Zero-knowledge proofs and Blockchains

Youssef El Housni

UM6P Benguerir — April 24, 2025

consensys

Linea

whoami

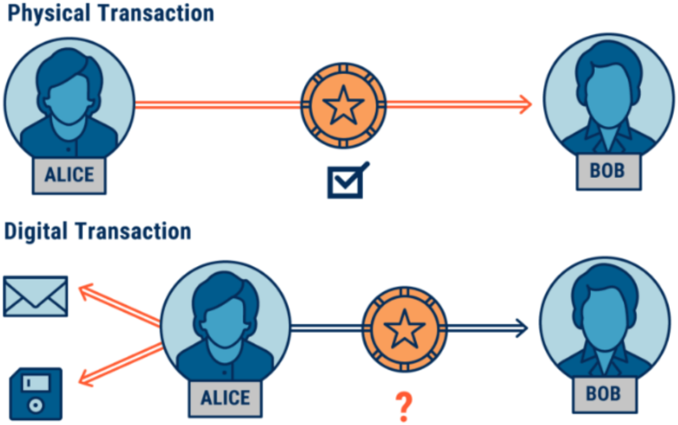
- PhD in cryptography — Ecole Polytechnique (Paris)
- Cryptographer — Consensys (New York)
- Co-founder of gnark
- Co-founder of linea

Overview

Overview

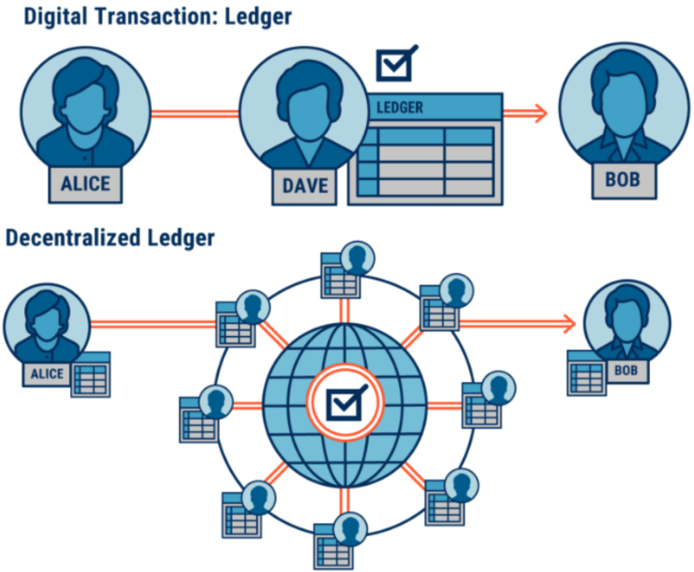
1. Motivation
2. Blockchain
3. Zero-knowledge proofs
4. Applications
5. Research

The story of Alice and Bob



(Courtesy of CBINSIGHTS)

The story of Alice and Bob



Overview

1. Motivation
2. Blockchain
3. Zero-knowledge proofs
4. Applications
5. Research

Blockchains

1991

Stuart Haber and W. Scott Stornetta published the first paper to outline the use of a chain of cryptographically secured blocks to preserve the integrity of past information and protect it.

1991

There's too much spam! The concept of proof-of-work was established, seeking to provide countermeasures to spam and other network abuses.

2008

Satoshi Nakamoto released a white paper that ushered in the age of blockchain. The paper was called: ***Bitcoin: A Peer-to-Peer Electronic Cash System***. This paper defined the most commonly known blockchain of today: Bitcoin.

2014

Ethereum is born! It is a distributed world computer that runs on a blockchain. Ethereum is a blockchain that can also be programmed and run computations. Ethereum's world computer is called the *"Ethereum Virtual Machine"*, or EVM.

2015

Bitcoin starts getting serious attention. Harvard Business Review stated *"bitcoin is backed by a widely distributed network"*.

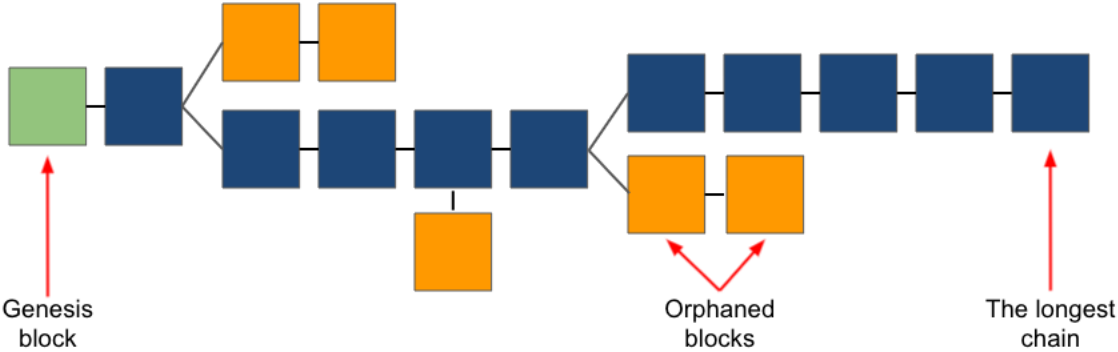
2016

Blockchain embraced by financial industry. JPMorgan Chase, led by CEO Jamie Dimon, is building a new system based on the Ethereum platform.

2017

Blockchain named a "foundational technology".

Blockchains



Blockchains

- How is a tx included in a block?
- How is the longest chain is agreed upon?

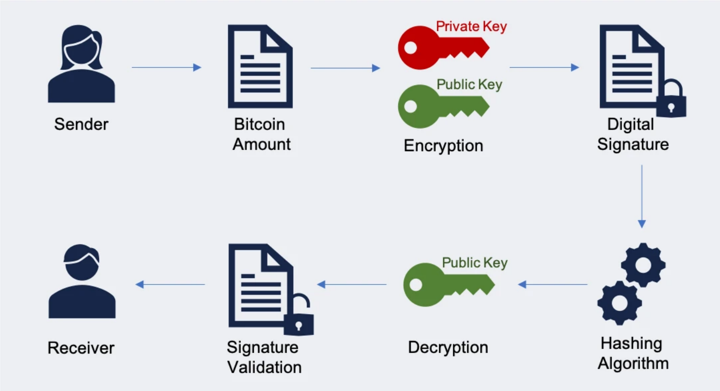
Blockchains

- How is a tx included in a block?
    - Signatures verification (Bitcoin: ECDSA/Schnorr, Ethereum: ECDSA/BLS)
- How is the longest chain is agreed upon?
    - Consensus (Bitcoin: proof-of-work, Ethereum: proof-of-stake)

Blockchains

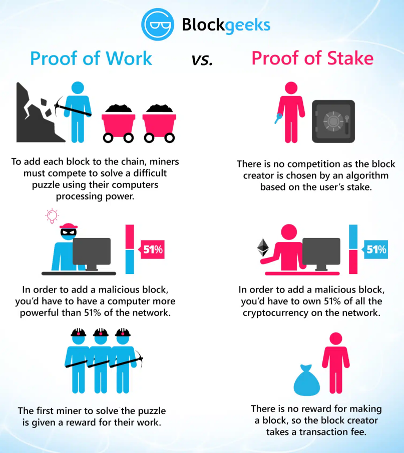
Digital signatures:

- Public-key cryptography: Shcnorr but patented until 2020
- ECDSA as a workaround but with caveats
- Ethereum chooses BLS for aggregation and Bitcoin Schnorr for simplicity



Blockchains

Consensus:



Blockgeeks

Proof of Work vs. Proof of Stake

To add each block to the chain, miners must compete to solve a difficult puzzle using their computers processing power.

There is no competition as the block creator is chosen by an algorithm based on the user's stake.

In order to add a malicious block, you'd have to have a computer more powerful than 51% of the network.

In order to add a malicious block, you'd have to own 51% of all the cryptocurrency on the network.

The first miner to solve the puzzle is given a reward for their work.

There is no reward for making a block, so the block creator takes a transaction fee.

Examples: proof-of-work, proof-of-stake, proof-of-space, proof-of-authority, proof-of-burn...

Blockchains

A blockchain is a public peer-to-peer *decentralized*, *transparent, immutable, paying* ledger.

- *Transparent*: everything is visible to everyone
- *Immutable*: nothing can be removed once written
- *Paying*: everyone should pay a fee to use

Blockchains

A blockchain is a public peer-to-peer *decentralized*, *transparent, immutable, paying* ledger.

- *Transparent*: everything is visible to everyone
- *Immutable*: nothing can be removed once written
- *Paying*: everyone should pay a fee to use

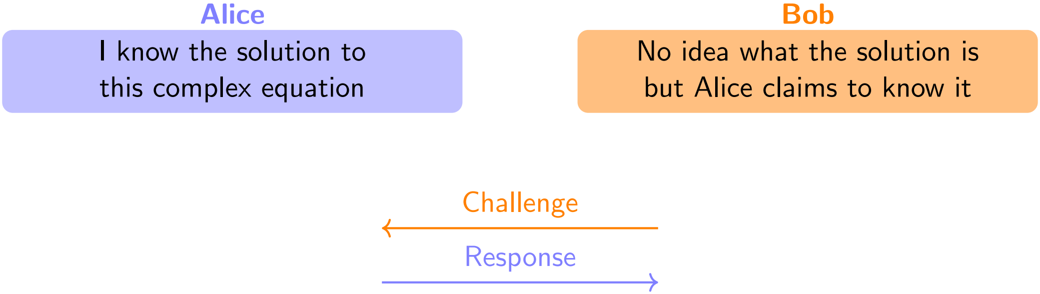
$$\text{Transparent} \xrightarrow[\text{Problem}]{} \text{confidentiality} \qquad \xrightarrow[\text{Solution}]{} ?$$

$$\text{Immutable} \xrightarrow[\text{Problem}]{} \text{scalability} \qquad \xrightarrow[\text{Solution}]{} ?$$

$$\text{Paying} \xrightarrow[\text{Problem}]{} \text{cost} \qquad \xrightarrow[\text{Solution}]{} ?$$

Overview

Zero-knowledge proofs (ZKP)



Alice

I know the solution to
this complex equation

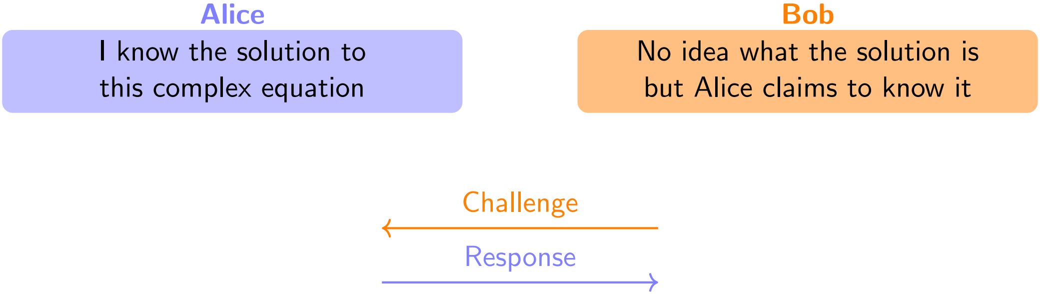
Bob

No idea what the solution is
but Alice claims to know it

Challenge

Response

Zero-knowledge proofs (ZKP)



Alice

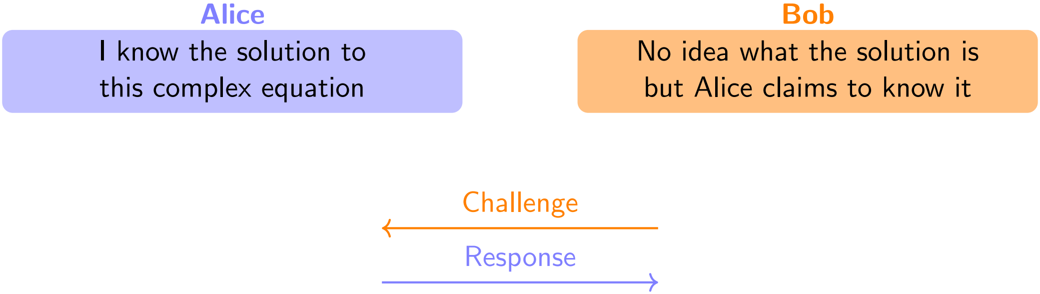
I know the solution to this complex equation

Bob

No idea what the solution is but Alice claims to know it

- **Sound**: **Alice** has a wrong solution $\implies$ **Bob** is not convinced.

Zero-knowledge proofs (ZKP)



Alice

I know the solution to
this complex equation

Bob

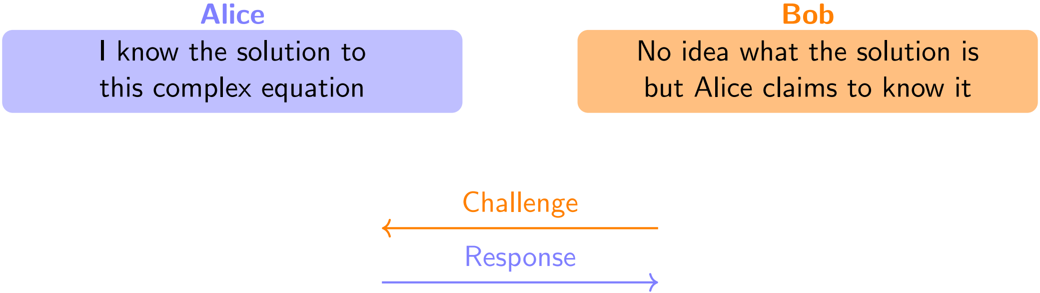
No idea what the solution is
but Alice claims to know it

Challenge

Response

- **Sound**: **Alice** has a wrong solution $\implies$ **Bob** is not convinced.
- **Complete**: **Alice** has the solution $\implies$ **Bob** is convinced.

Zero-knowledge proofs (ZKP)



Alice

I know the solution to
this complex equation

Bob

No idea what the solution is
but Alice claims to know it

Challenge

Response

- **Sound**: **Alice** has a wrong solution $\implies$ **Bob** is not convinced.
- **Complete**: **Alice** has the solution $\implies$ **Bob** is convinced.
- **Zero-knowledge**: **Bob** does NOT learn the solution.

Toy example

?

Example: Sigma protocol

Alice

Bob

I know x such that $g^x = y$

Example: Sigma protocol

Alice **Bob**

I know x such that $g^x = y$

$n \xleftarrow{\$} \mathbb{Z}_r$ $\xrightarrow{A = g^n}$

Example: Sigma protocol

Alice **Bob**

I know x such that $g^x = y$

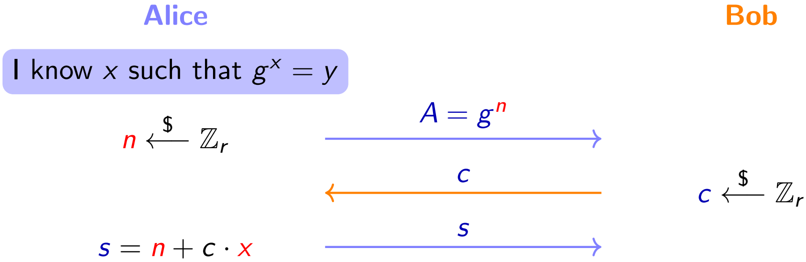
$n \xleftarrow{\$} \mathbb{Z}_r$

$$\xrightarrow{A = g^n}$$

$$\xleftarrow{c}$$

$c \xleftarrow{\$} \mathbb{Z}_r$

Example: Sigma protocol



Alice

Bob

I know x such that $g^x = y$

$n \xleftarrow{\$} \mathbb{Z}_r$ $\xrightarrow{A = g^n}$

$\xleftarrow{c}$ $c \xleftarrow{\$} \mathbb{Z}_r$

$s = n + c \cdot x$ $\xrightarrow{s}$

Example: Sigma protocol

Alice **Bob**

I know x such that $g^x = y$

$n \xleftarrow{\$} \mathbb{Z}_r$ $\xrightarrow{A = g^n}$

$\xleftarrow{c}$ $c \xleftarrow{\$} \mathbb{Z}_r$

$s = n + c \cdot x$ $\xrightarrow{s}$ $g^s \stackrel{?}{=} A \cdot y^c$

with $A \cdot y^c = g^n \cdot g^{x \cdot c}$

then $g^n \cdot g^{x \cdot c} = g^{n + x \cdot c}$

Non-Interactive Zero-Knowledge (NIZK) Sigma protocol

Alice

Bob

I know x such that $g^x = y$

$$n \xleftarrow{\$} \mathbb{Z}_r$$
$$A = g^n$$
$$c = H(A, y)$$
$$s = n + c \cdot x$$

$$\xrightarrow{\pi = (A, c, s)}$$

$$g^s \overset{?}{=} A \cdot y^c$$
$$c \overset{?}{=} H(A, y)$$

Non-Interactive Zero-Knowledge (NIZK) Sigma protocol

Alice

Bob

I know x such that $g^x = y$

$$n \xleftarrow{\$} \mathbb{Z}_r$$

$$\underbrace{g}_{\text{Setup}} \; ; \; A = g^{n}$$

$$\underbrace{\begin{gathered} c = H(A, y) \\ s = n + c \cdot x \end{gathered}}_{\text{Prove}}$$

$$\xrightarrow{\underbrace{\pi = (A, c, s)}_{\text{proof}}}$$

$$\underbrace{\begin{gathered} g^{s} \overset{?}{=} A \cdot y^{c} \\ c \overset{?}{=} H(A, y) \end{gathered}}_{\text{Verify}}$$

ZKP families

Expressivity

- *specific* statement vs. *general* statement

ZKP families

Expressivity

- *specific* statement vs. *general* statement

Deployability

- *interactive* vs. $non - interactive$ protocol
- *trapdoored* setup vs. *transparent* setup
- *Designated* verifier vs. *any* verifier

ZKP families

Expressivity

- *specific* statement vs. *general* statement

Deployability

- *interactive* vs. $non - interactive$ protocol
- *trapdoored* setup vs. *transparent* setup
- *Designated* verifier vs. *any* verifier

Complexity

- prover complexity (Alice)
- verifier complexity (Bob)
- communication complexity (size of the proof and the setup)

ZKP families

Expressivity

- *specific* statement vs. *general* statement

Deployability

- *interactive* vs. $non - interactive$ protocol
- *trapdoored* setup vs. *transparent* setup
- *Designated* verifier vs. *any* verifier

Complexity

- prover complexity (Alice)
- verifier complexity (Bob)
- communication complexity (size of the proof and the setup)

Security

- Cryptographic assumptions
- Cryptographic primitives

Blockchains and ZKP

A blockchain is a public peer-to-peer *decentralized*, *transparent, immutable, paying* ledger.

- *Transparent*: everything is visible to everyone
- *Immutable*: nothing can be removed once written
- *Paying*: everyone should pay a fee to use

Transparent $\xrightarrow[\text{Problem}]{}$ confidentiality $\xrightarrow[\text{Solution}]{}$ *ZKP*

setup, prover?, verifier?

Immutable $\xrightarrow[\text{Problem}]{}$ scalability $\xrightarrow[\text{Solution}]{}$ *ZKP*

Communication complexity

Paying $\xrightarrow[\text{Problem}]{}$ cost $\xrightarrow[\text{Solution}]{}$ *ZKP*

Verifier complexity, *prover*?

ZKP literature landmarks

- First ZKP work [GMR85]
- Non-Interactive ZKP [BFM88]
- Succinct ZKP [Kil92]
- Succinct Non-Interactive ZKP [Mic94]

ZKP literature landmarks

- First ZKP work [GMR85]
- Non-Interactive ZKP [BFM88]
- Succinct ZKP [Kil92]
- Succinct Non-Interactive ZKP [Mic94]
- Pairing-based succinct NIZK [Gro10]

ZKP literature landmarks

- First ZKP work [GMR85]
- Non-Interactive ZKP [BFM88]
- Succinct ZKP [Kil92]
- Succinct Non-Interactive ZKP [Mic94]
- Pairing-based succinct NIZK [Gro10]
- “SNARK” terminology and characterization of existence [BCCT12]
- Pairing-based SNARK in quasi-linear prover time [GGPR13]
- Pairing-based SNARK with shortest proof and verifier time [Gro16]

ZKP literature landmarks

- First ZKP work [GMR85]
- Non-Interactive ZKP [BFM88]
- Succinct ZKP [Kil92]
- Succinct Non-Interactive ZKP [Mic94]
- Pairing-based succinct NIZK [Gro10]
- “SNARK” terminology and characterization of existence [BCCT12]
- Pairing-based SNARK in quasi-linear prover time [GGPR13]
- Pairing-based SNARK with shortest proof and verifier time [Gro16]
- SNARK with universal and updatable setup [GKM$^+$18, MBKM19, GWC19, CHM$^+$20]

What is a zero-knowledge proof?

"I have a *sound*, *complete* and *zero-knowledge* proof that a statement is true" [GMR85].

Sound

False statement $\implies$ cheating prover cannot convince honest verifier.

Complete

True statement $\implies$ honest prover convinces honest verifier.

Zero-knowledge

True statement $\implies$ verifier learns nothing other than statement is true.

zk-SNARK: Zero-Knowledge Succinct Non-interactive ARgument of Knowledge

"I have a *computationally sound*, *complete*, *zero-knowledge*, **succinct**, **non-interactive** proof that a statement is true and that I know a related secret".

Succinct

A proof is very "short" and "easy" to verify.

Non-interactive

No interaction between the prover and verifier for proof generation and verification (except the proof message).

ARgument of Knowledge

Honest verifier is convinced that a computationally bounded prover knows a secret information.

Preprocessing zk-SNARK for NP language

F: public `NP` program, x, z: public inputs, w: private input

$$z := F(x, w)$$

Preprocessing zk-SNARK for NP language

F: public `NP` program, x, z: public inputs, w: private input

$$z := F(x, w)$$

A zk-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

$$\textit{Setup}: \quad (pk, vk) \quad \leftarrow \quad S(F, 1^{\lambda})$$

Preprocessing zk-SNARK for NP language

F: public $\mathtt{NP}$ program, x, z: public inputs, w: private input

$$z := F(x, w)$$

A zk-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

$$\begin{aligned} \mathit{Setup}: \quad & (pk, vk) & \leftarrow \quad & S(F, 1^{\lambda}) \\ \mathit{Prove}: \quad & \pi & \leftarrow \quad & P(x, z, w, pk) \end{aligned}$$

Preprocessing zk-SNARK for NP language

F: public $\mathtt{NP}$ program, x, z: public inputs, w: private input

$$z := F(x, w)$$

A zk-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

$$\begin{aligned} \mathit{Setup}: &\quad (pk, vk) &\leftarrow&\quad S(F, 1^{\lambda}) \\ \mathit{Prove}: &\quad \pi &\leftarrow&\quad P(x, z, w, pk) \\ \mathit{Verify}: &\quad \texttt{false/true} &\leftarrow&\quad V(x, z, \pi, vk) \end{aligned}$$

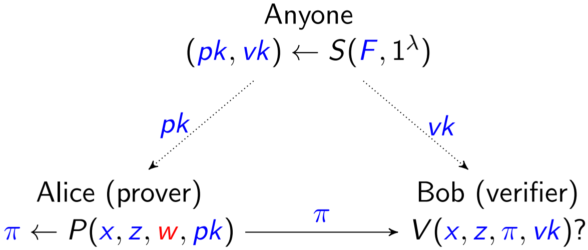
Preprocessing zk-SNARK for NP language

F: public $\mathtt{NP}$ program, x, z: public inputs, w: private input

$$z := F(x, w)$$

A zk-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

$$\begin{array}{llll} \mathit{Setup}: & (pk, vk) & \leftarrow & S(F, 1^\lambda) \\ \mathit{Prove}: & \pi & \leftarrow & P(x, z, w, pk) \\ \mathit{Verify}: & \mathtt{false/true} & \leftarrow & V(x, z, \pi, vk) \end{array}$$



Anyone

$(pk, vk) \leftarrow S(F, 1^\lambda)$

pk

vk

Alice (prover)

$\pi \leftarrow P(x, z, w, pk) \xrightarrow{\pi}$

Bob (verifier)

$V(x, z, \pi, vk)$?

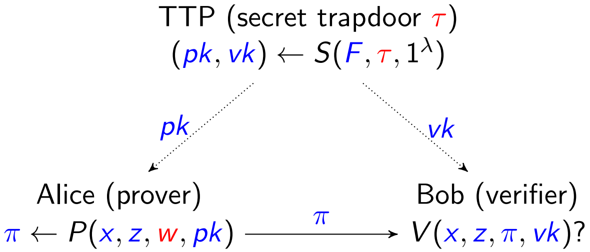
(Trapdoored) preprocessing zk-SNARK for NP language

F: public NP program, x, z: public inputs, w: private input

$$z := F(x, w)$$

A zk-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

$$\begin{aligned} \mathit{Setup}: &\quad (pk, vk) &\leftarrow&\quad S(F, \tau, 1^\lambda) \\ \mathit{Prove}: &\quad \pi &\leftarrow&\quad P(x, z, w, pk) \\ \mathit{Verify}: &\quad \texttt{false/true} &\leftarrow&\quad V(x, z, \pi, vk) \end{aligned}$$



TTP (secret trapdoor τ)

$(pk, vk) \leftarrow S(F, \tau, 1^\lambda)$

Alice (prover)

$\pi \leftarrow P(x, z, w, pk) \xrightarrow{\pi}$

Bob (verifier)

$V(x, z, \pi, vk)$?

(Trapdoored) preprocessing zk-SNARK for NP language

F: public `NP` program, x, z: public inputs, w: private input

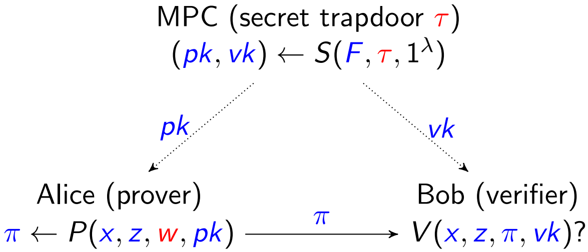
$$z := F(x, w)$$

A zk-SNARK consists of algorithms S, P, V s.t. for a security parameter λ:

Setup : $(pk, vk) \leftarrow S(F, \tau, 1^\lambda)$

Prove : $\pi \leftarrow P(x, z, w, pk)$

Verify : `false/true` $\leftarrow V(x, z, \pi, vk)$



MPC (secret trapdoor τ)

$(pk, vk) \leftarrow S(F, \tau, 1^\lambda)$

Alice (prover)

$\pi \leftarrow P(x, z, w, pk)$

Bob (verifier)

$V(x, z, \pi, vk)$?

zk-SNARK

Succinctness: A proof is very "short" and "easy" to verify.

Definition [BCTV14b]

A succinct proof π has size $O_\lambda(1)$ and can be verified in time $O_\lambda(|F| + |x| + |z|)$, where $O_\lambda(.)$ is some polynomial in the security parameter λ.

zk-SNARKs in a nutshell

Main ideas:

zk-SNARKs in a nutshell

Main ideas:

1. Reduce a ”general statement” satisfiability to a polynomial equation satisfiability.

zk-SNARKs in a nutshell

Main ideas:

1. Reduce a "general statement" satisfiability to a polynomial equation satisfiability.
2. Use Schwartz-Zippel lemma to succinctly verify the polynomial equation with high probability.

zk-SNARKs in a nutshell

Main ideas:

1. Reduce a ”general statement” satisfiability to a polynomial equation satisfiability.
2. Use Schwartz-Zippel lemma to succinctly verify the polynomial equation with high probability.
3. Use homomorphic hiding cryptography to blindly verify the polynomial equation.

zk-SNARKs in a nutshell

Main ideas:

1. Reduce a "general statement" satisfiability to a polynomial equation satisfiability.
2. Use Schwartz-Zippel lemma to succinctly verify the polynomial equation with high probability.
3. Use homomorphic hiding cryptography to blindly verify the polynomial equation.
4. Make the protocol non-interactive.

Arithmetization

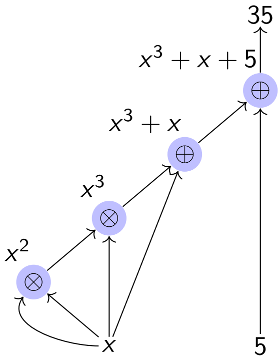
Statement → **Arithmetic circuit** → Intermediate representation → Polynomial identities → zk-SNARK proof

$$x^3 + x + 5 = 35 \qquad ({\color{red}x = 3})$$

Arithmetization

Statement → **Arithmetic circuit** → Intermediate representation → Polynomial identities → zk-SNARK proof

$$x^3 + x + 5 = 35 \qquad ({\color{red}x = 3})$$



Arithmetization

e.g. R1CS

Statement → Arithmetic circuit → **Intermediate representation** → Polynomial identities → zk-SNARK proof

$$L = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$O = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

witness:

$$\begin{aligned} \vec{w} &= \begin{pmatrix} \text{one} & x & d & a & b & c \end{pmatrix} \\ &= \begin{pmatrix} 1 & 3 & 35 & 9 & 27 & 30 \end{pmatrix} \end{aligned}$$

$$O \bullet \vec{w} = L \bullet \vec{w} \cdot R \bullet \vec{w}$$

Arithmetization

e.g. Quadratic Arithmetic Program

Statement → Arithmetic circuit → Intermediate representation → **Polynomial identities** → zk-SNARK proof

$$L(X)R(X) - O(X) = H(X)T(X) \qquad (QAP \in \mathbb{F}[X])$$

Arithmetization

e.g. Quadratic Arithmetic Program

Statement → Arithmetic circuit → Intermediate representation → **Polynomial identities** → zk-SNARK proof

$$L(X)R(X) - O(X) = H(X)T(X) \qquad (QAP \in \mathbb{F}[X])$$

$$L(\tau)R(\tau) - O(\tau) = H(\tau)T(\tau) \qquad (trapdoor\ \tau \xleftarrow{\$} \mathbb{F})$$

Arithmetization

e.g. Quadratic Arithmetic Program

Statement → Arithmetic circuit → Intermediate representation → **Polynomial identities** → zk-SNARK proof

$$\begin{aligned} L(X)R(X) - O(X) &= H(X)T(X) && (QAP \in \mathbb{F}[X]) \\ L(\tau)R(\tau) - O(\tau) &= H(\tau)T(\tau) && (trapdoor\ \tau \overset{\$}{\leftarrow} \mathbb{F}) \\ C(L(\tau)R(\tau) - O(\tau)) &= C(H(\tau)T(\tau)) && (Homomorphic\ commitment) \end{aligned}$$

Succinct evaluation of polynomials

Instead of verifying the QAP on the whole domain $\mathbb{F}$ → verify it in a single random point $\tau \in \mathbb{F}$.

Schwartz–Zippel lemma

Any two distinct polynomials of degree d over a field $\mathbb{F}$ can agree on at most a $d/|\mathbb{F}|$ fraction of the points in $\mathbb{F}$.

Blind evaluation of polynomials

Statement → Arithmetic circuit → Intermediate representation → Polynomial identities → **zk-SNARK proof**

Let's take the example of polynomial L:

Blind evaluation of polynomials

Statement → Arithmetic circuit → Intermediate representation → Polynomial identities → **zk-SNARK proof**

Let's take the example of polynomial L:

- Alice can send L to Bob and he computes $L(\tau)$ → breaks the zero-knowledge.

Blind evaluation of polynomials

Statement → Arithmetic circuit → Intermediate representation → Polynomial identities → **zk-SNARK proof**

Let's take the example of polynomial L:

- Alice can send L to Bob and he computes $L(\tau)$ → breaks the zero-knowledge.
- Bob can send τ to Alice and she computes $L(\tau)$ → breaks the soundness.

Blind evaluation of polynomials

Statement → Arithmetic circuit → Intermediate representation → Polynomial identities → **zk-SNARK proof**

Let's take the example of polynomial L:

- Alice can send L to Bob and he computes $L(\tau)$ → breaks the zero-knowledge.
- Bob can send τ to Alice and she computes $L(\tau)$ → breaks the soundness.

$\Longrightarrow$ homomorphic cryptography to evaluate $L(X)$ at τ without Bob learning L nor Alice learning τ.

Blind evaluation of polynomials

$$L(\tau) = l_0 + l_1\tau + l_2\tau^2 + \cdots + l_d\tau^d \in \mathbb{F}$$
$$C(L(\tau)) = l_0 C(1) + l_1 C(\tau) + l_2 C(\tau^2) + \cdots + l_d C(\tau^d)$$

Somewhat homomorphic commitment w.r.t.:

- depth-d **additions** (arbitrary d)
- depth-1 **multiplications** (for $L(\tau) \cdot R(\tau)$ and $H(\tau) \cdot T(\tau)$).

Blind evaluation of polynomials

Somewhat homomorphic commitment w.r.t.:

- depth-d **additions** (arbitrary d)

Blind evaluation of polynomials

Somewhat homomorphic commitment w.r.t.:

- depth-d **additions** (arbitrary d)

$$C(\tau) = \tau G \qquad (DL)$$

$$L(\tau)G = l_0 G + l_1 \tau G + l_2 \tau^2 G + \cdots + l_d \tau^d G$$

Blind evaluation of polynomials

Somewhat homomorphic commitment w.r.t.:

- depth-d **additions** (arbitrary d)

$$C(\tau) = \tau G \qquad (DL)$$
$$L(\tau)G = l_0 G + l_1 \tau G + l_2 \tau^2 G + \cdots + l_d \tau^d G$$

- depth-1 **multiplications** (for $L(\tau) \cdot R(\tau)$ and $H(\tau) \cdot T(\tau)$).

$$C(\tau_1) = \tau_1 G;\ C(\tau_2) = \tau_2 G$$
$$C(\tau_1) \cdot C(\tau_2) = C(\tau_1 \cdot \tau_2) \qquad (?)$$

Blind evaluation of polynomials

Somewhat homomorphic commitment w.r.t.:

- depth-d **additions** (arbitrary d)

$$C(\tau) = \tau G \qquad (DL)$$

$$L(\tau)G = l_0 G + l_1 \tau G + l_2 \tau^2 G + \cdots + l_d \tau^d G$$

- depth-1 **multiplications** (for $L(\tau) \cdot R(\tau)$ and $H(\tau) \cdot T(\tau)$).

$$C(\tau_1) = \tau_1 G;\ C(\tau_2) = \tau_2 G$$

$$C(\tau_1) \cdot C(\tau_2) = C(\tau_1 \cdot \tau_2) \qquad (?)$$

$$\underbrace{e(C(\tau_1), C(\tau_2))}_{\text{product of commitments}} = \underbrace{Z^{\tau_1 \cdot \tau_2}}_{\substack{\text{new commitment to } \tau_1 \cdot \tau_2 \\ \text{(where } Z = e(G,G) \neq 1)}} \qquad \text{(bilinear pairing)}$$

Blind evaluation of QAP

Blind evaluation can be achieved with *black-box* pairings:

$$\begin{aligned} e(C(H(\tau)), C(T(\tau)) \cdot e(C(O(\tau)), C(1)) &= e(C(L(\tau)), C(R(\tau))) \\ e(H(\tau)G, T(\tau)G) \cdot e(O(\tau)G, G) &= e(L(\tau)G, R(\tau)G) \\ e(G, G)^{H(\tau)T(\tau)} \cdot e(G, G)^{O(\tau)} &= e(G, G)^{L(\tau)R(\tau)} \\ Z^{H(\tau)T(\tau)+O(\tau)} &= Z^{L(\tau)R(\tau)} \end{aligned}$$

Somewhat homomorphic commitment

Elliptic curves (DL):

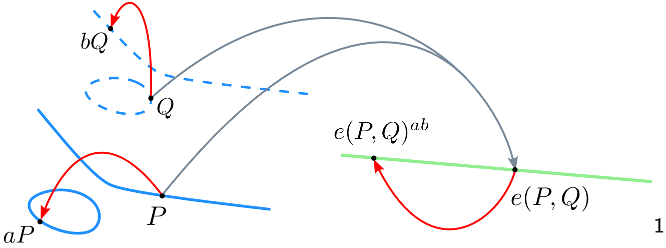
- $E \colon y^2 = x^3 + ax + b$ elliptic curve defined over $\mathbb{F}_q$, q a prime power.
- r prime divisor of $\#E(\mathbb{F}_q) = q + 1 - t$, t Frobenius trace.

A non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$

non-degenerate: $\forall P \in \mathbb{G}_1,\ P \neq \mathcal{O},\ \exists Q \in \mathbb{G}_2, e(P,Q) \neq 1_{\mathbb{G}_T}$

$\forall Q \in \mathbb{G}_2,\ Q \neq \mathcal{O},\ \exists P \in \mathbb{G}_1, e(P,Q) \neq 1_{\mathbb{G}_T}$

bilinear: $e([a]P,[b]Q) = e(P,[b]Q)^a = e([a]P,Q)^b = e(P,Q)^{ab}$



A pairing-based SNARK

Example: Groth16 [Gro16]

Given an instance $\Phi = (a_0, \ldots, a_\ell) \in \mathbb{F}_r^\ell$ of a public `NP` program F

A pairing-based SNARK

Example: Groth16 [Gro16]

Given an instance $\Phi = (a_0, \dots, a_\ell) \in \mathbb{F}_r^\ell$ of a public $\mathtt{NP}$ program F

- Setup: $(pk, vk) \leftarrow S(F, \tau, 1^\lambda)$ where

$$vk = (vk_{\alpha,\beta}, \{vk_{\pi_i}\}_{i=0}^{\ell}, vk_\gamma, vk_\delta) \in \mathbb{G}_T \times \mathbb{G}_1^{\ell+1} \times \mathbb{G}_2 \times \mathbb{G}_2$$

A pairing-based SNARK

Example: Groth16 [Gro16]

Given an instance $\Phi = (a_0, \ldots, a_\ell) \in \mathbb{F}_r^\ell$ of a public NP program F

- Setup: $(pk, vk) \leftarrow S(F, \tau, 1^\lambda)$ where

$$vk = (vk_{\alpha,\beta}, \{vk_{\pi_i}\}_{i=0}^{\ell}, vk_\gamma, vk_\delta) \in \mathbb{G}_T \times \mathbb{G}_1^{\ell+1} \times \mathbb{G}_2 \times \mathbb{G}_2$$

- Prove: $\pi \leftarrow P(\Phi, w, pk)$ where

$$\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \qquad (O_\lambda(1))$$

A pairing-based SNARK

Example: Groth16 [Gro16]

Given an instance $\Phi = (a_0, \ldots, a_\ell) \in \mathbb{F}_r^\ell$ of a public $\mathtt{NP}$ program F

- Setup: $(pk, vk) \leftarrow S(F, \tau, 1^\lambda)$ where

$$vk = (vk_{\alpha,\beta}, \{vk_{\pi_i}\}_{i=0}^{\ell}, vk_\gamma, vk_\delta) \in \mathbb{G}_T \times \mathbb{G}_1^{\ell+1} \times \mathbb{G}_2 \times \mathbb{G}_2$$

- Prove: $\pi \leftarrow P(\Phi, w, pk)$ where

$$\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \qquad (O_\lambda(1))$$

- Verify: $0/1 \leftarrow V(\Phi, \pi, vk)$ where V is

$$e(A, B) = vk_{\alpha,\beta} \cdot e(vk_x, vk_\gamma) \cdot e(C, vk_\delta) \qquad (O_\lambda(|\Phi|)) \tag{1}$$

  and $vk_x = \sum_{i=0}^{\ell} [a_i] vk_{\pi_i}$ depends only on the instance Φ and $vk_{\alpha,\beta} = e(vk_\alpha, vk_\beta)$ can be computed in the trusted setup for $(vk_\alpha, vk_\beta) \in \mathbb{G}_1 \times \mathbb{G}_2$.

Overview

1. Motivation
2. Blockchain
3. Zero-knowledge proofs
4. **Applications**
5. Research

Applications

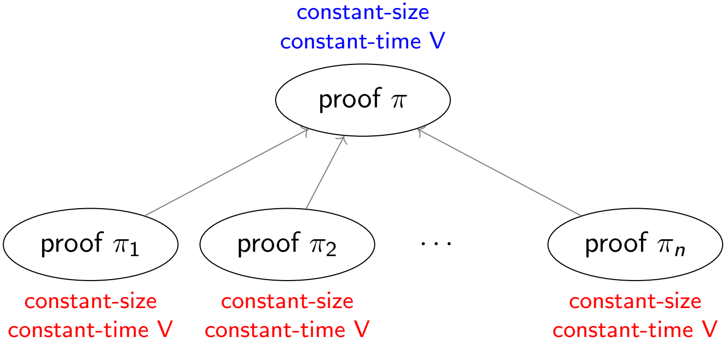
- Privacy: Monero, zcash, Aleo... or Tornado cash...
- Scalability: Mina... or Linea, Aztec...

Overview

1. Motivation
2. Blockchain
3. Zero-knowledge proofs
4. Applications
5. **Research**

Proof composition: why?

Aggregation:



constant-size
constant-time V

proof π

proof π_1

proof π_2

$\cdots$

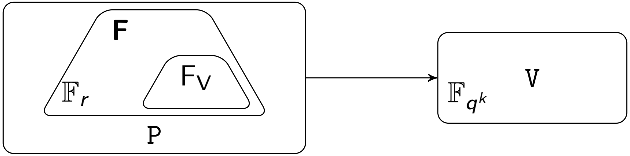
proof π_n

constant-size
constant-time V

constant-size
constant-time V

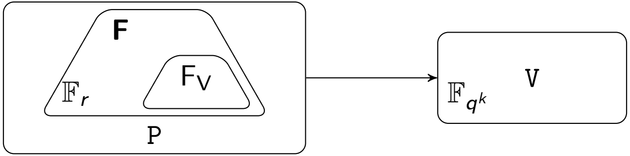
constant-size
constant-time V

Proof composition: how?



F	any program is expressed in $\mathbb{F}_r$
`P`	proving is performed over $\mathbb{G}_1$ (and $\mathbb{G}_2$) (of order r)
`V`	verification (eq. 1) is done in $\mathbb{F}^*_{q^k}$
F_V	program of `V` is natively expressed in $\mathbb{F}^*_{q^k}$ not $\mathbb{F}_r$

Proof composition: how?



F any program is expressed in $\mathbb{F}_r$

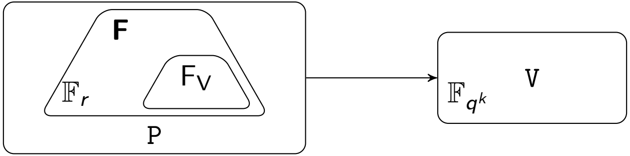
`P` proving is performed over $\mathbb{G}_1$ (and $\mathbb{G}_2$) (of order r)

`V` verification (eq. 1) is done in $\mathbb{F}^*_{q^k}$

F_V program of `V` is natively expressed in $\mathbb{F}^*_{q^k}$ not $\mathbb{F}_r$

- 1st attempt: choose a curve for which $q = r$ (impossible)

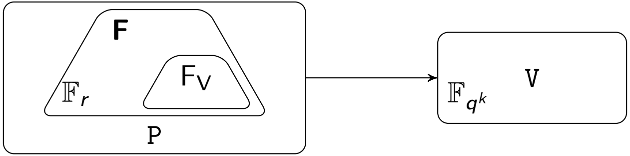
Proof composition: how?



F	any program is expressed in $\mathbb{F}_r$
P	proving is performed over $\mathbb{G}_1$ (and $\mathbb{G}_2$) (of order r)
V	verification (eq. 1) is done in $\mathbb{F}^*_{q^k}$
F_V	program of V is natively expressed in $\mathbb{F}^*_{q^k}$ not $\mathbb{F}_r$

- 1st attempt: choose a curve for which $q = r$ (impossible)
- 2nd attempt: simulate $\mathbb{F}_q$ operations via $\mathbb{F}_r$ operations ($\times \log q$ blowup)

Proof composition: how?



- **F** any program is expressed in $\mathbb{F}_r$
- `P` proving is performed over $\mathbb{G}_1$ (and $\mathbb{G}_2$) (of order r)
- `V` verification (eq. 1) is done in $\mathbb{F}^*_{q^k}$
- F_V program of `V` is natively expressed in $\mathbb{F}^*_{q^k}$ not $\mathbb{F}_r$

- 1st attempt: choose a curve for which $q = r$ (impossible)
- 2nd attempt: simulate $\mathbb{F}_q$ operations via $\mathbb{F}_r$ operations ($\times \log q$ blowup)
- 3rd attempt: use a cycle/chain of pairing-friendly elliptic curves [CFH+15, BCTV14a, BCG+20, EG20, EG22, AEG23]

2-cycles and 2-chains

A 2-cycle of elliptic curves:

$$\begin{tikzcd}
\boxed{E_1(\mathbb{F}_q)} \arrow[d, bend left, "\#E_0(\mathbb{F}_p) = q"] \\
\boxed{E_0(\mathbb{F}_p)} \arrow[u, bend left, "\#E_1(\mathbb{F}_q) = p"]
\end{tikzcd}$$

A 2-chain of elliptic curves:

$$\begin{tikzcd}
\boxed{E_1(\mathbb{F}_q)} \\
\boxed{E_0(\mathbb{F}_p)} \arrow[u, "\#E_1(\mathbb{F}_q) = h \cdot p"']
\end{tikzcd}$$

Some contributions

- Blockchain limitations: confidentiality and scalability
- pairing-based zk-SNARKs are a solution (constant-size proof and fast verification)

Some contributions

- Blockchain limitations: confidentiality and scalability
- pairing-based zk-SNARKs are a solution (constant-size proof and fast verification)
- What are SNARK-friendly curves? Fast arithmetic? **[DCC 2022, AfricaCrypt 2022]**

Some contributions

- Blockchain limitations: confidentiality and scalability
- pairing-based zk-SNARKs are a solution (constant-size proof and fast verification)
- What are SNARK-friendly curves? Fast arithmetic? **[DCC 2022, AfricaCrypt 2022]**
- Proof composition for better confidentiality and scalability → 2-chains and 2-cycles **[CANS 2020, EuroCrypt 2022, DCC 2022, JoC 2024]**

Some contributions

- Blockchain limitations: confidentiality and scalability
- pairing-based zk-SNARKs are a solution (constant-size proof and fast verification)
- What are SNARK-friendly curves? Fast arithmetic? **[DCC 2022, AfricaCrypt 2022]**
- Proof composition for better confidentiality and scalability → 2-chains and 2-cycles **[CANS 2020, EuroCrypt 2022, DCC 2022, JoC 2024]**
- Pairings in R1CS for fast generation of the composed proof **[ACNS 2023]**

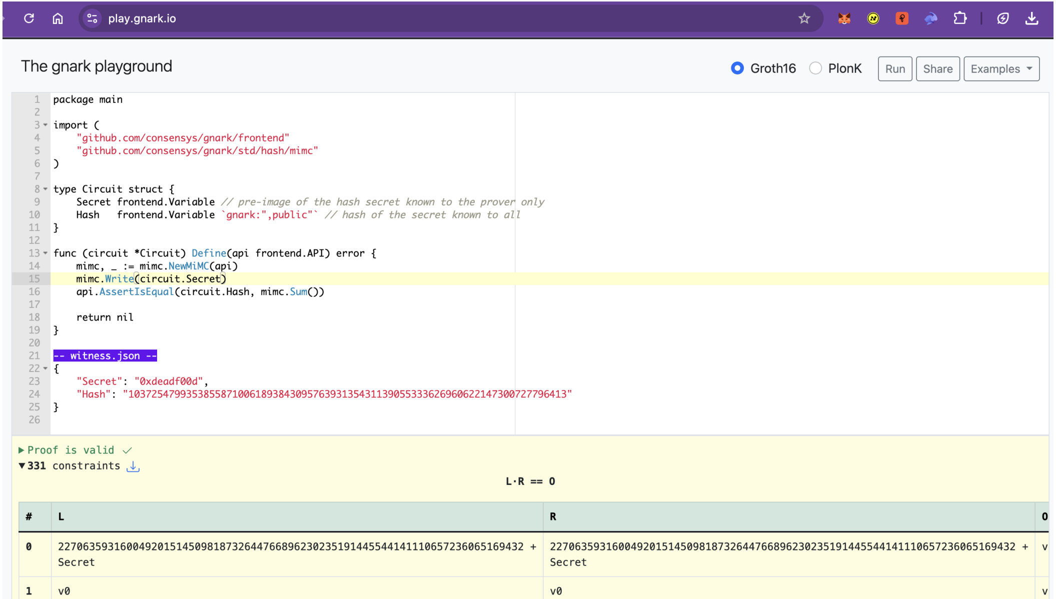
Some contributions

- Blockchain limitations: confidentiality and scalability
- pairing-based zk-SNARKs are a solution (constant-size proof and fast verification)
- What are SNARK-friendly curves? Fast arithmetic? **[DCC 2022, AfricaCrypt 2022]**
- Proof composition for better confidentiality and scalability → 2-chains and 2-cycles **[CANS 2020, EuroCrypt 2022, DCC 2022, JoC 2024]**
- Pairings in R1CS for fast generation of the composed proof **[ACNS 2023]**
- Multi-scalar multiplication for fast generation of proofs **[TCHES 2023, ZPRIZE winner]**

Some contributions

- Blockchain limitations: confidentiality and scalability
- pairing-based zk-SNARKs are a solution (constant-size proof and fast verification)
- What are SNARK-friendly curves? Fast arithmetic? **[DCC 2022, AfricaCrypt 2022]**
- Proof composition for better confidentiality and scalability → 2-chains and 2-cycles **[CANS 2020, EuroCrypt 2022, DCC 2022, JoC 2024]**
- Pairings in R1CS for fast generation of the composed proof **[ACNS 2023]**
- Multi-scalar multiplication for fast generation of proofs **[TCHES 2023, ZPRIZE winner]**
- Implementations: **gnark, linea, arkworks, sonobe, ...**

gnark playground



```
package main

import (
	"github.com/consensys/gnark/frontend"
	"github.com/consensys/gnark/std/hash/mimc"
)

type Circuit struct {
	Secret frontend.Variable // pre-image of the hash secret known to the prover only
	Hash   frontend.Variable `gnark:",public"` // hash of the secret known to all
}

func (circuit *Circuit) Define(api frontend.API) error {
	mimc, _ := mimc.NewMiMC(api)
	mimc.Write(circuit.Secret)
	api.AssertIsEqual(circuit.Hash, mimc.Sum())

	return nil
}

-- witness.json --
{
	"Secret": "0xdeadf00d",
	"Hash": "1037254799353855871006189384309576393135431139055333626960622147300727796413"
}
```

```
▶ Proof is valid ✓
▼ 331 constraints
L·R == 0
```

#	L	R	O
0	227063593160049201514509818732644766896230235191445544141110657236065169432 + Secret	227063593160049201514509818732644766896230235191445544141110657236065169432 + Secret	v
1	v0	v0	v

Thank you

- website: `https://yelhousni.eth.limo`
- email: `youssef.elhousni@consensys.net`
- telegram: `@ElMarroqui`
- x: `@YoussefElHoun3`
- github: `@yelhousni`

References I

Diego F. Aranha, Youssef El Housni, and Aurore Guillevic.
A survey of elliptic curves for proof systems.
DCC, 91(11):3333–3378, 2023.

Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer.
From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again.
In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu.
ZEXE: Enabling decentralized private computation.
In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press, May 2020.

References II

Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza.
Scalable zero knowledge via cycles of elliptic curves.
In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Berlin, Heidelberg, August 2014.

Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza.
Succinct non-interactive zero knowledge for a von neumann architecture.
In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

Manuel Blum, Paul Feldman, and Silvio Micali.
Non-interactive zero-knowledge and its applications (extended abstract).
In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

References III

Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur.
Geppetto: Versatile verifiable computation.
In *2015 IEEE Symposium on Security and Privacy*, pages 253–270. IEEE Computer Society Press, May 2015.

Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward.
Marlin: Preprocessing zkSNARKs with universal and updatable SRS.
In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020.

References IV

Youssef El Housni and Aurore Guillevic.
Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition.
In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 259–279. Springer, Cham, December 2020.

Youssef El Housni and Aurore Guillevic.
Families of SNARK-friendly 2-chains of elliptic curves.
In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 367–396. Springer, Cham, May / June 2022.

Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova.
Quadratic span programs and succinct NIZKs without PCPs.
In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013.

References V

Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers.
Updatable and universal common reference strings with applications to zk-SNARKs.
In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Cham, August 2018.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff.
The knowledge complexity of interactive proof-systems (extended abstract).
In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

Jens Groth.
Pairing-based non-interactive zero-knowledge proofs (invited talk).
In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *PAIRING 2010*, volume 6487 of *LNCS*, page 206. Springer, Berlin, Heidelberg, December 2010.

References VI

Jens Groth.
On the size of pairing-based non-interactive arguments.
In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru.
PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge.
Cryptology ePrint Archive, Report 2019/953, 2019.

Joe Kilian.
A note on efficient zero-knowledge proofs and arguments (extended abstract).
In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

References VII

Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn.
Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings.
In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

Silvio Micali.
CS proofs (extended abstracts).
In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.