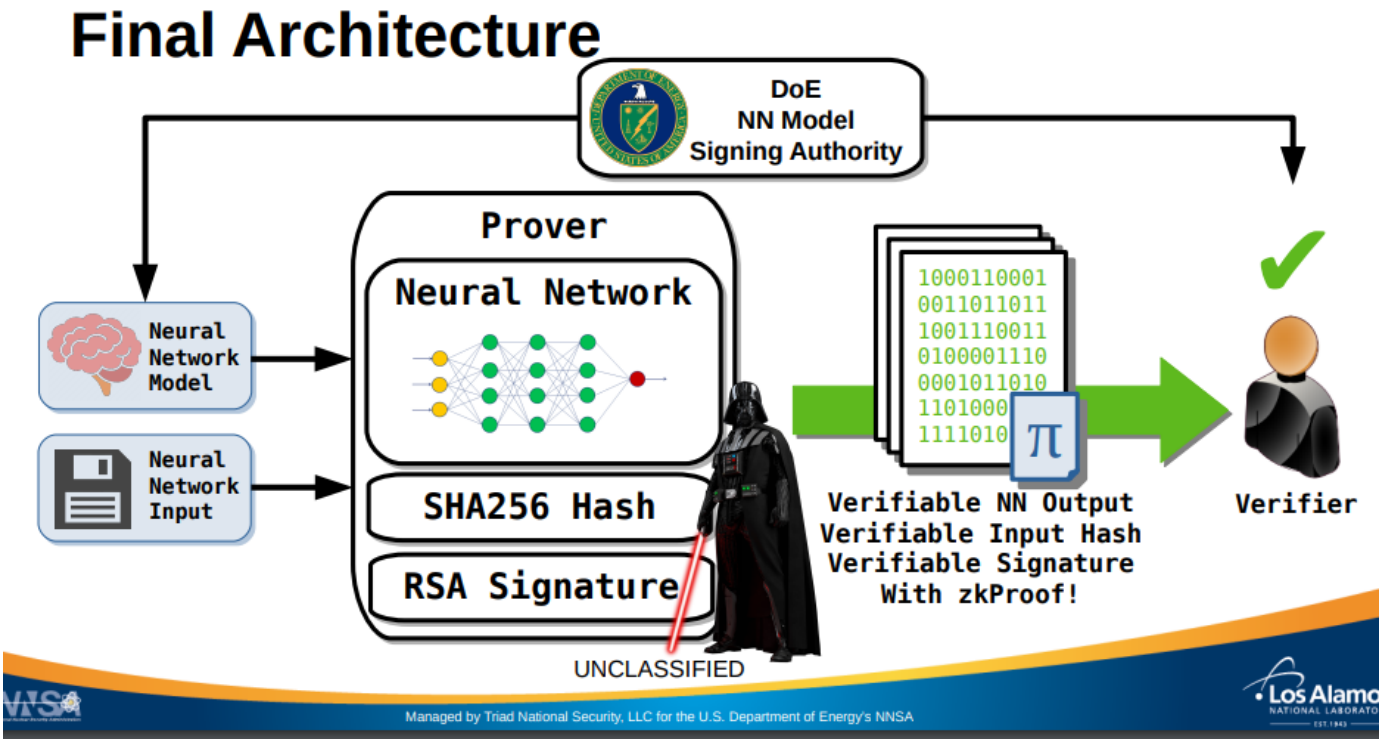
ZkSync (https://medium.com/matter-labs/introducing-zk-sync-the-missing-link-to-mass-adoption-of-ethereum-14c9cea83f58)
ZK Sync is designed to bring a VISA-scale throughput of thousands of transactions per second to Ethereum.

Nuclear Treaty Verification **(https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-20-20260)**

LA-UR-20-20260
Approved for public release; distribution is unlimited.

Title: SNNzkSNARK An Efficient Design and Implementation of a Secure Neural Network Verification System Using zkSNARKs

Author(s): DeStefano, Zachary Louis

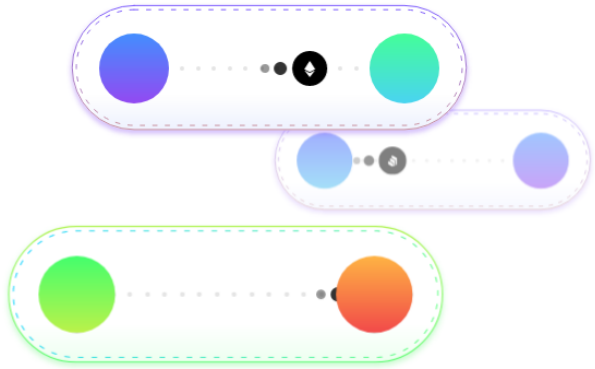


Privacy Preserving Financial Systems

Aztec

Privacy Guarantee

The new internet of money is secured by openness, but at a high price — all your counterparties know your entire financial history. Aztec is the ultimate security shield for the internet of money, protecting user and business data on Web3.0.



Identity Privacy

With cryptographic anonymity, sender and recipient identities are hidden

Balance Privacy

Transaction amounts are encrypted, making your crypto balances private

Code Privacy

Network observers can't even see which asset or service a transaction belongs to

Breakthrough Privacy Technology

Aztec is built on PLONK, the new superfast standard in universal SNARK technology — created by our world-class cryptography team

Fast Transactions

Private transactions are computed in seconds on all devices, and our rollup service saves gas and settles in minutes

DeFi Compatible

Fund and manage your DeFi positions anonymously, and trade them inside the Aztec privacy shield

Programmable

Program private money — escrow, money streaming, micropayments — your imagination is the limit

Public ERC20 45,000 Gas per Tx

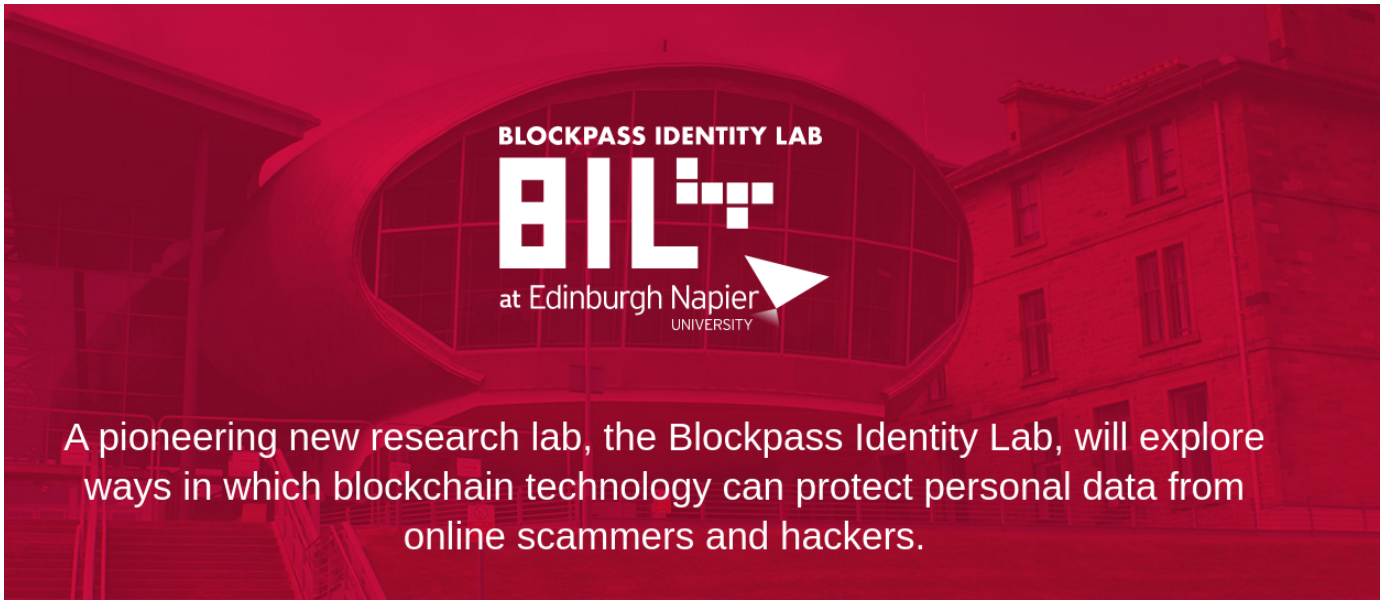
TurboPLONK 5,000 Gas per Tx TESTNET

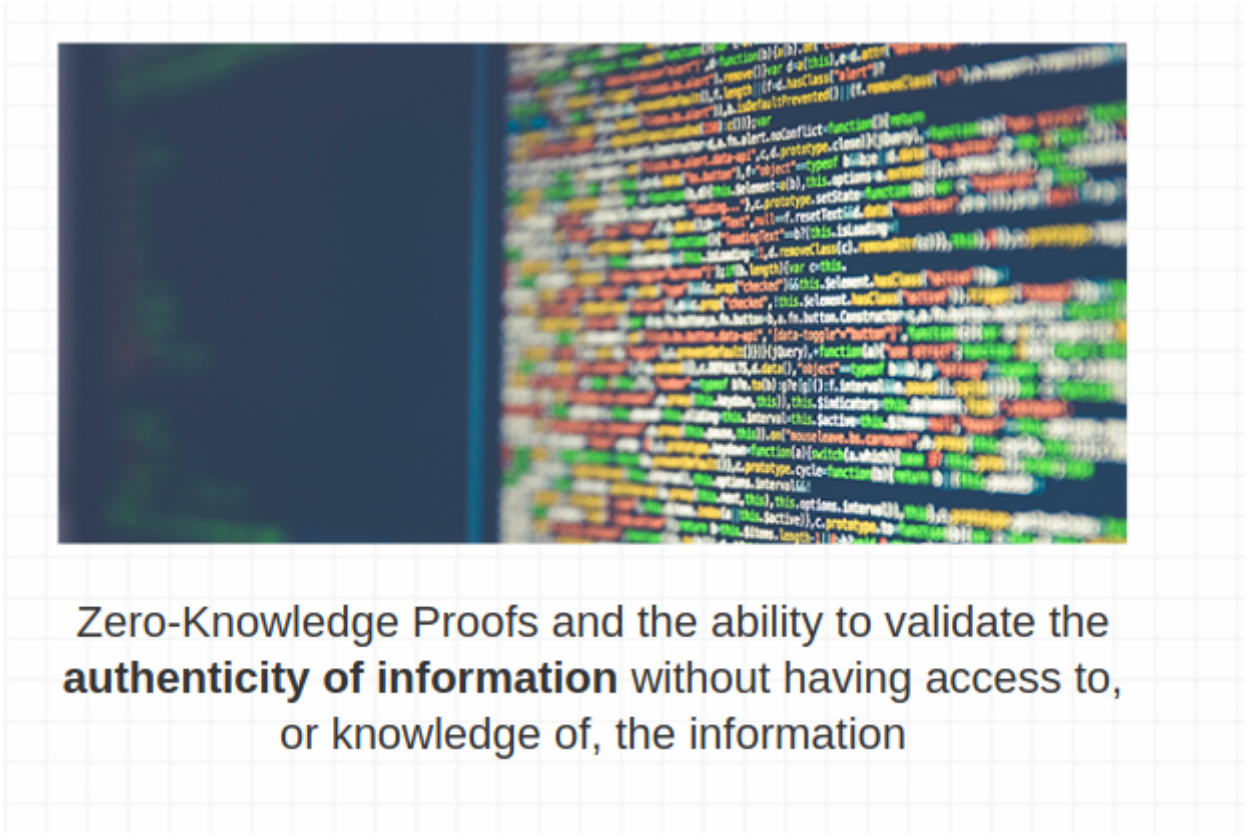
UltraPLONK 3,000 Gas per Tx COMING SOON

"The AZTEC protocol was created to enable privacy on public blockchains. It enables logical checks to be performed on encrypted values without the underlying values being revealed to the blockchain. The inputs and outputs of a transaction are encrypted using a series of zero-knowledge proofs and homomorphic encryption, yet the blockchain can still test the logical correctness of these encrypted statements."

Aztec 2.0 has been live for 6 months. Since launch, $5.5 million dollars has been sent through the system, giving users affordable privacy. Over 21,000 transactions have been sent, despite a 1 ETH cap, designed to protect against the experimental nature of the cryptography.

Identity and Privacy





Resources

ZCash links

Home (https://z.cash/)

Zcash deep dive (https://drive.google.com/file/d/1MwLcfoQ093euQk-DD8-aJ348BjvtxG1h/view)

Explaining Snarks Series (https://z.cash/blog/snark-explain/)

Rollup

Repo (https://github.com/barryWhiteHat/roll_up)

Libraries and toolkits

There is a problem that the various libraries aren't standardised.

libsnark (https://github.com/scipr-lab/libsnark)

- Tutorial (https://github.com/christianlundkvist/libsnark-tutorial)

Bellman (https://github.com/zkcrypto/bellman/)

- Demo (https://github.com/ebfull/bellman-demo)

jsnark - Java bindings to libsnark (https://github.com/akosba/jsnark)

snarkjs (https://github.com/iden3/snarkjs)

Snarky (https://github.com/o1-labs/snarky)

Zokrates (https://github.com/Zokrates/ZoKrates)

Ethsnarks (https://github.com/HarryR/ethsnarks)

- Ethsnarks Talk (http://tiny.cc/zksnarks)

Circom - circuit compiler (https://github.com/iden3/circom)

Useful Resources

Blogs

http://coders-errand.com/zero-knowledge-proofs-a-laymans-introduction/ (http://coders-errand.com/zero-knowledge-proofs-a-laymans-introduction/)
https://blog.decentriq.ch/zk-snarks-primer-part-one/ (https://blog.decentriq.ch/zk-snarks-primer-part-one/)
http://coders-errand.com/zk-snarks-and-their-algebraic-structure/ (http://coders-errand.com/zk-snarks-and-their-algebraic-structure/)

Talks from the last workshop

- Zcash deep dive by Jack (https://drive.google.com/file/d/1MwLcfoQ093euQk-DD8-aJ348BjvtxG1h/view)
- Designing zero knowledge circuits by Daira (https://drive.google.com/file/d/1DpPUoCTZ_78V5uvJE58IbeppgMM8Ztye/view)

Additional Reading

- Comparisons (https://github.com/gluk64/awesome-zero-knowledge-proofs)
- Zcash protocol specification (https://github.com/zcash/zips/blob/master/protocol/protocol.pdf)
- What are zk-SNARKs? by Zcash (https://z.cash/technology/zksnarks)
- ConsenSys intro to zk-SNARKs (https://media.consensys.net/introduction-to-zksnarks-with-examples-3283b554fc3b)
- Pinnochio Protocol (https://eprint.iacr.org/2013/279.pdf)
- Schnorr's Protocol document (http://www.lsv.fr/Software/spore/schnorr.pdf)
- Blockchain at Berkeley course (https://learnblockcha.in/)
- Discrete Logarithm problem (https://blog.goodaudience.com/towards-zero-knowledge-proof-0x01-the-discrete-log-assumption-a-constructive-approach-44705bbf57a7)
- Snarky (https://github.com/o1-labs/snarky)
- 3rd BIU Winter School on Cryptography 2013 - YouTube (https://www.youtube.com/playlist?list=PLXF_IJaFk-9C4p3b2tK7H9a9axOm3EtjA)
- 6th Bar-Ilan Winter School on Cryptography 2016 - YouTube (https://www.youtube.com/playlist?list=PLXF_IJaFk-9ADeshgHXrSfuwPyEMa5qS7)

Hackathon projects

- harnen/zk-game-of-life (https://github.com/harnen/zk-game-of-life)
- aspiers/zkp-eddsa-point-doubler: Zero Knowledge proof for point doubling on baby JubJub elliptic curve, using Zokrates (https://github.com/aspiers/zkp-eddsa-point-doubler)