An Introduction to

ZERO KNOWLEDGE

CRYPTOGRAPHY

on Solidity Friday’s

LinumLabs

Contents

What this workshop will cover

1 What is zero knowledge?

Cryptography basics, an introduction to zk & its applications

zk-Proofs 2

Properties, Types & the Maths Behind Them

3 zk-Privacy

Private Transactions on Public Blockchains

zk-Scalability 4

Rollups & zkEVMs

5 SNARKs

Succinct Non-interactive Arguments of Knowledge

STARKs 6

Scalable Transparent Arguments of Knowledge

LinumLabs

Cryptography

History & Basics

Cryptography is the study of secure communication in the presence of adversarial behaviour

It enables privacy in the digital world:

- Emails
- Bank Records
- Browsing History

Cryptography

History & Basics

Before the modern era, cryptography focused on **message confidentiality** (i.e., encryption) — Conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge (namely the key needed for decryption of that message)

The main classical cipher types are:

- **Transposition ciphers,** which rearrange the order of letters in a message (e.g., 'hello world' becomes 'ehlol owrdl' in a simple rearrangement scheme)

- **Substitution ciphers** which systematically replace letters or groups of letters with other letters or groups of letters (e.g., 'fly at once' becomes 'gmz bu podf' by replacing each letter with the one following it in the Latin alphabet)

Cryptography

History & Basics

Earliest record of cryptographic practices is 1900 BC

- Secret messages hidden under hair
- Julius Caesar shifted the letters of the alphabet before sending messages to generals (Caesar Cipher)
- WW1 morse code modulations
- WW2 electro mechanical rotor cipher machine known as Enigma
- Encryption standards, hashing & public key cryptography

Cryptography

Basics

Cryptography takes data & scrambles it up with an algorithm to form encrypted data which is nearly impossible to unscramble without the parameters that were passed into the algorithm.

- Hashing
- Encryption
- Signing
- Signatures
- Commitments
- Zero Knowledge Proof Systems

Cryptography

Basics - Hashing

A cryptographic hash function is a mathematical algorithm that maps data of an arbitrary size to a bit array of a fixed size. It is a one-way function, that is, a function for which it is practically infeasible to invert or reverse the computation.

Start with an input of variable length, pass it off to a hashing function (sha, md5, argon2, keccak256) which produces an indecipherable output of fixed length.

1. Hashing algorithms will always produce the same output given the same input.

2. Hashes are fast to compute and are unique.

Hashes allow developers to store data without needing to know its true value

- Password storage (but a hash alone is insufficient for storing passwords in a database)

Cryptography

Basics - Salt

Random value that gets added to the password before it is hashed, making it more difficult to crack with brute force.

Proof of Work algorithm for hashing blocks in blockchains.

Cryptography

Basics - HMAC

Hash-Based Message Authentication Code.

Hash that also requires a password, so the only person that can create the same hash signature must also have the corresponding password/key.

Cryptography

Basics - Encryption

Encryption - Take a message, scramble up the bytes to make it unreadable (**ciphertext**) then provide a key that allows someone else to decrypt it and read the message.

The cipher is typically **randomized**, so each encryption pass yields a different output, even if the key & message are the same.

Cryptography

Basics - Symmetric Encryption

Symmetric Encryption - Both the sender and receiver of the message have access to a shared password/key.

- This can lead to security vulnerabilities
- Both parties have to agree on one password.

Cryptography

Basics - Asymmetric Encryption

Public Key Cryptosystem.

2 keys are used

- A public key can be shared with other people.
- The private key should be kept secret.

The data gets e**ncrypted** with the public key, and **decrypted** with the private key.

Cryptography

Basics - Digital Signatures

Sender of message uses their **private key** to **sign** a **hash** of the original message.

- The private key guarantees authenticity.
- The hash guarantees that the message can't be tampered with (as this would produce an entirely different signature)

The **recipient (**verifier**)** can then use the **public key** to validate the authenticity of the message.

Cryptography

Basics - Commitments

Emulates envelope (conceals data**)** e.g. dApp for sealed bid auction - Once all bids are in, everyone opens their commitments.

A commitment scheme is 2 algorithms:

- **commit**(msg, r) => commitment string
- **verify**(msg, com, r) => accept/reject

*r is a secret random value chosen uniformly from space R

Properties:

- Binding
- Hiding

Zero Knowledge Cryptography

Introduction

A zero knowledge proof or protocol is a way for a “**prover**” to convince a “**verifier**” that a statement about some **secret information** is true without revealing the information itself.

First mentioned in the 1985 paper “The Knowledge Complexity of Interactive Proof Systems” by Shafi Goldwasser, Silvio Micali & Chales Rackoff.

Zero Knowledge Cryptography

Examples

- Suppose I want to prove to a company that I know what my password is, without entering my password (in case they get hacked)

- Imagine we are playing where’s Wally & suppose I want to prove to you that I know where he is, without revealing his location to you and thus ruining your chance of playing the game.

- If I have two balls that are slightly different colours & a colourblind friend, how could I prove to them that the balls are different colours without saying which ball is what colour.

- Ali Baba cave

Zero Knowledge Applications

Obscure Data

Condense Data

Provide Data

LinumLabs

zk Applications

- Security
- Patient Record Encryption
- Rollups
- Proof of Solvency
- Nuclear Disarmament
- Reputation Systems
- Voting
- Identity Verification
- Proof of Attack Malware Lab
- CO2 Emission Verification
- Gaming
- Ring Signatures
- Providing Private Data to Smart Contracts (Zorakles)
- ML on Blockchains
- Multi-party Computation
- Private Transactions on Public Blockchains
- Tax Payments
- Marketing Data

Privacy

Private Transactions on Public Blockchains

One property of public blockchains is **Universal Verifiability** - Only valid txs are posted to the blockchain.

Transaction data must be public to the ledger, otherwise how can miners verify it?

- Private transactions are possible on pubic blockchains if universal verifiability is maintained
- This is possible through cyptographic commitments and zero knowledge proofs.

Privacy

Private Transactions on Public Blockchains

- Universal Verifiability is achieved by committing data - Blockchain only stores a commitment to the data, not the actual verified data (hiding commitment)
- Hiding commitments reveal nothing about the actual state of the blockchain.
- They are **very** short strings (32kbs)

But if everything is committed, how do we know it is correct?

- Proof π: Short zkp that is fast to verify that proves:
  - Committed tx data is consistent with current state
  - Commited new state is correct

Privacy

Types of Digital Privacy

- **Pseudonymity:** One consistent name
  - Pro - Reputation
  - Con - Linkable to you

- **True Anonymity:** Unlinkable - System can't tell if 2 txs are from the same person
  - Reputation is possible but way more complex
  - Done through cryptographic commitments & zkps

Privacy

Simple Blockchain Anonymity via Mixing

Mixing service @ address M

3 people (addresses a,b,c) send 1 ETH each to M, M now has 3 ETH
They all privately send new addresses (x,y,z) to M who sends 1 ETh to each new address.

Observer knows y belongs to one of them, but not who (anonymity set of size 3) but can mix again

Problems:

- M knows everything
- M can abscond with 3 ETH!

Mixing without a mixer:

- Bitcoin - CoinJoin protocol (e.g. Wasabi Wallet)
- Ethereum - Trustless mix protocol (e.g. ZCoin, Keep3r Network)
- Solana - Rayduim

Privacy

Negative Aspects

- Criminal Activity
    - Tax Evasion
    - Ransomware (e.g. Wannacry)

Can we support positive applications of privacy on blockchains & prevent the negative ones

- Can we ensure legal compliance while preserving privacy?

- Yes! Through Zero Knowledge Proofs

Scalability

Rollups

A ZK-rollup chain is an off-chain protocol that operates on top of the Ethereum blockchain and is managed by on-chain Ethereum smart contracts.

ZK-rollups execute transactions outside of Mainnet, but periodically commit off-chain transaction batches to an on-chain rollup contract.

This transaction record is immutable, much like the Ethereum blockchain, and forms the ZK-rollup chain.

Scalability

Rollups

ZK-rollups are "hybrid scaling solutions"—off-chain protocols that operate independently but derive security from Ethereum. Specifically, the Ethereum network enforces the validity of state updates on the ZK-rollup and guarantees the availability of data behind every update to the rollup's state.

As a result, ZK-rollups are considerably safer than pure off-chain scaling solutions, such as sidechains, which are responsible for their security properties, or validiums, which also verify transactions on Ethereum with validity proofs, but store transaction data elsewhere

Scalability

Rollups

The ZK-rollup's core architecture is made up of the following components:

- **On-chain contracts**: As mentioned, the ZK-rollup protocol is controlled by smart contracts running on Ethereum. This includes the main contract which stores rollup blocks, tracks deposits, and monitors state updates. Another on-chain contract (the verifier contract) verifies zero-knowledge proofs submitted by block producers. Thus, Ethereum serves as the base layer or "layer 1" for the ZK-rollup.

- **Off-chain virtual machine** (VM): While the ZK-rollup protocol lives on Ethereum, transaction execution and state storage happen on a separate virtual machine independent of the EVM. This off-chain VM is the execution environment for transactions on the ZK-rollup and serves as the secondary layer or "layer 2" for the ZK-rollup protocol. Validity proofs verified on Ethereum Mainnet guarantee the correctness of state transitions in the off-chain VM.

Scalability

ZK Rollups rely on the main Ethereum protocol for the following

- **Data Availability -** ZK-rollups publish state data for every transaction processed off-chain to Ethereum. With this data, it is possible for individuals or businesses to reproduce the rollup's state and validate the chain themselves. Ethereum makes this data available to all participants of the network as calldata.

- **Transaction Finality -** Ethereum acts as a settlement layer for ZK-rollups: L2 transactions are finalized only if the L1 contract accepts the validity proof. This eliminates the risk of malicious operators corrupting the chain (e.g., stealing rollup funds) since every transaction must be approved on Mainnet. Also, Ethereum guarantees that user operations cannot be reversed once finalized on L1.

Scalability

ZK Rollups rely on the main Ethereum protocol for the following

- **Censorship Resistance -** Most ZK-rollups use a "supernode" (the operator) to execute transactions, produce batches, and submit blocks to L1. While this ensures efficiency, it increases the risk of censorship: malicious ZK-rollup operators can censor users by refusing to include their transactions in batches.

- As a security measure, ZK-rollups allow users to submit transactions directly to the rollup contract on Mainnet if they think they are being censored by the operator. This allows users to force an exit from the ZK-rollup to Ethereum without having to rely on the operator's permission.

How do ZK Rollups Work?

Transactions

Users in the ZK-rollup sign transactions and submit to L2 operators for processing and inclusion in the next batch. In some cases, the operator is a centralized entity, called a sequencer, who executes transactions, aggregates them into batches, and submits to L1. The sequencer in this system is the only entity allowed to produce L2 blocks and add rollup transactions to the ZK-rollup contract.

Other ZK-rollups may rotate the operator role by using a proof-of-stake validator set. Prospective operators deposit funds in the rollup contract, with the size of each stake influencing the staker’s chances of getting selected to produce the next rollup batch. The operator’s stake can be slashed if they act maliciously, which incentivizes them to post valid blocks.

How do ZK Rollups Work?

Transaction Data Publication

calldata is a data area in a smart contract used to pass arguments to a function and behaves similarly to memory. While calldata isn't stored as part of Ethereum's state, it persists on-chain as part of the Ethereum chain's history logs.

ZK-rollups use calldata to publish compressed transaction data on-chain; the rollup operator simply adds a new batch by calling the required function in the rollup contract and passes the compressed data as function arguments.

This helps reduce costs for users since a large part of rollup fees go toward storing transaction data on-chain.

How do ZK Rollups Work?

State Commitments

The ZK-rollup’s state, which includes L2 accounts and balances, is represented as a Merkle tree. A cryptographic hash of the Merkle tree’s root (Merkle root) is stored in the on-chain contract, allowing the rollup protocol to track changes in the state of the ZK-rollup.

The rollup transitions to a new state after the execution of a new set of transactions. The operator who initiated the state transition is required to compute a new state root and submit to the on-chain contract. If the validity proof associated with the batch is authenticated by the verifier contract, the new Merkle root becomes the ZK-rollup’s canonical state root.

How do ZK Rollups Work?

State Commitments

The ZK-rollup's state, which includes L2 accounts and balances, is represented as a Merkle tree. A cryptographic hash of the Merkle tree's root (Merkle root) is stored in the on-chain contract, allowing the rollup protocol to track changes in the state of the ZK-rollup.

The rollup transitions to a new state after the execution of a new set of transactions. The operator who initiated the state transition is required to compute a new state root and submit to the on-chain contract. If the validity proof associated with the batch is authenticated by the verifier contract, the new Merkle root becomes the ZK-rollup's canonical state root.

How do ZK Rollups Work?

Validity Proofs

Validity proofs allow parties to prove the correctness of a statement without revealing the statement itself (zero knowledge proofs) ZK Rollups use validity proofs to confirm the correctness of off-chain state transitions without having to re-execute transactions on Ethereum.

These proofs can come in the form of a **ZK-SNARK** (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) or **ZK-STARK** (Zero-Knowledge Scalable Transparent Argument of Knowledge).

Scalability

zk-EVMs

zk-EVMs use ZK-SNARK technology to make cryptographic proofs of execution of Ethereum-like transactions, either to make it much easier to verify the Ethereum chain itself or to build ZK Rollups that are (close to) equivalent to what Ethereum provides but are much more scalable. But there are subtle differences between these projects, and what tradeoffs they are making between practicality and speed.

- Fully Ethereum Equivalent
- Fully EVM Equivalent
- Almost EVM Equivalent
- High Level Language Equivalent

ZK-EVMS

Fully Ethereum Equivalent

They do not change any part of the Ethereum system to make it easier to generate proofs. They do not replace hashes, state trees, transaction trees, pre-compiles or any other in-consensus logic, no matter how peripheral.

These ZK-EVMs are what we ultimately need make the Ethereum layer 1 itself more scalable.

Disadvantage: Ethereum was not originally designed around ZK-friendliness, so there are *many* parts of the Ethereum protocol that take a large amount of computation to ZK-prove.

- The Privacy & Scaling Explorations team is working to create this type of zkEVM.

ZK-EVMS

Fully EVM Equivalent

These zk-EVMs strive to be exactly EVM-equivalent, but not quite Ethereum-equivalent. That is, they look exactly like Ethereum "from within", but they have some differences on the outside, particularly in data structures like the block structure and **state tree**.The goal is to be fully compatible with existing applications, but make some minor modifications to Ethereum to make development easier and to make proof generation faster.

These zkEVMs make changes to data structures that hold things like the Ethereum state. Fortunately, these are structures that the EVM itself cannot access directly, and so applications that work on Ethereum would almost always still work on these EVMs

Disadvantage: These zk-EVMs provide faster prover times than the previous ones mainly by removing parts of the Ethereum stack that rely on needlessly complicated and zk-unfriendly cryptography. The slowness from having to prove the EVM as-is, with all of the inefficiencies and zk-unfriendliness inherent to the EVM, still remains. But there is a slight improvement.

- **Scroll's zkEVM**
- **Polygon Hermez -** Both working towards the becoming this type of zk-EVM

ZK-EVMS

Partially EVM Equivalent

These zk-EVMs are *almost* EVM-equivalent, but make a few sacrifices to exact equivalence to further improve prover times and make the EVM easier to develop. They might remove a few features that are exceptionally hard to implement in a zk-EVM implementation.

Precompiles are often at the top of the list here;. Additionally, these zk-EVMs sometimes also have minor differences in how they treat contract code, memory or stack.

Disadvantage: There will be some applications that would need to be rewritten either because they use pre-compiles that these types of zk-EVMs remove or because of subtle dependencies on edge cases that the VMs treat differently.

- **Scroll's zk-EVM**
- **Polygon Hermez -** Both currently this type of zk-EVM

ZK-EVMS

High Level Language Equivalent

These work by taking smart contract source code written in a high-level language (eg. Solidity, Vyper, or some intermediate that both compile to) and compiling *that* to some language that is explicitly designed to be ZK-SNARK-friendly.

There is a *lot* of overhead that you can avoid by not ZK-proving all the different parts of each EVM execution step, and starting from the higher-level code directly. Compiling from high-level languages directly really can greatly reduce costs and help decentralization by making it easier to be a prover.

Disadvantages:

- zk-EVM address will differ from Ethereum address.
- Lots of debugging infrastructure cannot be carried over, because such infrastructure runs over the EVM bytecode.

- **ZKSync**

Protocols, Tools & Tech

Languages:

- Circom
- Cairo
- Solidity
- zkASM
- ZoKrates

Frameworks:

- snarkjs
- zksnark-rs

Tools:

- Warp

Blockchains:

- Ethereum
- Polygon Hermez
- Polygon Miden
- StarkNet
- StarkWare
- zkSync
- Scroll
- Mina
- Aztec

LinumLabs

ZERO KNOWLEDGE PROOFS

Properties of a Zero Knowledge Proof

Complete

All **valid inputs** return **true**

Sound

All **invalid inputs** return **false**

Zero Knowledge

The party requesting verification learns **nothing** about the statements that they didn't already know

LinumLabs

Zero Knowledge Proofs

- **Interactive zero-knowledge proofs**: In this type of ZKP, the prover and the verifier interact several times. The verifier challenges the prover who provides replies to these challenges until the verifier is convinced.
    - Useful if single verifier (e.g. compliance auditor)

- **Non-interactive zero-knowledge proofs**: Here the interaction between a prover and a verifier can be simulated by the prover, making direct communication to the verifier unnecessary and making proof generation possible to do offline. Subsequently such a proof can be sent and verified by also verifying that the simulation of the verifier was done correctly (via Fiat-Shamir)
    - Used when many verifiers (e.g. Miners in a rollup blockchain)

zk Proof Setup

The setup phase is where the prover creates a public and corresponding private key. The prover sends the public key to the verifier, who uses it to generate challenges.

The way a prover and a verifier know they're with the same statement is by using a Common Reference String (CRS). trusted setup is a process to generate this CRS.

CRS is combined with the Structured Reference String (SRS) which is encrypted so that it can be reused.

zk Proof Setup

- **Trusted Setups:** A trusted setup ceremony is a procedure that is done once to generate a piece of data that must then be used every time some cryptographic protocol is run.

- **Universal Setups:** A universal protocol is one that does not require a separate trusted setup for each circuit

Trusted Setup

The setup phase is where the prover creates a public and corresponding private key. The prover sends the public key to the verifier, who uses it to generate challenges

- **Trusted Setups:** A trusted setup ceremony is a procedure that is done once to generate a piece of data that must then be used every time some cryptographic protocol is run. Generating this data requires some secret information; the "trust" comes from the fact that some person or some group of people has to generate these secrets, use them to generate the data, and then publish the data and forget the secrets. But once the data is generated, and the secrets are forgotten, no further participation from the creators of the ceremony is required.
- Needed in Groth16, Plonk zkp systems
- Powers of Tau
- Constant time verification of setup

Transparent Proofs

A zk proof is transparent if it requires no trusted setup.

This greatly reduces the computation time for creating the witness & thus the proof.

They instead rely on verifiable randomness for the setup ceremony.

Relation, Statement, Witness

“I know a **secret** message, such that H(secret) = Some hash value”

- Statement - Public data describing what is being proven: Hash value
- Witness *w* - Private data that supports our statement: Secret
- Relation *R* - Between *w* & *x*: R(*w, x*)*: x =* H(*w*)

Relation, Statement, Witness

“I know a **secret** message, such that H(secret) = Some hash value”

- Statement - Public data describing what is being proven: Hash value
- Witness *w* - Private data that supports our statement: Secret
- Relation *R* - Between *w* & *x*: R(*w*, *x*)*: x* = H(*w*)

Statement ***π*** is **true** if there is a ***w*** s.t. R(*w*, *x*) is true.

Grading of a Zero Knowledge Proof

Prover Time

Proof Size

Verification Time

CRS & SRS Time

LinumLabs

Arithmetic Circuits

A finite field is a set of integers.

Fix a finite field $\mathbf{F}$ = {0, …, p-1} for some prime p > 2, where all arithmetic is done % p

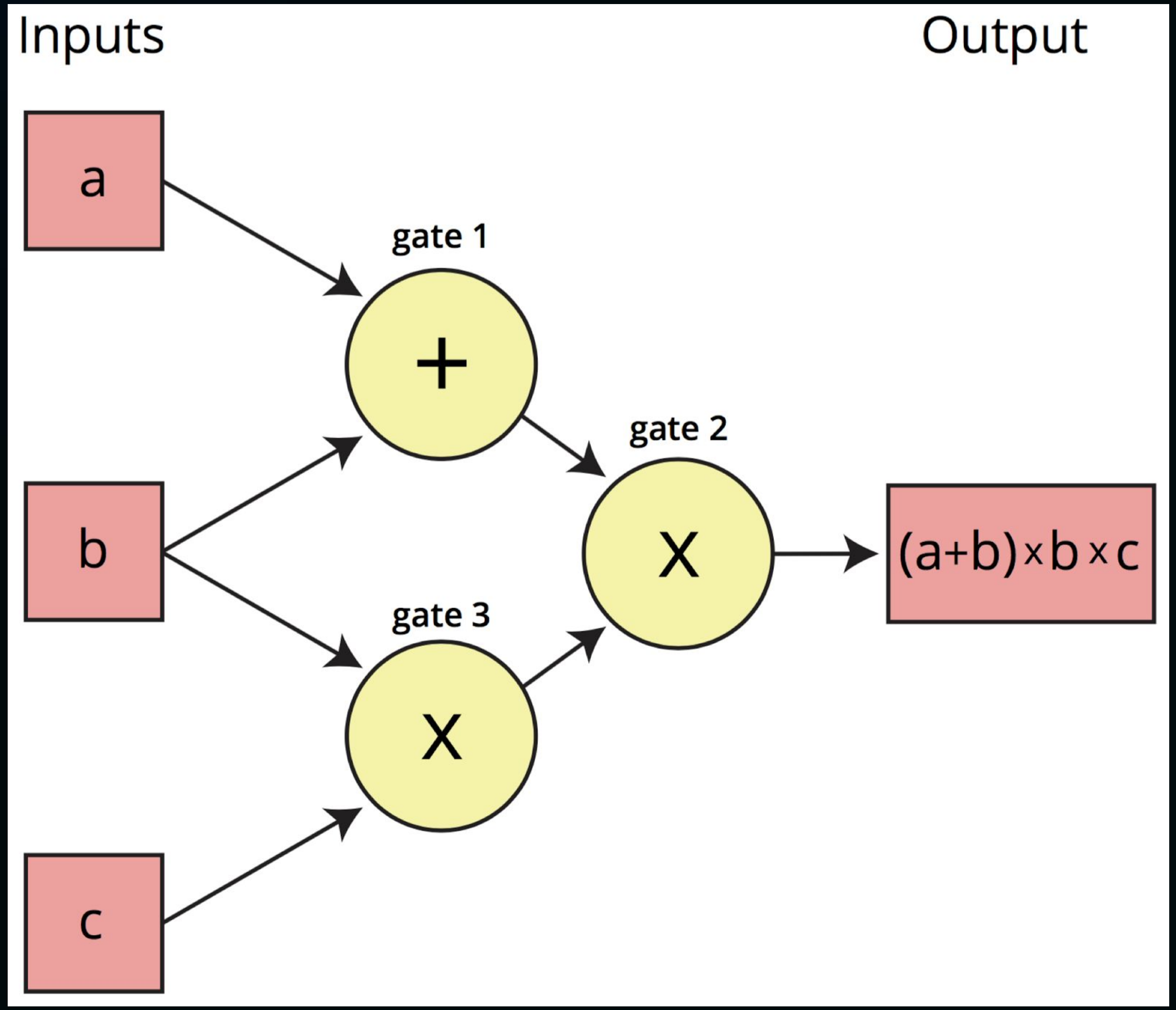
Arithmetic Circuit: C: F^n => $\mathbf{F}$

This is a Directed Acyclic Graph (DAG) where

- Internal nodes are labelled +, -, x
- Inputs are labelled 1, x1, …, xn

Evaluates an n-variate polynomial with an evaluation recipe

| C | = # gates in C (size)



https://www.pinterest.com/pin/682436149758405324/

Arithmetic Circuits

Example

C-hash(h, m): Outputs 0 if SHA256(m) = h

C-hash(h,m) = h - SHA256(m)

| C-hash | ≃ 20k gates

Argument Systems (for NP Problems)

2 parties, **prover** & **verifier** (p, v)

Fix circuit C, which takes 2 inputs:

- Public **statement** in $\mathbf{F}$^n (list of n elements in field $\mathbf{F}$)
- Secret **witness** in $\mathbf{F}$^m (list of m elements in field $\mathbf{F}$)

Prover given statement x & witness *w*

Verifier **only** given statement x

P’s goal is to “convince” V (via message passing) that ∃*w* such that C(x,*w*) = 0

Argument systems can be interactive or non-interactive.

Preprocessing Argument Systems

V accepts => P "knows" *w* such that C(x, *w*) = 0

If V accepts π, then it must be that P actually knows the witness *w*.

If a malicious prover, who does not "know" the witness tries to convince V of π, V will accept π with negligible probability.

Argument System Completeness

Prover P(sp, x, w) => proof π => Verifier V(sv, x, π)

If w = Correct witness for x,

$\forall(x, w)$: C(x, w) = 0 => Then an honest prover will always convince the honest verifier to accept π

LinumLabs

Argument Systems of Knowledge

V accepts => P “knows” *w* such that C(x, *w*) = 0

If V accepts π, then it must be that P actually knows the witness *w*.

If a malicious prover, who does not “know” the witness tries to convince V of π, V will accept π with negligible probability.

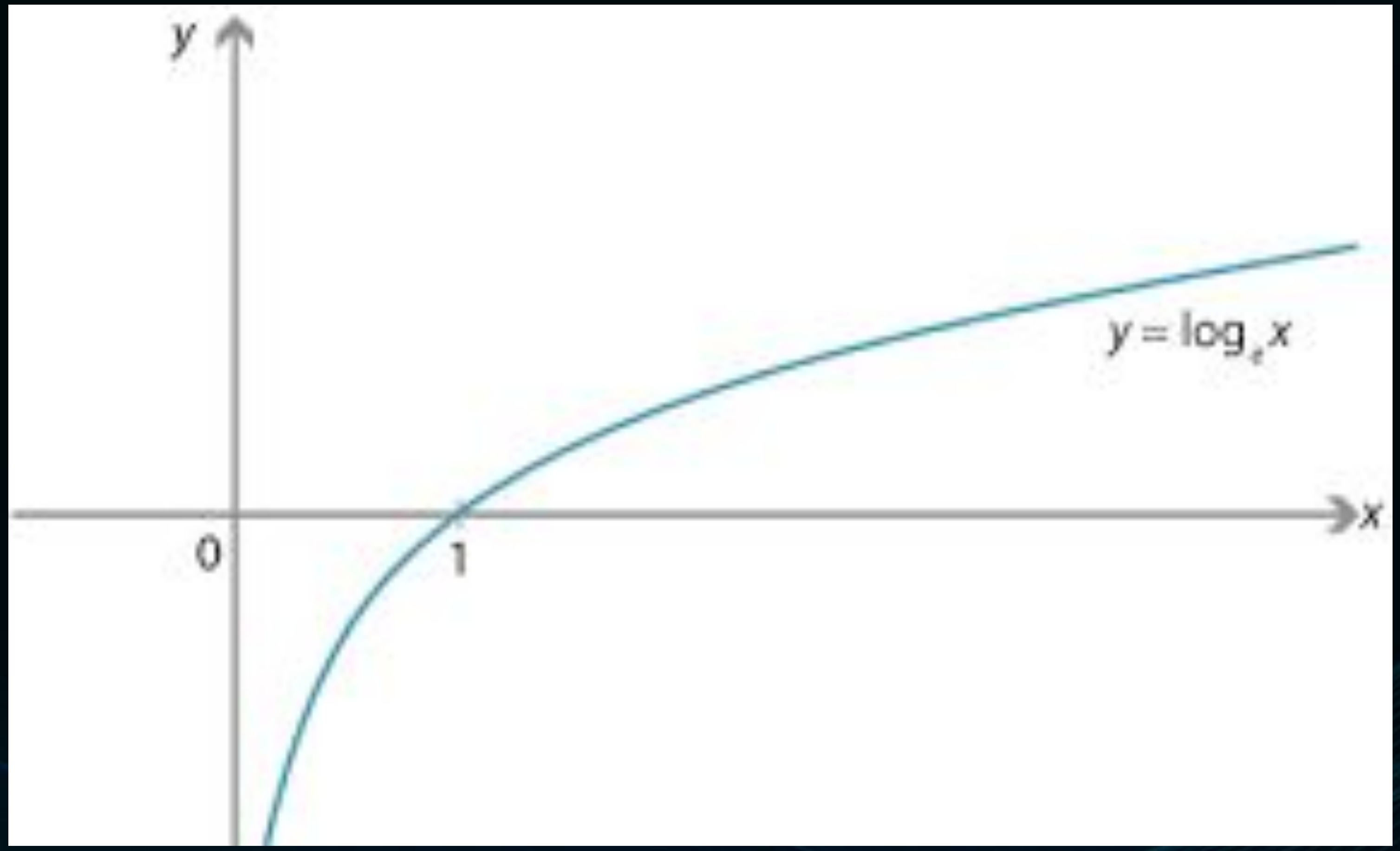
zk Argument Systems

π Reveals nothing about w to V

Succinct Argument Systems

We can require a 4th property (from preprocessing argument systems) which is that the argument is succinct

- For a π of size n, V can verify in time log(n)



zk-SNARKS

Zero-**K**nowledge **S**uccinct **N**on-interactive **AR**gument of **K**nowledge

Construction Flow:

1. Computation
2. Arithmetic Circuit
3. Rank 1 Constraint System
4. Quadratic Arithmetic Program
5. SNARK

zk-SNARKS

Zero-**K**nowledge **S**uccinct **N**on-interactive **AR**gument of **K**nowledge

Short proof, verifies in little time

Is a tuple (S, P, V)

- S(c) => Public params (Sp, Sv) for P & V
- P(Sp, x, w) => Short proof π, size = O(log | c |)
- V(Sv, x, π) => Accept / Reject, time = O(log| x |)

LinumLabs

zk-SNARKS

Zero-**K**nowledge **S**uccinct **N**on-interactive **AR**gument of **K**nowledge

- V(Sv, x, π) => Accept / Reject, time = O(log| x |) - V has to be so fast that it doesn't have to read the circuit, how?
- Through the pre-processing system.

To help V verify such large circuits we summarize the circuit and create the parameters (Sp, Sv), where Sv is a short summary of the circuit being verified

- If (S, P, V) is succinct & V learns nothing about *w* from π => zk-SNARK

zk-STARKs

Zero-**K**nowledge **S**calable **T**ransparent **AR**gument of **K**nowledge

(A zk proof is transparent if it requires no trusted setup)

Like ZK-SNARKs, ZK-STARKs prove the validity of off-chain computation without revealing the inputs. However, ZK-STARKs are considered an improvement on ZK-SNARKs because of their scalability and transparency.

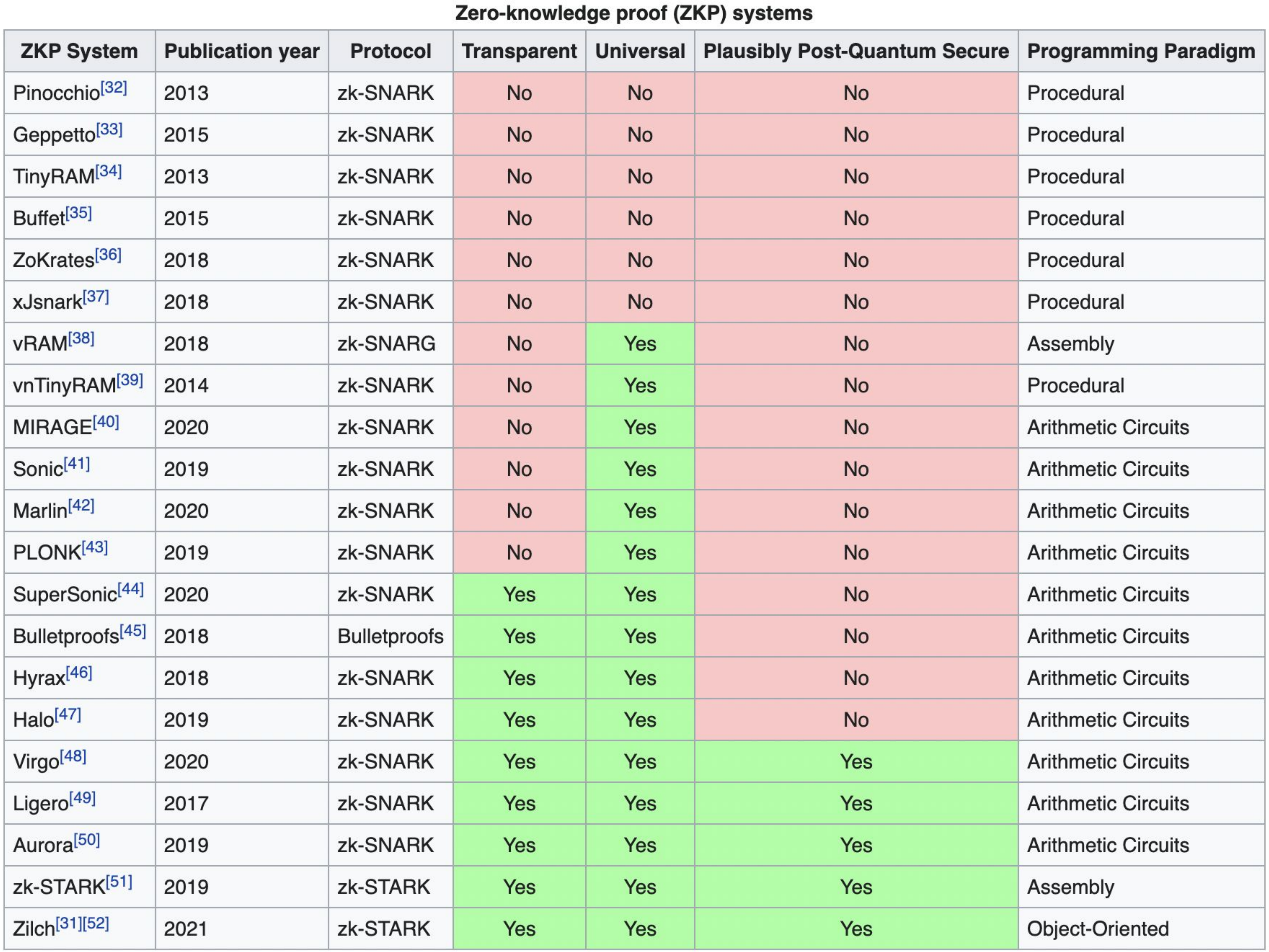
ZK-STARKs are 'transparent', as they can work without the trusted setup of a Common Reference String (CRS). Instead, ZK-STARKs rely on publicly verifiable randomness to set up parameters for generating and verifying proofs.

With ZK-SNARKs, proving and verification times scale *linearly* in relation to the size of the underlying computation. This means ZK-STARKs require less time than ZK-SNARKs for proving and verifying when large datasets are involved, making them useful for high-volume applications.

zk-STARKs

Zero-**K**nowledge **S**calable **T**ransparent **AR**gument of **K**nowledge

- STARKs have **no** quantum risk



Zero-knowledge proof (ZKP) systems

ZKP System	Publication year	Protocol	Transparent	Universal	Plausibly Post-Quantum Secure	Programming Paradigm
Pinocchio[32]	2013	zk-SNARK	No	No	No	Procedural
Geppetto[33]	2015	zk-SNARK	No	No	No	Procedural
TinyRAM[34]	2013	zk-SNARK	No	No	No	Procedural
Buffet[35]	2015	zk-SNARK	No	No	No	Procedural
ZoKrates[36]	2018	zk-SNARK	No	No	No	Procedural
xJsnark[37]	2018	zk-SNARK	No	No	No	Procedural
vRAM[38]	2018	zk-SNARG	No	Yes	No	Assembly
vnTinyRAM[39]	2014	zk-SNARK	No	Yes	No	Procedural
MIRAGE[40]	2020	zk-SNARK	No	Yes	No	Arithmetic Circuits
Sonic[41]	2019	zk-SNARK	No	Yes	No	Arithmetic Circuits
Marlin[42]	2020	zk-SNARK	No	Yes	No	Arithmetic Circuits
PLONK[43]	2019	zk-SNARK	No	Yes	No	Arithmetic Circuits
SuperSonic[44]	2020	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Bulletproofs[45]	2018	Bulletproofs	Yes	Yes	No	Arithmetic Circuits
Hyrax[46]	2018	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Halo[47]	2019	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Virgo[48]	2020	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
Ligero[49]	2017	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
Aurora[50]	2019	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
zk-STARK[51]	2019	zk-STARK	Yes	Yes	Yes	Assembly
Zilch[31][52]	2021	zk-STARK	Yes	Yes	Yes	Object-Oriented

https://en.wikipedia.org/wiki/Zero-knowledge_proof#Variants_of_zero-knowledge

LinumLabs

Bridging the gap between
people and Web3 technology.

@LinumLabs
/LinumLabs
Linum-Labs
linumlabs.com

Web3 Sanctuary

Learn Web3 the easy way

An edtech platform to train the next generation of
solidity developers for blockchain companies

Thank You

Please reach out to me if you have any questions

- tamara.ringas@gmail.com
- Twitter: @tamaraRingas
- Instagram: @tamrias