Zero Knowledge Proofs

•••

Gokul Alex

Can the Russians Prove to the US that a sealed envelope contains an authentic nuclear warhead without revealing anything about its design?

Can I prove to you that the number 385, 608, 108, 395, 369, 363, 400, 501, 273, 594, 475, 104, 405, 448, 848, 047, 062, 278, 473, 983 has a prime factor whose last digit is 7 without giving you any information about its prime factors?

I know that 26781 is not a prime since it is 113 times 237, to prove to you that fact, I will present these factor and demonstrate that indeed 113 × 237 = 26781.

What are Proofs?

Introduction

- A proof is a claim or theorem you are trying to prove &
- A sequence of derivations made to declare the theorem has been proved

What are Proofs?

An introduction

- Prover Makes Claim
- Verifier Chooses to Accept/Reject
- Both the Prover and Verifier are algorithms
- These short proofs should be verified in polynomial time.

Proofs

Introduction

- In an NP-language = two conditions must hold:
- Completeness: True claims will be accepted by verifier (allows honest provers to reach verification)
- Soundness: False claims will have no proofs (for all cheating prover strategy they will be unable to prove correctness of incorrect claim)

Limitations of the Conventional Proofs

Proofs Could Be Redundant

- A typical byproduct of a normal proof system is that you gained some knowledge, other than that you are now convinced that the statement is true.
- In the example before, not only are you convinced that 26781 is not a prime, but you also learned its factorization

Proofs

Interactive and Probabilistic

- Interaction: Rather than just reading the proof, the verifier engages with a prover back and forth over several message rounds.
- Randomness: Verifier's requests to prover are randomised and prover must be able to answer correctly to each.

Interactive Proofs

Probabilistic Polynomial Time

- Using interaction and randomness together it is possible to prove a claim to a blind verifier in Probabilistic Polynomial Time (PPT

What is Zero Knowledge?

Introduction

- What a verifier can compute after an interaction is identical to what they could prove prior.
- The interaction over multiple rounds between the prover & verifier has not increased the computational power of the verifier.

Why do we need zero knowledge proofs?

Zero-Knowledge (ZK) crypto lets you prove to you that I know a fact, without telling you the fact.

- I know the private key corresponding to an Ethereum account - but I won't tell you what my private key is!
- I know a way to fill in a map with 3 colors such that no two adjacent regions are the same color - but I won't tell you the coloring!
- I know a number x such that SHA256(x) = 0x77af... - but I won't tell you x!

Zero Knowledge Applications

Proving Statements on Private Data

- Person A has more than X in his bank account
- In the last year, a bank did not transact with an entity Y
- Matching DNA without revealing full DNA
- One has a credit score higher than Z

Zero Knowledge Applications

Anonymous Authorizations

- Proving that requester R has right to access web-site’s restricted area without revealing its identity (e.g., login, password)
- Prove that one is from the list of allowed countries/states without revealing from which one exactly
- Prove that one owns a monthly pass to a subway/metro without revealing card’s id

Zero Knowledge Applications

Anonymous Payments

- Payment with full detachment from any kind of identity
- Paying taxes without revealing one’s earnings

Zero Knowledge Applications

Outsourcing Computation

- Outsource an expensive computation and validate that the result is correct without redoing the execution; it opens up a category of trustless computing
- Changing a blockchain model from everyone computes the same to one party computes and everyone verifies

Zero Knowledge Concepts

Simulation Paradigm

Introduction

- Real View: All possible histories of interactions between Prover & Verifier (P,V)
- Simulated View: The verifier simulates all possible interactions between Prover & Verifier

Simulation Paradigm

Introduction

- A polynomial-time distinguisher makes an attempt to determine whether they are looking at the real or simulated view and requests a sample from both repeatedly.
- The two views are said to be "computationally indistinguishable" if for all distinguisher algorithms/strategies, even after receiving a polynomial number of samples from real or simulated, the probability is >1/2.

Zero Knowledge Arguments of Knowledge!

Definition

- An interactive protocol (P,V) is zero-knowledge if there exists a simulator (algorithm) such that for every probability polynomial-time verifier, the probability distributions determining the real from simulated view are computationally indistinguishable.
- Interactive Protocols are useful when there is a single verifier. An example would be a tax auditor in a zero-knowledge 'proof of taxes' application.

Zero Knowledge Proof Systems

A beautiful concept in computer science and cryptography

- In a zero-knowledge proof Alice will prove to Bob that a statement X is true
- Bob will completely convinced that X is true, but will not learn anything as a result of this process.
- That is, Bob will gain zero knowledge.
- There are applications ranging from practical signature schemes to proving that many NP-complete problems are hard even to approximate.

Zero Knowledge Proofs are proofs that fully convince that a statement is true without yielding any additional knowledge.

Zero Knowledge Proofs were invented by Goldwasser, Micali and Rackoff (GMR) in 1982.

Zero Knowledge Proof System

Essential Elements

Setup: a prover wants to convince a verifier that they know something, without revealing the underlying information.

Verifier: Asks questions / issues challenges to the prover, and checks responses.

Prover: Responds to verifier questions or challenges.

ZKP Properties

ZK Protocols have 3 properties:

- **Zero Knowledge :** The Prover's responses don't reveal the underlying information.
- **Completeness** : If the Prover knows the underlying information, they're always able to answer satisfactorily.
- **Soundness** : If the Prover doesn't know the underlying info, they'll eventually get caught.

What are the components of zero knowledge proof system?

- A proof system can be thought of as an algorithm V (for “verifier”) that takes as input a statement which is some string **X** and another string **P** known as the proof and outputs 1 if and only if **P** is a valid proof that the statement **X** is correct.
- For example, in Euclidean geometry, statements are geometric facts such as **“in any triangle the degrees sum to 180 degrees”** and the proofs are step by step derivations of the statements from the five basic postulates.

In the zero knowledge proof system the verifying algorithm should be efficient.

The proof should be a generalised protocol such that it makes easy to verify that the statement is true.

GMR considered a generalisation of proof such that an interactive probabilistic protocol was constructed between the prover and verifier.

GMR generalized ZKP as a game between prover and verifier. The game can be interactive, where the verifier asks questions and prover gives answers. The goal of the game is for the prover to convince the verifier that the statement is true.

Thus an interactive proof is a game between a computationally bounded verier and a computationally unbounded prover whose goal is to convince the verier of the validity of the assertion.

Implementation Methods

Identification Protocols

Alice knows a solution X to a puzzle P, and proves her identity to Bob by, for example, providing an encryption C of X and proving in zero knowledge that C is indeed an encryption of a solution for P.

Bob can verify the proof, but because it is zero knowledge, learns nothing about the solution of the puzzle and will not be able to impersonate Alice.

Implementation Methods

Compiling Protocols

The idea is that all parties prove in zero knowledge that they follow a protocol specifications. Normally such proofs might require the parties to reveal their secret inputs, hence violating security.

But zero knowledge precisely guarantees that we can verify correct behavior without access to these inputs.

Implementation Methods

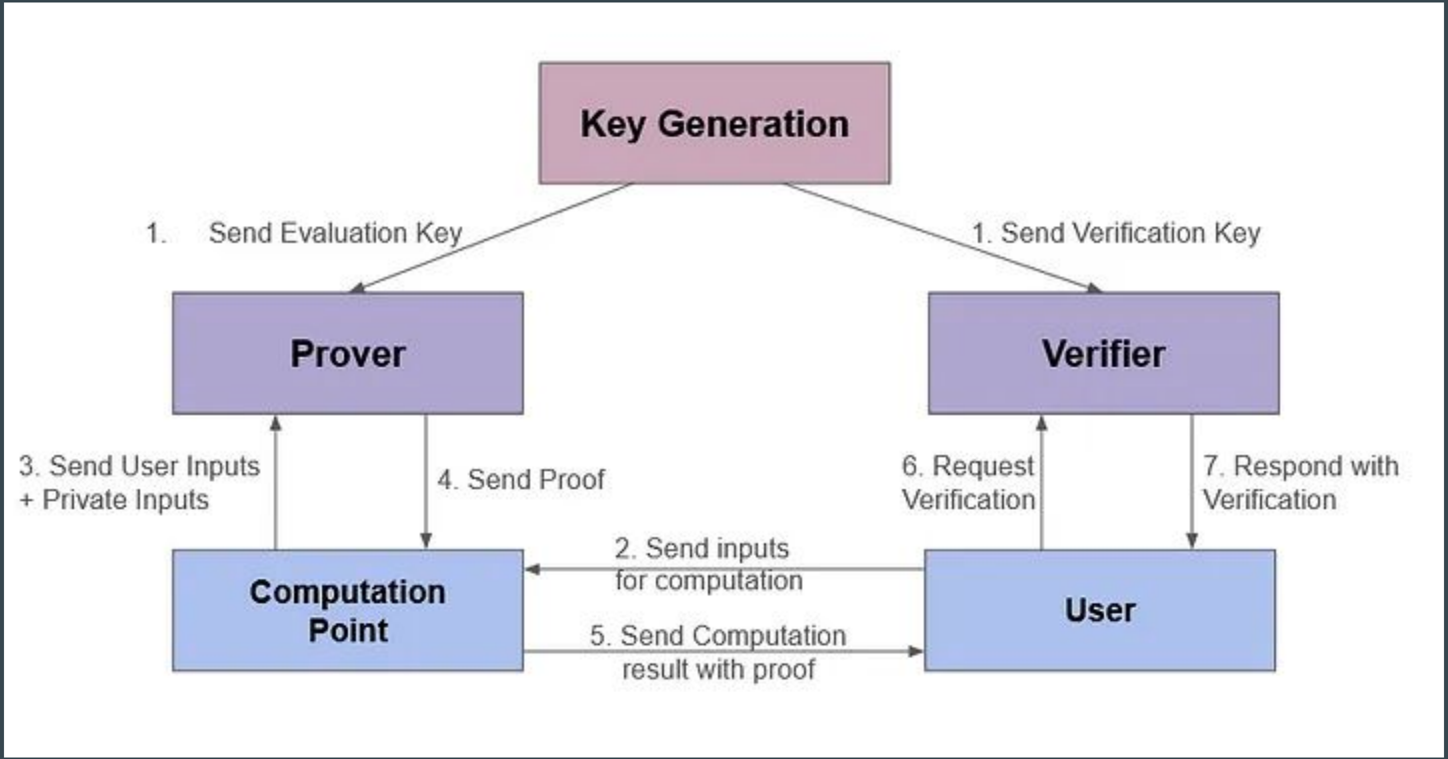
Zero Knowledge Proofs and Electronic Voting

Zero Knowledge Proofs could allow completely transparent vote counting, where every citizen could verify that the votes were counted correctly.

For example, David Chaum suggested an approach by publishing an encryption of every vote and then having the central authority prove that the final outcome corresponds to the counts of all the plaintexts.

Zero Knowledge Proof as Verifiable Computation

Verifiable Computation Using Cryptographic Proofs


Key Generation
1. Send Evaluation Key
1. Send Verification Key
Prover
Verifier
3. Send User Inputs + Private Inputs
4. Send Proof
6. Request Verification
7. Respond with Verification
2. Send inputs for computation
Computation Point
User
5. Send Computation result with proof

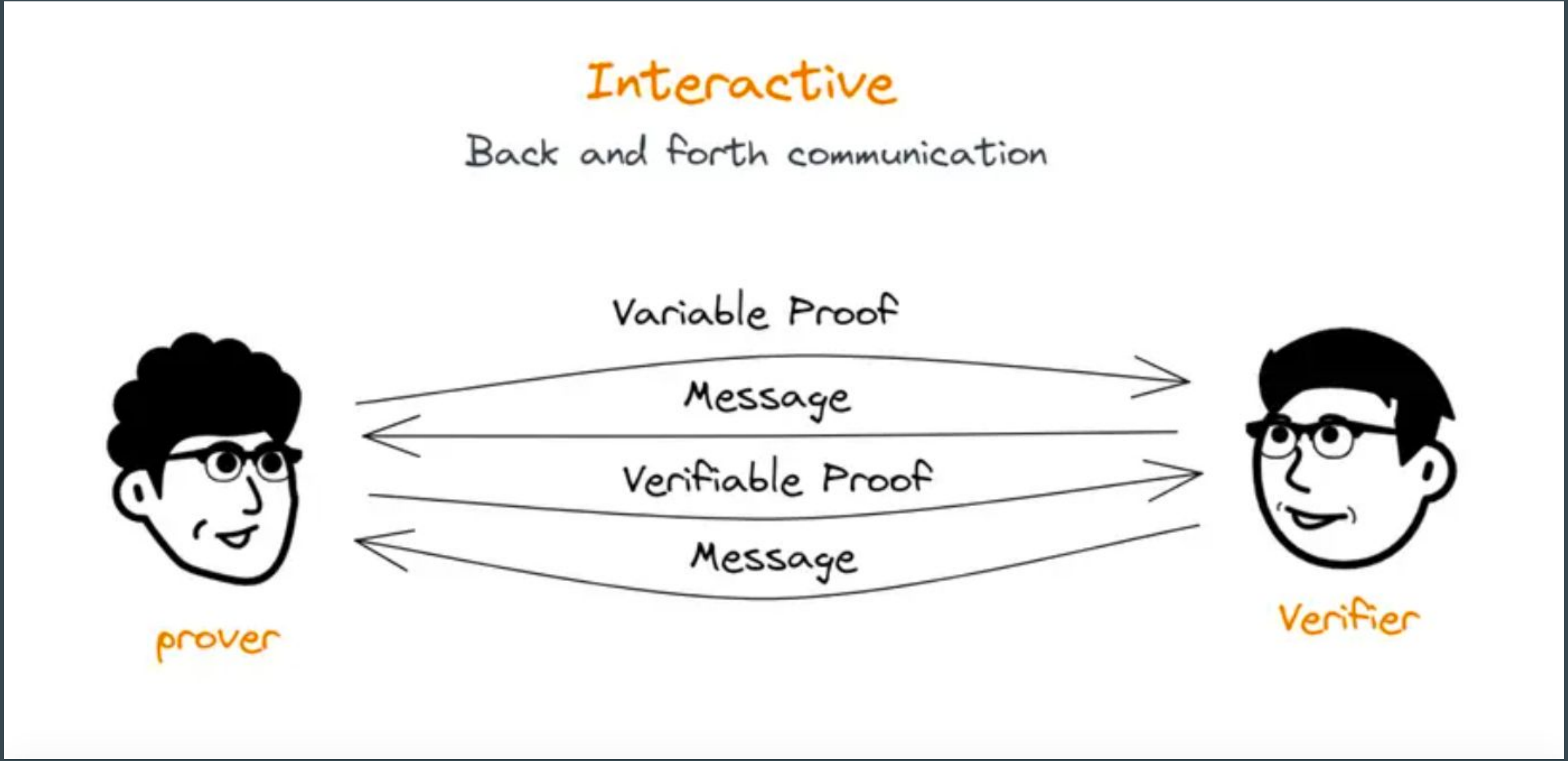
Verifiable Computation Using Cryptographic Proofs

The scheme typically consists of three main steps:

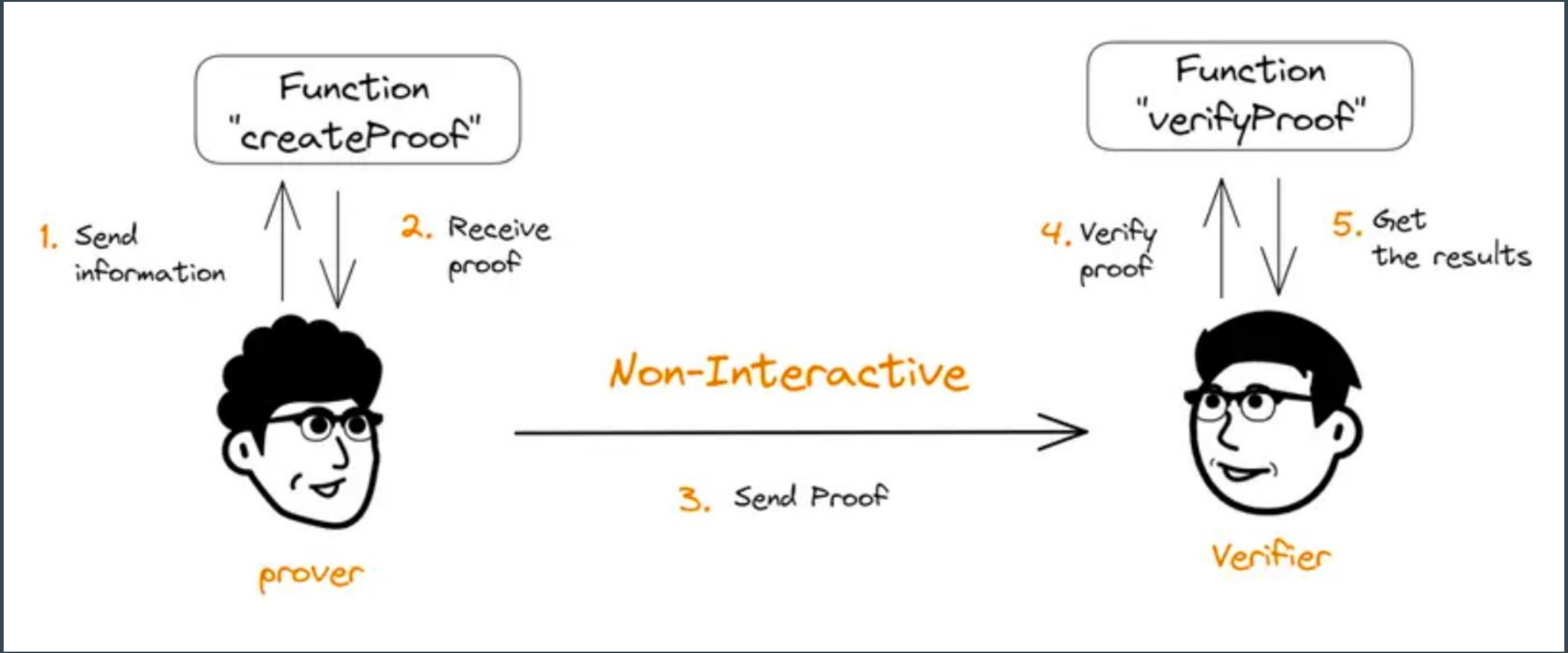
- Key Generation: this process takes the function to be computed, and generates an evaluation and a verification key
- Computation with Proof: this process performed at a prover node performs the desired computation on a given input u, and uses the evaluation key to generate a proof of correct computation.
- Verification: A Verifier then uses the proof from the prover, along with the verification key from the generation step, to verify that the computation was indeed done correctly.

ZKP Approaches

Interactive Zero Knowledge (IZK)



Non-Interactive Zero Knowledge (NIZK)



NIZK

Schemes

Pinnochio

Groth16

Sonic

Marlin

PLONK

STARK

Aurora

Halo

NIZK

Tooling

Circom

Arkworks

ZoKrates

zkSNARK

Three Important Phases

- Conducting trusted setup by Multi Party Computation protocol to generate proving and verification keys,
- Generating proof by prover using prover key, public input, and secret input (witness)
- Verifying the proof by verifier.

zkSNARKs

Key Features

- SNARKs require a trusted setup phase, where an initial set of parameters, often called a Structured Reference String (SRS), is generated.
- This setup phase uses a secret that, if exposed, will compromise the security of all subsequent proofs created with that setup.
- This setup data is often referred to as "toxic waste". Trusted setups are often seen as a drawback since they introduce potential trust issues
- Users must trust that the setup was performed correctly and that the secret was destroyed afterward.

zkSNARKs

Key Features

- Many SNARK constructions rely on elliptic curve cryptography, which depends on the hardness of the discrete logarithm problem (DLP).
- While this provides strong security against classical computers, it makes SNARKs potentially vulnerable to future quantum computers that could solve the DLP efficiently.

zkSNARK

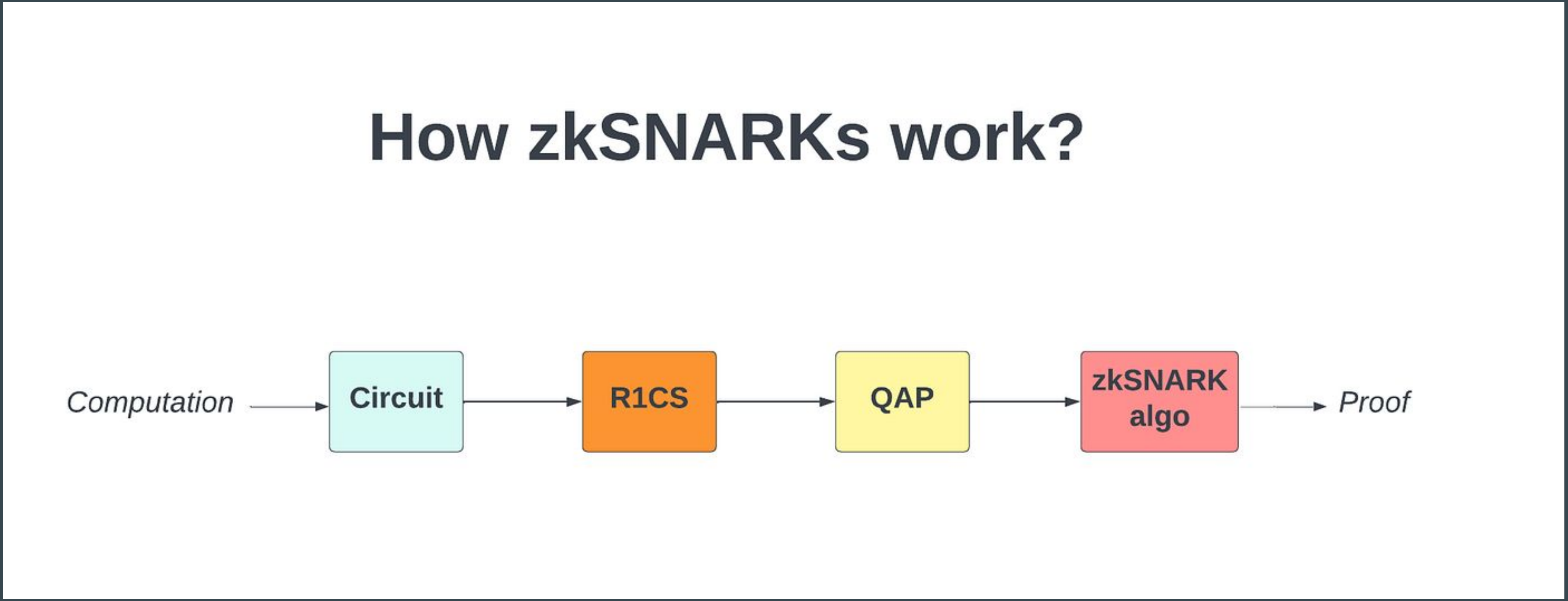
Tricks

Homomorphic Commitments

Bilinear Pairings

Different Polynomials each time

Approach of Early zkSNARK Implementations



Arithmetic Circuit

A model for computing polynomials

- More generally it can be defined as a Directed Acyclic Graph on which at each node of the graph an arithmetic operation is performed.
- The circuit consists of addition gates, multiplication gates and some constant gates.
- In the same way Boolean circuits carry bits in wires, Arithmetic circuits carry integers.

Prover wants to convince the Verifier that he knows a solution to the Arithmetic Circuit.

Commitment

Definition

- To do this, the prover will put all of the values (private and public) associated with the circuit into a commitment.
- Commitments hide their inputs by using a function whose output is irreversible.
- Sha256 is one example of a hashing function that can be used in a commitment scheme.

Commitment

Workflow

- After the prover commits to the values, the commitments are sent to verifier (being confident they are unable to uncover any of the original values).
- The prover is then able to show to the verifier knowledge of each of the values on the nodes of the graph.

Fiat Shamir Transform

Making
Non Interactive Proofs
Possible

- To make the protocol non-interactive, the prover generates randomness (used for the hidden challenge) on behalf of the verifier using a cryptographic hash function.
- This is known as the random oracle.
- The prover can then send a single message to the verifier who can then check it is correct

General Purpose SNARKs

Construction Elements

- Functional commitment scheme: Allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial.
- Polynomial interactive oracle: Verifier asks prover (algorithm) to open all commitments at various points of their choosing using polynomial commitment scheme & checks identity holds true between them.

Types of Pre-Processing Steps

Categories of SNARKs

- Trusted Setup per circuit - Is run once per circuit. Is specific to a circuit & the secret randomness (Common Reference String) must be kept secret + destroyed. A compromised setup in this method means a dishonest prover can prove false statements.
- Trusted but Universal Setup - Only has to run trusted setup once and is able to then deterministically preprocess multiple circuits.
- Transparent Setup (No Trusted Setup)- The preprocessing algorithm does not use any secret randomness at all.

Types of SNARKs

- Groth16 : Requires Trusted Setup but has very short proofs that can be verified quickly.
- Sonic /Marlin /Plonk : Universally Trusted Setup.
- DARK /HALO /STARK : No Trusted Setup but produce slightly longer proofs or may take longer for prover to run.

SNARKs are useful when multiple verifiers are needed such as a blockchain like Zcash or zkRollup such as Aztec so that multiple validating nodes don't have to interact over several rounds with each proof.

Application of Zero Knowledge

zkRollups

This is achieved by 'Outsourcing Computation'. There is no strict need for zero-knowledge for an L1 chain to verify the work of an off-chain service. Transactions are not necessarily private on a zk-EVM.

The advantage of a proof based Rollup (zk-Rollup) service is to process a batch of hundreds/thousands of transactions & the L1 is able to verify a succinct proof that all transactions were processed correctly, scaling the networks transaction throughput by a factor of 100 or 1000.

Application of Zero Knowledge

zkBridges

Interoperability is achieved on a zk-Bridge by 'locking' assets on a source chain and proving to the target chain the assets have been locked (proof of consensus).

Applications of Zero Knowledge

Compliance

Projects such as Espresso are able to prove that a private transaction is compliant with local banking laws without revealing the details of the transaction.

Groth16

Key features

Groth16 is one of the most widely used SNARK protocols.

It requires a circuit-specific trusted setup and is highly efficient, producing very small proofs and fast verification times.

It is commonly used in blockchain projects, such as Zcash, due to its compact proof size.

Groth16

In 2016, Jens Groth introduced a zkSNARK protocol, constructed based on a set of polynomial equations and pairing based cryptography.

The prover computes a number of group elements using generic group operations and the verifier checks the proof using a number of pairing product equations.

Groth16

- In the Groth16 scheme, any program F(x,w) = y with public input x and private witness w is converted into an arithmetic circuit C.
- To prove the correctness of circuit C, we must verify the correctness of each gate. To do this, Groth16 uses Quadratic Arithmetic Languages (QAP) to encode the circuit.

PLONK

Key features

PLONK is defined as Permutation Argument over Lagrange bases for Oecumenical Noninteractive Arguments of Knowledge:

PLONK is a more flexible SNARK protocol that uses a universal and updatable SRS, meaning it can be used for any circuit and can be modified to support larger circuits.

Unlike Groth16, PLONK's setup is not specific to any particular circuit and can be reused for multiple circuits.

This reduces the need for repeated trusted setups and enables easier addition of new programs or circuits without redoing the entire setup.

PLONK

PlonK protocol addresses the one-time and non-unpalatable trusted setup issues by introducing universal and updatable SRS.

Several programs can share CRS, so long as their circuit size does not exceed the CRS threshold limit.

PLONK

- Likewise Groth16, in Plonk every program is converted into an arithmetic circuit of multiplicative and additive gates.
- To prove the correctness of Plonk's circuit, we need to prove the validity of each gate constraint, as well as the gate relationship.
- In addition to the verification of the polynomial equation, a mathematical method of permutation argument is adopted to verify the constraint relationship between the gates, that is, the copy constraint.

zkSTARKs

Definition

Scalable Transparent Arguments of Knowledge (STARK)

They are designed to be scalable and "transparent," meaning they do not require a trusted setup phase.

Instead, zk-STARKs use hash functions and publicly known randomness to construct proofs, which enhances their security and scalability.

zkSTARKs

Key Features

- STARKs do not rely on secret parameters. Instead, their proofs are generated using public randomness,
- Thus there is no “toxic waste” that could compromise the system, and no trusted setup is required.

zkSTARKs

Key Features

- STARKs rely on hash functions, such as SHA-256, instead of elliptic curve cryptography.
- This makes them resistant to quantum attacks, as hash functions are currently considered secure in the presence of quantum computers.

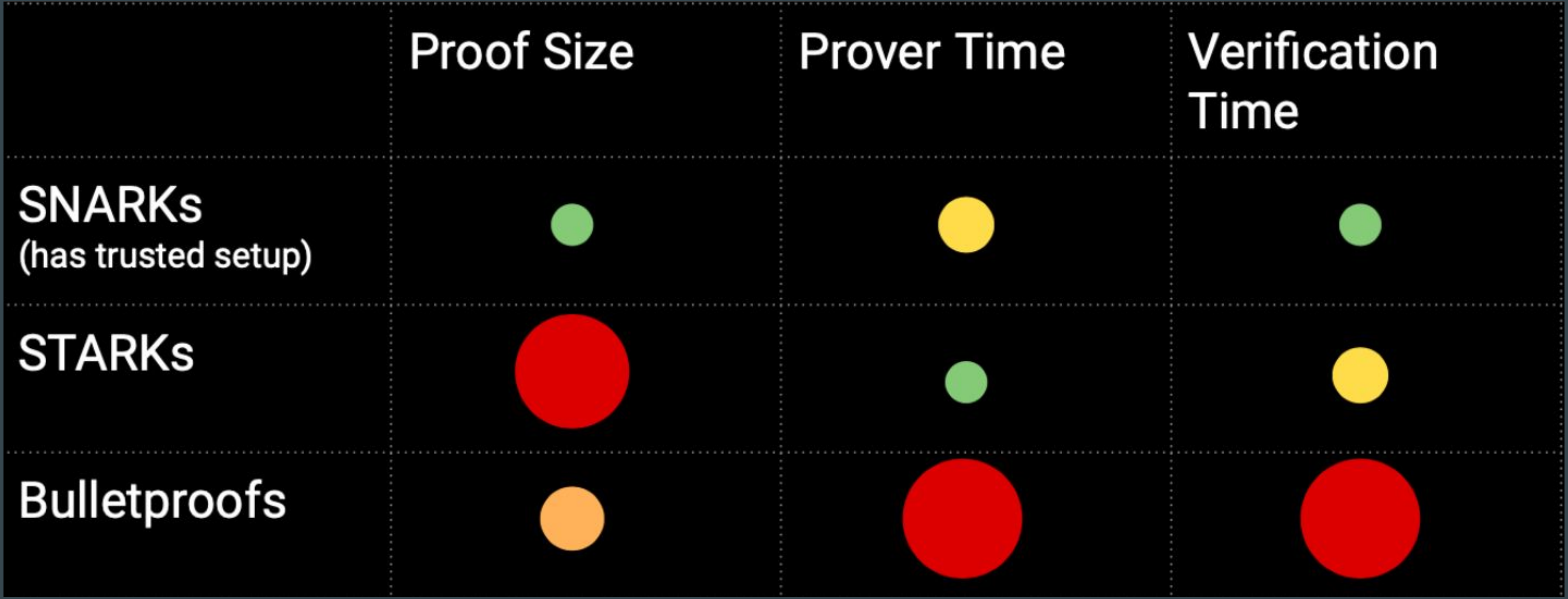
zkSTARKs

Characteristics

STARK proofs can be several times larger than SNARK proofs, which increases verification time and is a disadvantage in environments with limited bandwidth or storage.

This is due to their transparency, use of polynomial commitments, and approach to achieving scalability.

Comparison of Zero Knowledge Proof Systems



	Proof Size	Prover Time	Verification Time
SNARKs (has trusted setup)	🟢	🟡	🟢
STARKs	🔴	🟢	🟡
Bulletproofs	🟠	🔴	🔴

STARK Advantages

However, proofs generated by STARK are much larger.

Lower Gas

Larger Batch Size

Faster Proving

No Trusted Setup

Post Quantum Security

Polygon Hermez uses SNARK to verify the correctness of STARK, thereby reducing the gas fee when the proof is finally settled.

Benchmarks in Zero Knowledge Proofs

Circom + SnarkJS

Proving System

- Circom is a popular DSL for writing circuits and generating R1CS constraints while snarkjs is able to generate Groth16 or Plonk proof for Circom.
- Rapidsnark is also a prover for Circom that generates Groth16 proof and is usually much faster than snarkjs due to the use of the ADX extension, which parallelizes proof generation as much as possible.

gnark

Proving System

gnark is a comprehensive Golang framework from Consensys that supports Groth16, Plonk, and many more advanced features.

Arkworks

A comprehensive rust framework for zkSNARKs

Halo2

Proving System

Halo2 is Zcash's zk-SNARK implementation with Plonk.

It is equipped with the highly-flexible Plonkish arithmetization that supports many useful primitives, such as custom gates and lookup tables.

Ethereum Foundation uses a Halo2 fork with KZG support.

Plonky2

- Plonky2 is a SNARK implementation based on techniques from PLONK and FRI from Polygon Zero.
- Plonky2 uses a small Goldilocks field and supports efficient recursion.

Starky

A highly performant STARK framework from Polygon Zero.

Applications to Blockchains

Blockchain Applications

Confidentiality, Integrity, Availability

- zkRollups
- zkVMs
- zkEVMs
- zkPayments
- zkGames
- zkIdentity

zkRollups

zkRollups

Introduction

Zero-knowledge rollups are layer 2 scaling solutions that increase throughput on Ethereum Mainnet by moving computation and state-storage off-chain

ZK-rollups can process thousands of transactions in a batch and then only post some minimal summary data to Mainnet.

This summary data defines the changes that should be made to the Ethereum state and some cryptographic proof that those changes are correct

What is Layer 2 Scaling on Ethereum?

Blockchain Scalability Approach

A layer 2 is a separate blockchain that extends Ethereum and inherits the security guarantees of Ethereum.

Ethereum also functions as a data availability layer for layer 2s. Layer 2 projects will post their transaction data onto Ethereum, relying on Ethereum for data availability.

This data can be used to get the state of the layer 2, or to dispute transactions on layer 2.

zkRollups in Action

Zero-knowledge rollups (ZK-rollups) bundle (or 'roll up') transactions into batches that are executed off-chain.

Off-chain computation reduces the amount of data that has to be posted to the blockchain.

ZK-rollup operators submit a summary of the changes required to represent all the transactions in a batch rather than sending each transaction individually.

zkRollups in Action

They produce validity proofs to prove the correctness of their changes.

The ZK-rollup's state is maintained by a smart contract deployed on the Ethereum network.

To update this state, ZK-rollup nodes must submit a validity proof for verification.

zkVM

zkVM

Introduction

zkVM is a virtual machine that guarantees secure and verifiable trustworthiness by zero-knowledge proofs.

zkVM is simply the machine that you enter the old state and program, and it return the new state in a trusted manner.

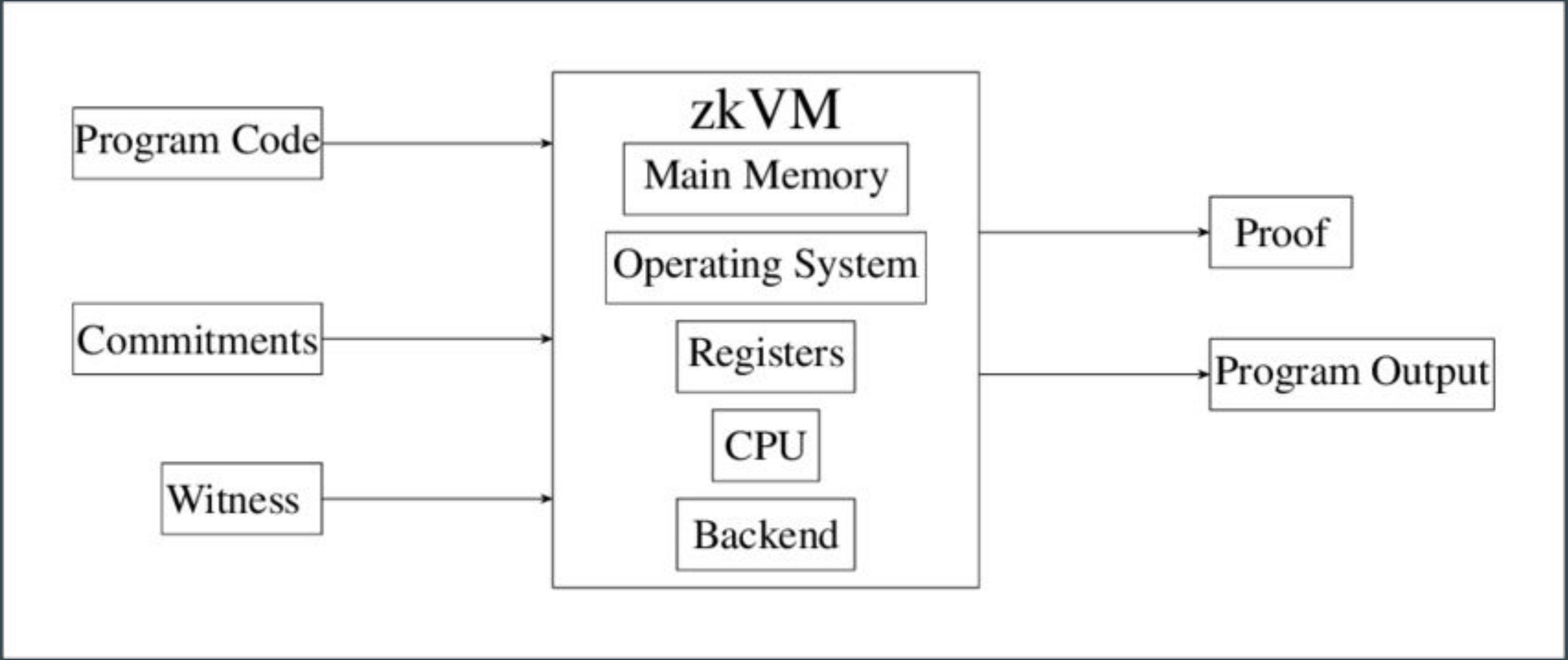
zkVM

Model

Prover runs a public program on private inputs and wants to convince Verifier that the program has executed correctly and produced an asserted output without revealing anything about the input of computation or intermediate state.

As an example, Prover may want to convince the Verifier that they have a password that hashes to a given value without revealing anything about the password.

zkVM General Architecture



zkVM

Advantages

- zkVM can generate proofs for any program or computation.
- A relatively small number of constraints can describe the entire VM
- No need to repeatedly generate the entire VM’s circuit
- Free built-in recursive feature.
- Verification of VMs can be performed by VMs.
- You can put a zkVM inside a zkVM.

zkVM

Types

Mainstream (WASM, RISC-V)

EVM Equivalent(EVM Bytecode)

zk-Optimized(New Instruction Set)

zkVM

Types

RISC-V based - Risk0

WASM based - zkWASM

zkVM

Configurability

Need to build LLVM for ZK where a higher description of a command set can be used to auto-generate both the prover / verifier components and LLVM backend targeting the command set with minimal effort.

zkSNARKS

What is zkSNARK

Definition

Succinct

Non Interactive

ARgument

Knowledge

What are zkSNARKS?

A cryptographic tool that can efficiently generate a zero-knowledge protocol for any problem or function.

- zk: hides inputs
- Succinct: generates short proofs that can be verified quickly
- Noninteractive: doesn’t require a back-and-forth
- ARgument of Knowledge: proves you know the input

zkSNARKs

Benefits

- Easy to verify
- Protect completeness of computation
- Do not bring too much overhead to non-participants of the proof

The size of messages are tiny in comparison to the length of the actual computation

There is no or only little interaction between the prover and verifier.

Deep Dive

Non Interactive Property

For some of the zkSNARks there is a setup phase, and after that a single message from the prover to the verifier.

Furthermore, SNARKs often have the public verifier property meaning that anyone can verify without interacting anew which is important for blockchains.

Deep Dive

Arguments

The verifier is only protected against computationally limited provers.

Provers with enough computational power can create proofs/arguments about wrong statements.

This is known as computational soundness as opposed to perfect soundness.

Deep Dive

Knowledge

It is not possible for the prover to construct a proof/argument without knowing a certain witness (for example address he/she wants to spend from, the preimage of the hash function or the path to a certain Merkles Tree node)

Deep Dive

Zero Knowledge

During the limited interaction, the verifier learns nothing apart from the validity of the statement.

The verifier does not learn the witness string.

Foundations

Fields

Mathematical Foundations

A field is just a set of numbers equipped with two binary operations (addition and multiplication) satisfying certain properties (namely, associativity, commutativity, distributivity, existence of identity and inverses, for both operations).

Common examples of fields are the set of rational numbers (Q), real numbers (R), and finite fields with p elements (Fp).

Finite Fields

Mathematical Foundations

Finite fields (also known as Galois fields) are used in cryptography extensively, with the number of elements being referred to as their order.

The field Fp consists of the first p whole numbers: {0, 1, ... (p-1)}.

Basic operations like addition, subtraction, and multiplication can be defined defined as with whole numbers but with the modulo p operation so that the answer remains within the set.

The division is defined a little differently to ensure the output remains in the field.

Groups

Mathematical Foundations

A group is a set G equipped with a binary operation (addition or multiplication) that satisfies closure, associativity, having an identity element e, and an inverse element for each element.

Cyclic Groups

Mathematical Foundations

A group G is called a cyclic group if there exists an element $g \in G$ (called a generator) such that every element in G can be written as g^n (i.e., g multiplied n times in case of a multiplicative group) or $n \cdot g$ (in case of an additive group) for some integer n.

Abelian Groups

Mathematical Foundations

All cyclic groups are Abelian (i.e. the operation is commutative).

The multiplicative group formed by any finite-field Fp is cyclic.

The additive group formed by any finite field with a prime order is also cyclic.

Lagrange Theorem

Mathematical Foundations

Lagrange's theorem states that if G is a finite group and H is a subgroup of G, then the order (number of elements) of H divides the order of G.

The theorem also shows that any group of prime order is cyclic and simple, since the subgroup generated by any non-identity element must be the whole group itself.

zkSNARKs construction

Consider the transaction validation computation - **F(h1, h2, S,R,P_S,P_R, V) = 1**

- where h1, h2 are the root hashes of account Merkle Trees (pre- and post-state)
- S and R are sender and receiver accounts. P_S and P_R are Merkle Tree Proofs.
- V is the threshold in the test of the Merkle Proofs.
- When turning **F** into a zkSNARK where h1 and h2 are publicly known, then (S,R,P_S,P_R,V) is the witness string.

Early Implementations of zkSNARKs

Four Main Ingredients

- Encoding as a polynomial problem
- Succinctness by random sampling
- Homomorphic encryption
- Zero knowledge

Early zkSNARKs implementations

A polynomial problem

Program that is to be checked is compiled into a quadratic equation of polynomials $t(x)h(x) = w(x)v(x)$

The equality holds iff the program is computed correctly

The prover wants to convince the verifier that the equality holds

Early zkSNARKs implementations

Succinctness by random sampling

The verifier choses a secret evaluation point S to reduce the problem from multiplying polynomials and verifying polynomial function equality to simple multiplication check on numbers.

$t(S)h(S) = w(S)v(S)$

This reduces both the proof size and verification time tremendously

Early zkSNARKs implementations

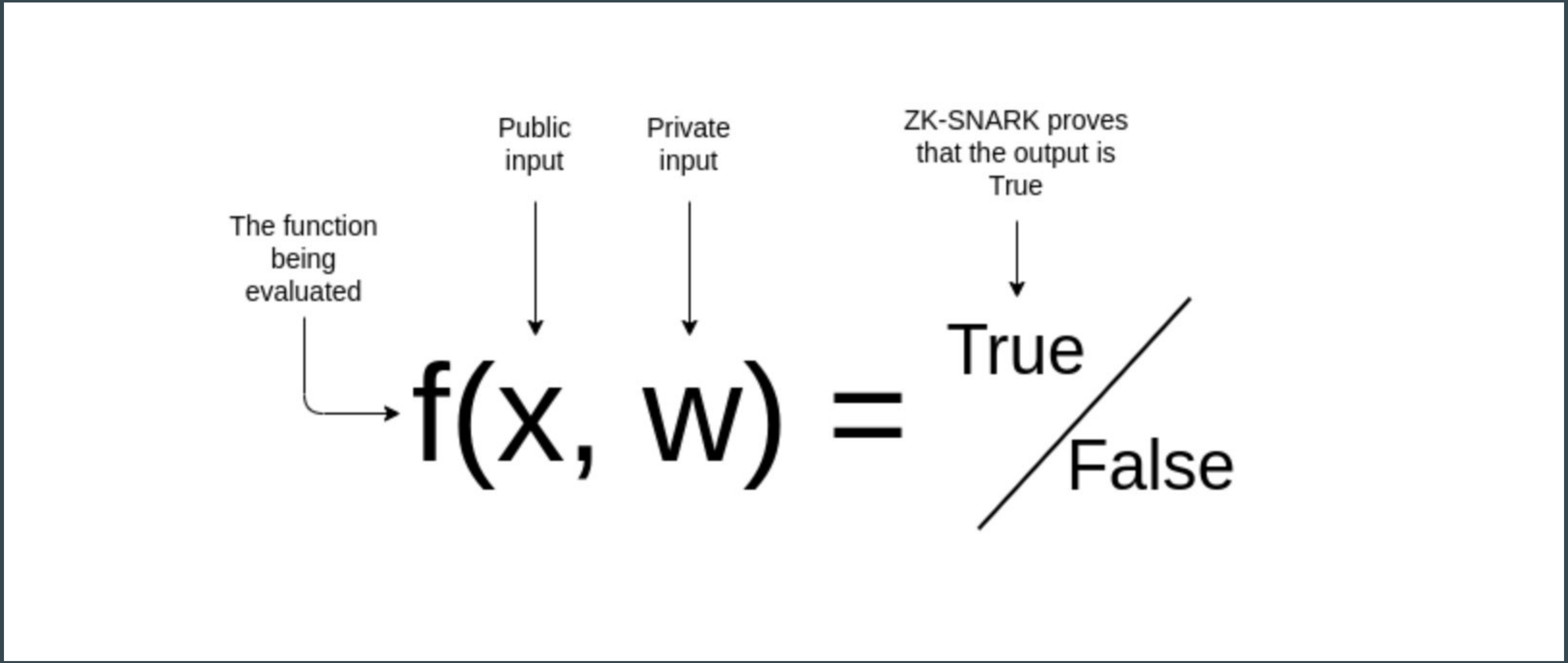
Homomorphic Encryption

The encryption function is used that has some homomorphic properties

This allows the prover to compute E(t(S)), E(h(S)), E(w(S)), E(v(S)) without knowing S.

The prover only knows E(S) and some other helpful encrypted values

What are zkSNARKS?


Public input
Private input
ZK-SNARK proves that the output is True
The function being evaluated
f(x, w) = True / False

How to construct zkSNARKS?

High Level Idea

- Turn your problem (graph isomorphism, discrete log, etc.) into a function whose inputs you want to hide.
- Turn that function into an equivalent set of "R1CS" (or other) equations
  - Basically, an arithmetic circuit - a bunch of + and * operations on prime field elements
  - Simplified: equations of the form x_i + x_j = x_k, or x_i * x_j = x_k
- Generate a ZKP for satisfiability of the constraint system(R1CS).

zkSNARKs construction

- Function inputs: x1, x2, x3, x4
- OUT = f(x) = (x1 + x2) * x3 - x4

- zkSNARK:
  a. I know some secret (x1, x2, x3, x4) such that the result of this computation is OUT.
  b. It is a signature that proves that I know such a tuple, without telling what the tuple actually is.

zkSNARKS construction

- Function inputs: x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 * x3
- OUT := y2 - x4

- SNARK prover inputs: x1, x2, x3, x4, y1, y2, OUT
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
  - y1 == x1 + x2
  - y2 == y1 * x3
  - y2 == OUT + x4

zkSNARKS construction

- Function inputs: 02, 04, 08, 05
- 06 := 02 + 04
- 48 := 06 * 08
- 043 := 48 - 05

- SNARK prover inputs: 02, 04, 08, 05, 06, 48, 043
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
  - 06 == 02 + 04
  - 48 == 06 * 08
  - 48 == 043 + 05

zkSNARKS construction

- Function inputs: x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 * x3
- 043 := y2 - x4

- SNARK prover inputs: x1, x2, x3, x4, y1, y2, 043
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
    - y1 == x1 + x2
    - y2 == y1 * x3
    - y2 == 043 + x4

zkSNARKS prove constraints (only + and *)

- Function inputs: x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 / x3
- OUT := y2 - x4

- SNARK prover inputs: x1, x2, x3, x4, y1, y2, OUT
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
    - y1 == x1 + x2
    - y1 == y2 * x3
    - y2 == OUT + x4