Zero-Knowledge Proofs

A Practical Introduction

Dmytro Zakharov

R&D Cryptography Engineer @ Distributed Lab

Privacy-Preserving ML Lead @ Rarimo

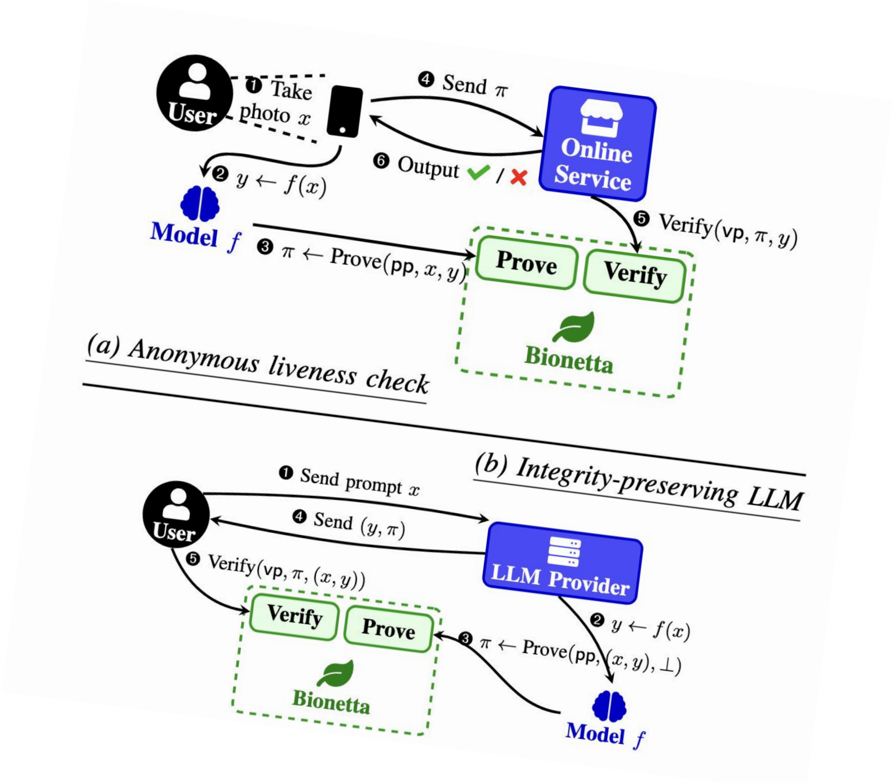
MSc Student in Mathematics @ KSE

`zamdimon.github.io`

ZKP is awesome!

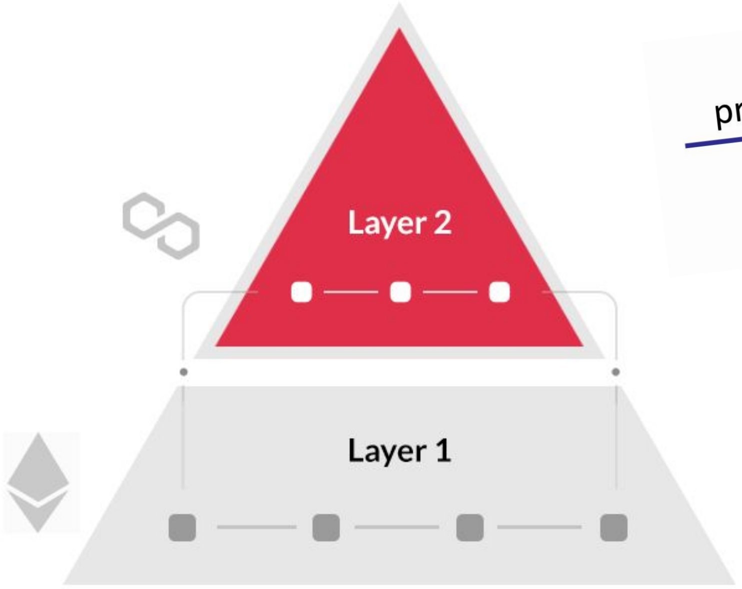
Reason 1. Many applications!

Voting for the next President anonymously!

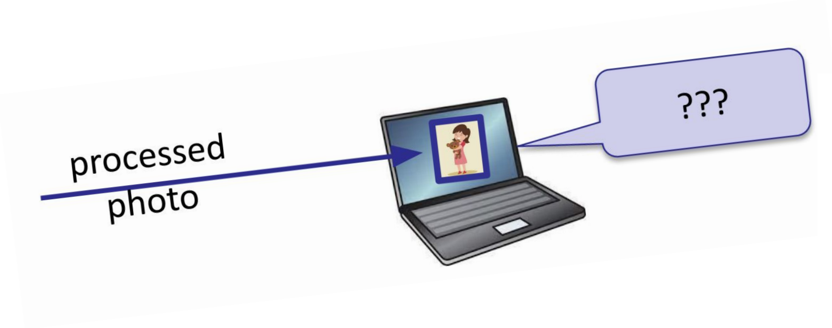


Ensuring LLM integrity

ZKP is awesome!



Scaling Blockchain infrastructure



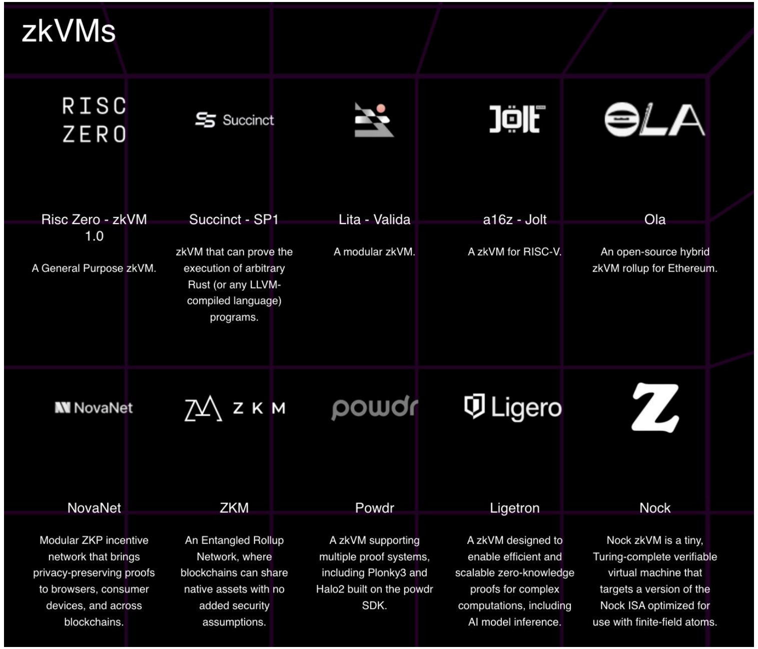
Verifying photo processing

Note

We will consider how this is mathematically formulated!

ZKP is awesome!

Reason 2. Immense commercial interest!



See https://zkhack.dev/the-map-of-zk/

ZKP is awesome!

Reason 3. This is where advanced mathematics is applied!

$$l(X) = (\mathbf{a}_L - z \cdot \mathbf{1}^{n \cdot m}) + \mathbf{s}_L \cdot X \in \mathbb{Z}_p^{n \cdot m}[X] \quad (70)$$

$$r(X) = \mathbf{y}^{n \cdot m} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{n \cdot m} + \mathbf{s}_R \cdot X) + \sum_{j=1}^{m} z^{1+j} \cdot \left(\mathbf{0}^{(j-1)\cdot n} \parallel \mathbf{2}^n \parallel \mathbf{0}^{(m-j)\cdot n}\right) \in \mathbb{Z}_p^{n \cdot m} \quad (71)$$

In the computation of τ_x, we need to adjust for the randomness of each commitment V_j, so that $\tau_x = \tau_1 \cdot x + \tau_2 \cdot x^2 + \sum_{j=1}^{m} z^{1+j} \cdot \gamma_j$. Further, $\delta(y,z)$ is updated to incorporate more cross terms.

$$\delta(y,z) = (z - z^2) \cdot \langle \mathbf{1}^{n \cdot m}, \mathbf{y}^{n \cdot m} \rangle - \sum_{j=1}^{m} z^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{2}^n \rangle$$

The verification check (65) needs to be updated to include all the V_j commitments. (7

$$g^{\hat{t}} h^{\tau_x} \stackrel{?}{=} g^{\delta(y,z)} \cdot \mathbf{V}^{z^2 \cdot \mathbf{z}^m} \cdot T_1^x \cdot T_2^{x^2}$$

Finally, we change the definition of P (66) such that it is a commitment to the new $\mathbf{r}$.

$$P = AS^x \cdot \mathbf{g}^{-z} \cdot \mathbf{h}'^{z \cdot \mathbf{y}^{n \cdot m}} \prod_{j=1}^{m} \mathbf{h}'^{z^{j+1} \cdot \mathbf{2}^n}_{[(j-1)\cdot n : j \cdot n - 1]}$$

$$\begin{aligned}
\langle \tau_{\mathbf{D}}, \mathbf{t}^{(z)} \rangle &= \langle \tau_{\mathbf{D}}, \mathsf{tensor}(\mathbf{c}^{(z)}) \otimes \mathbf{s}' \otimes (1, d', \ldots, d'^{\ell-1}) \otimes (1, X, \ldots, X^{d-1}) \rangle \\
&= \sum_{i \in [\kappa], j \in [dk], o \in [d], p \in [\ell]} T_{i,j,o,p} \cdot \mathsf{tensor}(\mathbf{c}^{(z)})_i \cdot \mathbf{s}'_j \cdot d'^o \cdot X^p \\
&= \langle \mathsf{tensor}(\mathbf{c}^{(z)}), \sum_{j \in [dk]} \left(\sum_{o \in [d], p \in [\ell]} T_{*,j,o,p} \cdot d'^o \cdot X^p \right) \cdot \mathbf{s}'_j \rangle \\
&= \langle \mathsf{tensor}(\mathbf{c}^{(z)}), \sum \mathsf{pow}(\tau_{\mathbf{D}})_{*,j} \cdot \mathbf{s}'_j \rangle = \langle \mathsf{tensor}(\mathbf{c}^{(z)}), \mathsf{pow}(\tau_{\mathbf{D}}) \mathbf{s}' \rangle .
\end{aligned}$$

random public input

5. Honest-Verifier Zero-Knowledge: Π is zero-knowledge if there exists a PPT simulator every PPT adversary $\mathcal{A}$:

$$\left| \Pr\left[\begin{array}{c} \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) = 1 \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} \end{array} : \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array} \right] - \right.$$

$$\left. \geq \Pr\left[\begin{array}{c} \langle \mathcal{S}(\sigma, \mathsf{pp}, \mathbb{x}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle = 1 \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} \end{array} : \begin{array}{c} (\mathsf{gp}, \sigma) \leftarrow \mathcal{S}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbb{i}) \end{array} \right] \right| \leq \mathsf{negl}(\lambda)$$

Lemma 4.9. *For every function $f \colon \mathcal{L} \to \mathbb{F}$, degree parameter $d \in \mathbb{N}$, folding parameter $k \in \mathbb{N}$, and distance parameter $\delta \in (0, \min\{\Delta(f, \mathsf{RS}[\mathbb{F}, \mathcal{L}, d]), 1 - \mathsf{B}^\star(\rho)\})$, letting $\rho := d/|\mathcal{L}|$,*

$$\Pr_{r^{\mathsf{fold}} \leftarrow \mathbb{F}} \left[\Delta(\mathsf{Fold}(f, k, r^{\mathsf{fold}}), \mathsf{RS}[\mathbb{F}, \mathcal{L}^k, d/k]) \leq \delta \right] \leq \mathsf{err}^\star(d/k, \rho, \delta, k) .$$

Above, $\mathsf{B}^\star$ and $\mathsf{err}^\star$ are the proximity bound and error (respectively) described in Section 4.1.

Proof. Suppose towards contradiction that

$$\Pr_{r^{\mathsf{fold}} \leftarrow \mathbb{F}} \left[\Delta(\mathsf{Fold}(f, k, r^{\mathsf{fold}}), \mathsf{RS}[\mathbb{F}, \mathcal{L}^k, d/k]) \leq \delta \right] > \mathsf{err}^\star(d/k, \rho, \delta, k) .$$

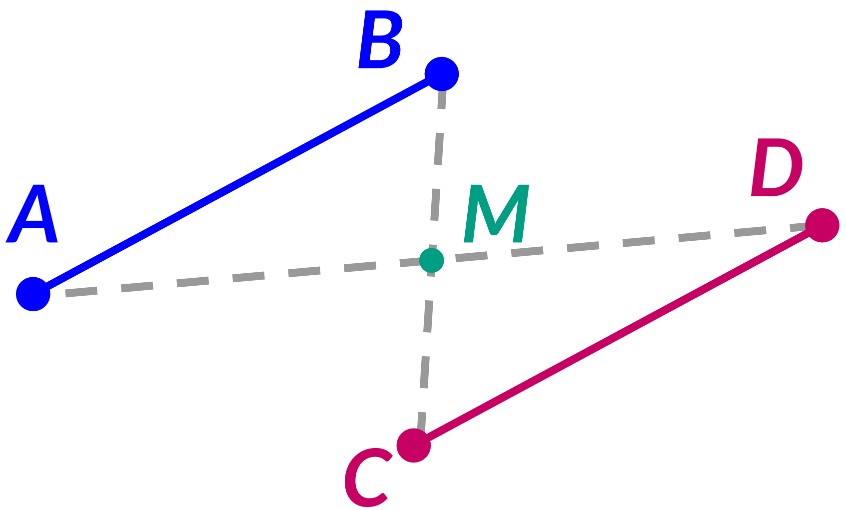
Letting $\hat{p}_x$ be defined from f as in Definition 4.8, define $c_0, \ldots, c_{k-1}$ where $c_j \colon \mathcal{L}^k \to \mathbb{F}$ is the function where $c_j(x)$ is the j-th coefficient of $\hat{p}_x$ (i.e., so that $\hat{p}_x(X) \equiv \sum_{j=0}^{k-1} c_j(x) \cdot X^j$ for every $x \in \mathcal{L}^k$). Observe that

$$\mathsf{Fold}(f, k, \alpha)(x) = \hat{p}_x(\alpha) = \sum_{j=0}^{k-1} c_j(x) \cdot \alpha^j .$$

Introduction

Classical Proofs (in high school geometry)

Problem: Suppose *M* is the midpoint of *AD* and *BC*. Prove that *AB* and *CD* are parallel.



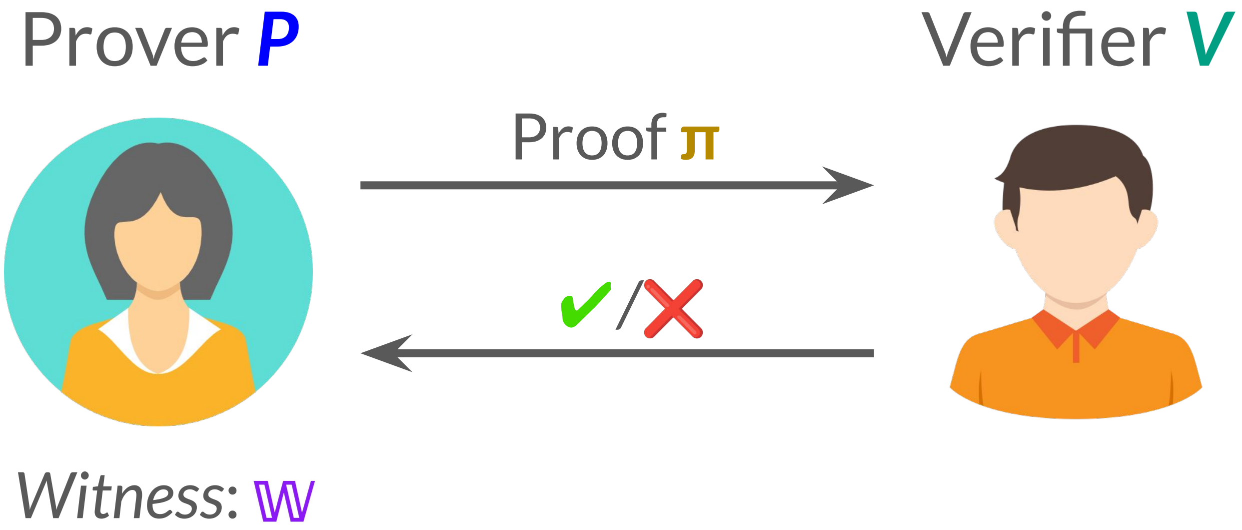
- **Prover**: you on the test.
- **Verifier**: your teacher.
- **Public Statement**: theorem.
- **Proof (and witness)**: sequence of axioms and logical facts that proves the given theorem.

Question

Why and **how** is this concept generalized to crypto systems?

General Setup

Claim: $\mathbb{X}$



Example: Product of two primes

Claim: $N = pq$

Prover P

Verifier V

Proof $\pi = (p, q)$ →

← ✔ iff $N = pq$

Witness: (p, q)

Some Definitions

Definition

Relation $\mathcal{R}$ is **effective** if $(x, w) \in \mathcal{R}$ can be verified in polynomial time. More specifically, in poly($|x|$).
The **language** of $\mathcal{R}$ is defined as $\mathcal{L}_{\mathcal{R}} = \{x\colon \exists\, w \text{ s.t. } (x, w) \in \mathcal{R}\}$

Examples

- Suppose $(N, (p, q)) \in \mathcal{R}$ iff $N = pq$. This is an effective relation since computing pq is polynomially fast.
- Fix hash function $\mathcal{H}$. Suppose $(d, m) \in \mathcal{R}$ iff $d = \mathcal{H}(m)$. This is trivially an effective relation.

Some Definitions

$$\exists \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$$

Effective Relation. Encodes a logic of the statement to be proven.

Public statement. Public part of the statement (e.g., public key / output of function).

Witness. Secret data which is not computable from the public statement.

Example

Take **SHA256** preimage relation. Given, say,

x=`0x163004120d6e29aacc023568b6d8ca5f9dd3e09beeeb1e359fcf671de5466bf3`

you cannot determine ***w*** such that **SHA256**(***w***) = ***x***.

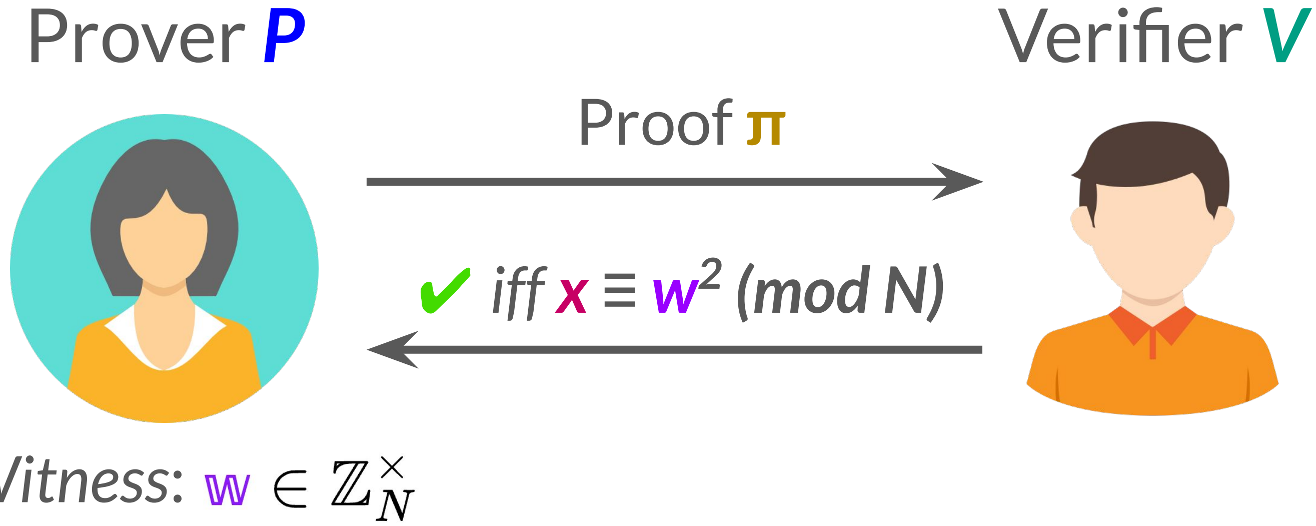
Turns out that in this particular case, ***w*** = *"KSE"*!

This way, proving "I know hash preimage of ***x***" totally makes sense!

ZKP for Quadratic Residues

Relation: $(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \subseteq (\mathbb{Z}_N^\times)^2 \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^2 \pmod N$

Language: $\mathcal{L}_\mathcal{R} = \{\mathbb{x} \in \mathbb{Z}_N^\times : \exists \mathbb{w} \in \mathbb{Z}_N^\times \text{ s.t. } \mathbb{x} \equiv \mathbb{w}^2 \pmod N\}$



ZKP for Quadratic Residues

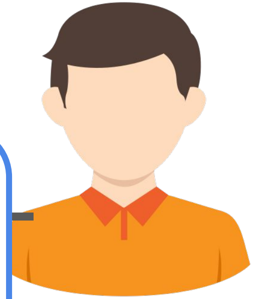
Relation: $(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \subseteq (\mathbb{Z}_N^\times)^2 \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}$

Language: $\mathcal{L}_\mathcal{R} = \{\mathbb{x} \in \mathbb{Z}_N^\times : \exists \mathbb{w} \in \mathbb{Z}_N^\times \text{ s.t. } \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}\}$

Prover ***P***

Proof **π**

Verifier ***V***



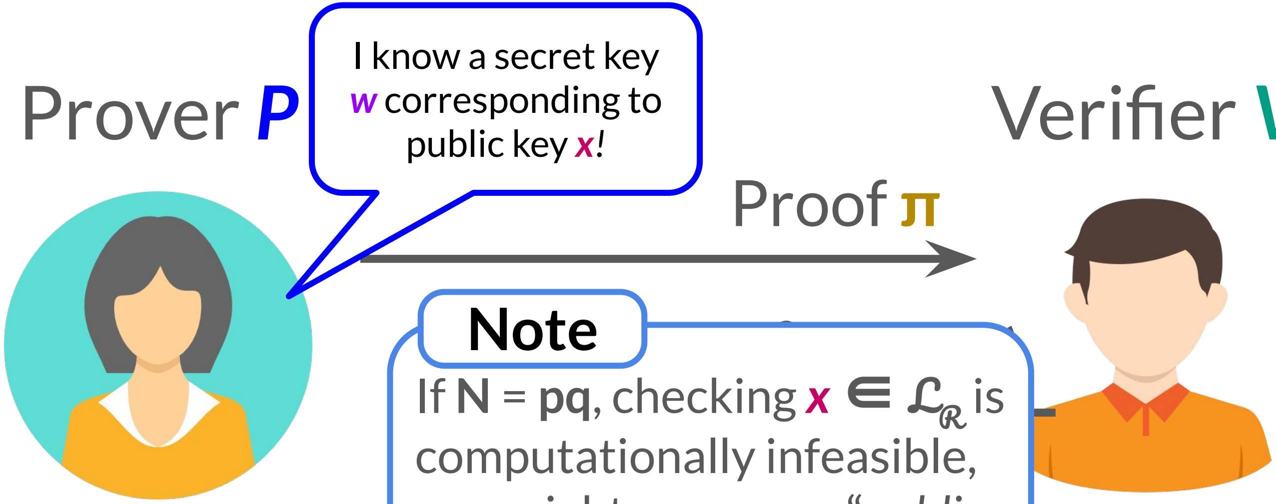
Note

If **N** = **pq**, checking ***x*** ∈ $\mathcal{L}_\mathcal{R}$ is computationally infeasible, so ***x*** might serve as a "*public key*", while ***w*** as a "*secret key*".

ZKP for Quadratic Residues

Relation: $(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \subseteq (\mathbb{Z}_N^\times)^2 \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}$

Language: $\mathcal{L}_\mathcal{R} = \{\mathbb{x} \in \mathbb{Z}_N^\times : \exists \mathbb{w} \in \mathbb{Z}_N^\times \text{ s.t. } \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}\}$



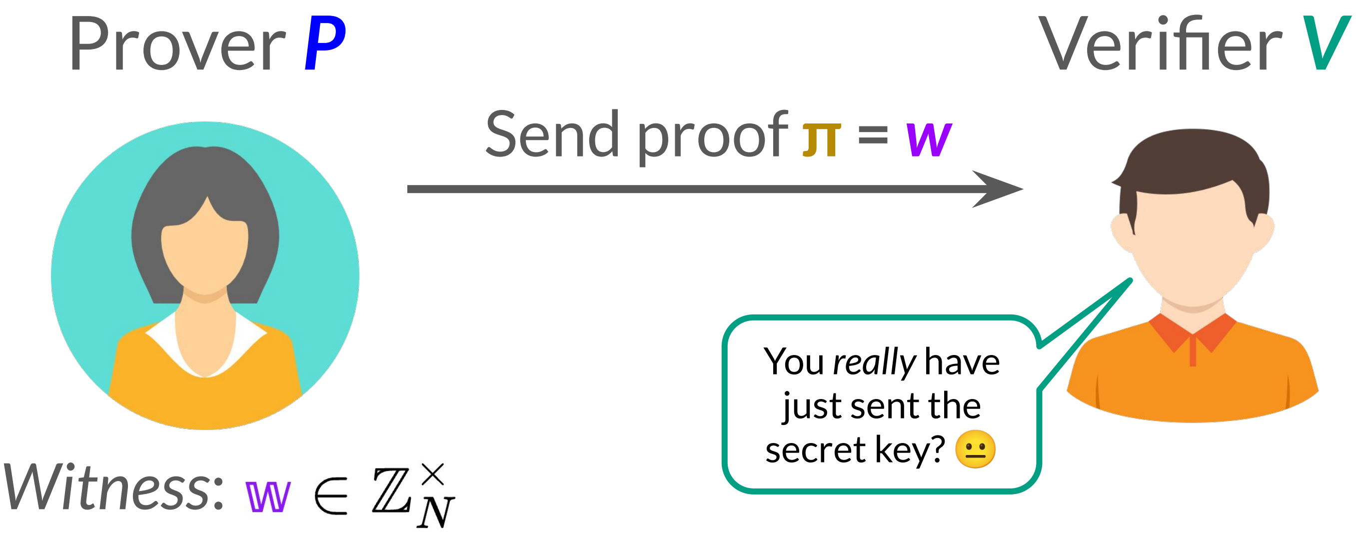
Note

If **N** = **pq**, checking ***x*** ∈ $\mathcal{L}_\mathcal{R}$ is computationally infeasible, so ***x*** might serve as a "*public key*", while ***w*** as a "*secret key*".

ZKP for Quadratic Residues

Relation: $(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \subseteq (\mathbb{Z}_N^\times)^2 \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}$

Language: $\mathcal{L}_\mathcal{R} = \{\mathbb{x} \in \mathbb{Z}_N^\times : \exists \mathbb{w} \in \mathbb{Z}_N^\times \text{ s.t. } \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}\}$



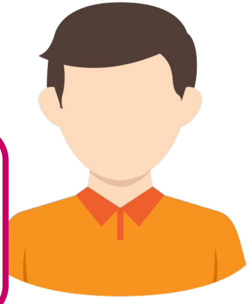
ZKP for Quadratic Residues

Relation: $(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \subseteq (\mathbb{Z}_N^\times)^2 \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}$

Language: $\mathcal{L}_\mathcal{R} = \{\mathbb{x} \in \mathbb{Z}_N^\times : \exists \mathbb{w} \in \mathbb{Z}_N^\times \text{ s.t. } \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}\}$

Prover ***P*** | Verifier ***V***

Send proof **π** = ***w***



Question

Is there any better way to build this protocol?

Witness: $\mathbb{w} \in \mathbb{Z}_N^\times$

Idea: reveal *just enough* information to convince ***V***!

Interactive Protocols

Goldwasser, Micali, and Rackoff: inventors of ZK (~1985)

Interactive Protocol

Claim: $\mathbb{x}$

Prover ***P***

Witness: $\mathbb{w}$

Send commitment c_1 →

← Send query q_1

Send commitment c_2 →

← Send query q_2

...

Verifier ***V***

$\mathcal{V}(\mathbb{x}, c_1, q_1, \ldots, c_n, q_n)$

Notation: $\text{view}_{\mathcal{V}}(\mathcal{P}, \mathcal{V})[\mathbb{x}] = (\mathbb{x}, c_1, q_1, \ldots, c_n, q_n)$

Interactive Protocol: Formal Definition

Notation: $\langle P, V\rangle(x)$ – *interaction between* ***P*** *and* ***V*** *on statement* ***x***

Definition

A pair of algorithms (P, V) is an **interactive proof** (**IP**) with security parameter λ for language $\mathcal{L}_{\mathcal{R}}$ if V runs in polynomial time & the following two properties holds:

- **Completeness**: For any $x \in \mathcal{L}_{\mathcal{R}}$,

$$\Pr[\langle P, V\rangle(x) = accept] = 1.$$

- **Soundness**: For any $x \notin \mathcal{L}_{\mathcal{R}}$, and any P^*,

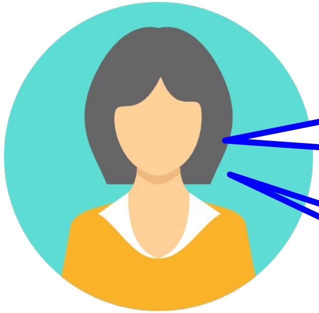
$$\Pr[\langle P^*, V\rangle(x) = accept] \leq \text{negl}(\lambda).$$

in practice, means very small: less than 2^{-100}

IP for Quadratic Residues

Claim: $\exists \mathbb{w} : \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}$

Prover ***P***



Witness: $\mathbb{w} \in \mathbb{Z}_N^\times$

$r \leftarrow_\$ \mathbb{Z}_N, c \leftarrow r^2 \bmod N$ *

💡 I can send you both square roots of ***c*** and ***cx***, but then you will know ***w***!

So you choose what I send!

Verifier ***V***

* $\leftarrow_\$$ = sample uniformly

IP for Quadratic Residues

Claim: $\exists \mathtt{w} : \mathtt{x} \equiv \mathtt{w}^2 \pmod{N}$

Prover ***P***

$r \leftarrow_\$ \mathbb{Z}_N, c \leftarrow r^2 \bmod N$ →

← $b \leftarrow_\$ \{0, 1\}$

$z \leftarrow r\mathtt{w}^b$ →

Verifier ***V***

Witness: $\mathtt{w} \in \mathbb{Z}_N^\times$

Repeat **λ** times

$z^2 = c\mathtt{x}^b$?

Proposition. This protocol is **complete** and **sound**.

Proof of Completeness and Soundness

Proof.

Completeness. If the prover ***P*** is honest, we have:

$$z^2 = (r\mathbb{w}^b)^2 = r^2(\mathbb{w}^2)^b = c\mathbb{x}^b$$

Soundness. If the prover ***P**** is dishonest with $\boldsymbol{x} \notin \mathcal{L}_{\mathcal{R}}$, there are two possible cases:

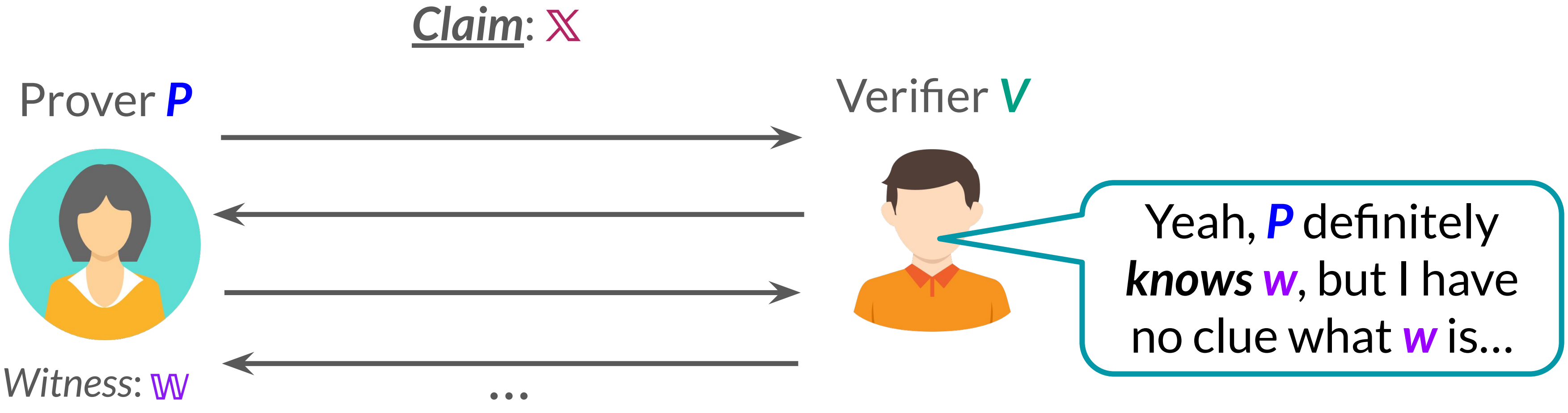
1. $\boldsymbol{c} \notin \mathcal{L}_{\mathcal{R}}$. Then if ***V*** outputs $\boldsymbol{b}$ = *0*, ***P**** cannot produce the valid corresponding $\boldsymbol{z}$, thus he loses.
2. $\boldsymbol{c} \in \mathcal{L}_{\mathcal{R}}$. Then if ***V*** outputs $\boldsymbol{b}$ = *1*, ***P**** similarly loses.

Thus, ***P**** cheats with probability **at most 1/2**.

Zero-Knowledge (Finally!)

$$\Pr[\langle P^*, V\rangle(x) = \textit{accept} \text{ after } \lambda \text{ rounds}] \leq \boxed{?}$$

What is also nice about this protocol is that it is additionally ***zero-knowledge*** and ***argument of knowledge***!



Zero-Knowledge (Finally!)

So what is ***zero-knowledge***?

Informally: **view**$_V$(P, V)[x] does not reveal any information about the underlying witness w. Formally:

Definition

An interactive protocol (P, V) is (***honest-verifier***) **zero-knowledge** if there exists a poly-time simulator **Sim** such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$:

$$\textbf{view}_V(P, V)[x] \approx \textbf{Sim}(x, 1^\lambda)$$

computational indistinguishability

Argument of Knowledge

So what is **argument of knowledge**?

Idea: proving that $x \in \mathcal{L}_{\mathcal{R}}$ is not enough! P must **know** w!

Example

Let $\mathbf{G}$ be a cyclic group of prime order q generated by some element $g \in \mathbf{G}$. Define the relation $\mathcal{R}$ over $\mathbf{G} \times \mathbf{Z}_q$ as follows: $\mathcal{R}(h, \alpha) = 1$ iff $h = g^{\alpha}$. Then, $\mathcal{L}_{\mathcal{R}} = \mathbf{G}$! So proving that $h \in \mathcal{L}_{\mathcal{R}}$ is pointless.

(*almost*) **Formally**: exists extractor E^P that, given P as an oracle, for any $x \in \mathcal{L}_{\mathcal{R}}$, outputs witness w (s.t. $\mathcal{R}(x,w)=1$) in poly-time.

Summary

Proposition. IP for Quadratic Residues is **zero-knowledge** and **argument of knowledge**.
We omit the proof (and will come back if we have time).

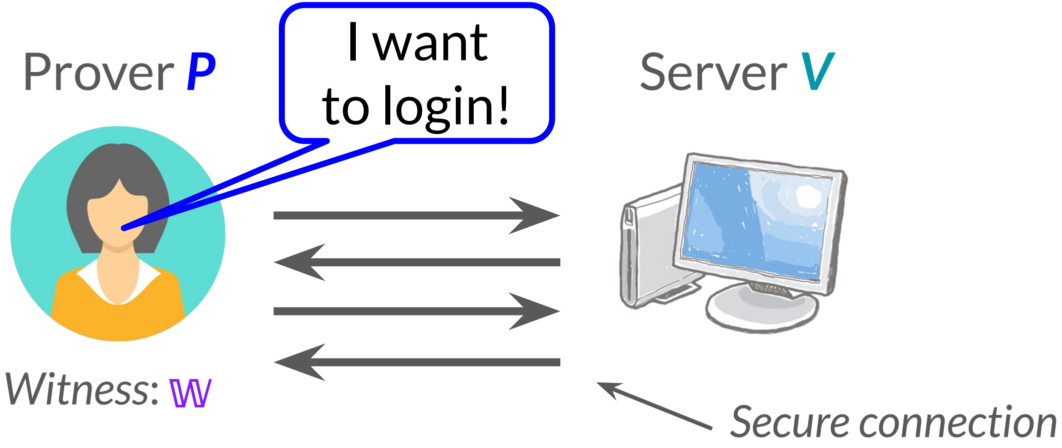
Summary: Specified IP Π_{QR} has the following properties:

- Π_{QR} is **complete**: valid statement is always accepted;
- Π_{QR} is **sound**: invalid statement is impossible to prove;
- Π_{QR} is **zero-knowledge**: V does not get anything about w;
- Π_{QR} is **argument of knowledge**: P knows square root of x.

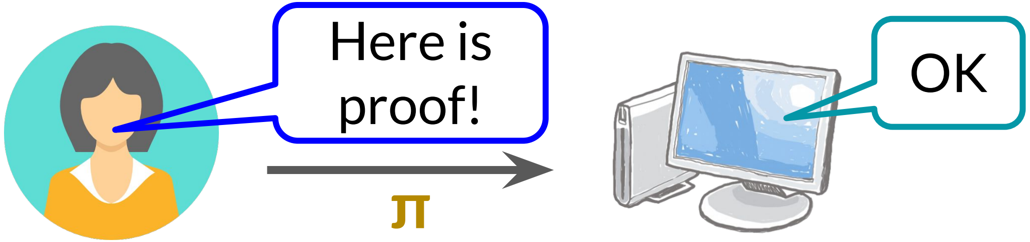
Non-Interactiveness

Suppose we want to deploy authentication based on Π_{QR}...

With ***interactive*** protocol we have:



Instead, we want the following:



Motivation

Π_{QR} is **public-coin** (meaning, V only sends random challenges). It seems like an overkill to require connection just to receive challenges!

💡 **Idea.** Let P generate the whole transcript on its own:

π := $\mathbf{view}_{V^*}(P, V^*)[x]$ (where challenges of V^* are sampled by P)

😡 **Problem.** How can V be sure that P generated all coins fairly?

Fiat-Shamir Transformation

Theorem

(*almost*) Any constant-round public-coin IP can be made non-interactive argument of knowledge (**NARK**).

💡 Idea (Fiat-Shamir)

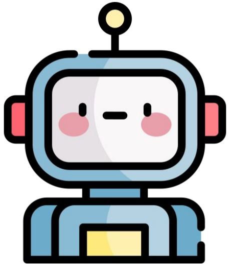
Suppose current transcript (view) is T. Set next randomness r as

$$r = H(T)$$

for *random oracle* ***H***.

Prover ***P***

"Verifier" ***V***



c_1 →

← $r_1 = H(x, c_1)$

c_2 →

← $r_2 = H(x, c_1, r_1, c_2)$

…

🎉 Congratulations!

By applying Fiat-Shamir transformation to **Π_{QR}***(with certain subtleties)*, we have constructed the **first zk-NARK** for

$$(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^2 \pmod{N}$$

Using very similar idea, we can construct NARKs for:

Knowledge of root: $(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \subseteq (\mathbb{Z}_N^\times)^2 \Leftrightarrow \mathbb{x} \equiv \mathbb{w}^r \pmod{N}$

Schnorr IP: $(h, \alpha) \in \mathcal{R} = \mathbb{Z}_q \times \mathbb{G} \Leftrightarrow h = g^\alpha$

Problems (again)

But what about...

$$(\mathbb{x}, \mathbb{w}) \in \mathcal{R} \Leftrightarrow \mathbb{x} = \mathcal{H}(\mathbb{w}),\ \mathcal{H} \text{ is a hash function}$$

Turns out that we can effectively prove this relation by:

1. Implementing ***H*** as an *arithmetic circuit*.
2. Building zk-**S**NARK over arithmetic circuits.

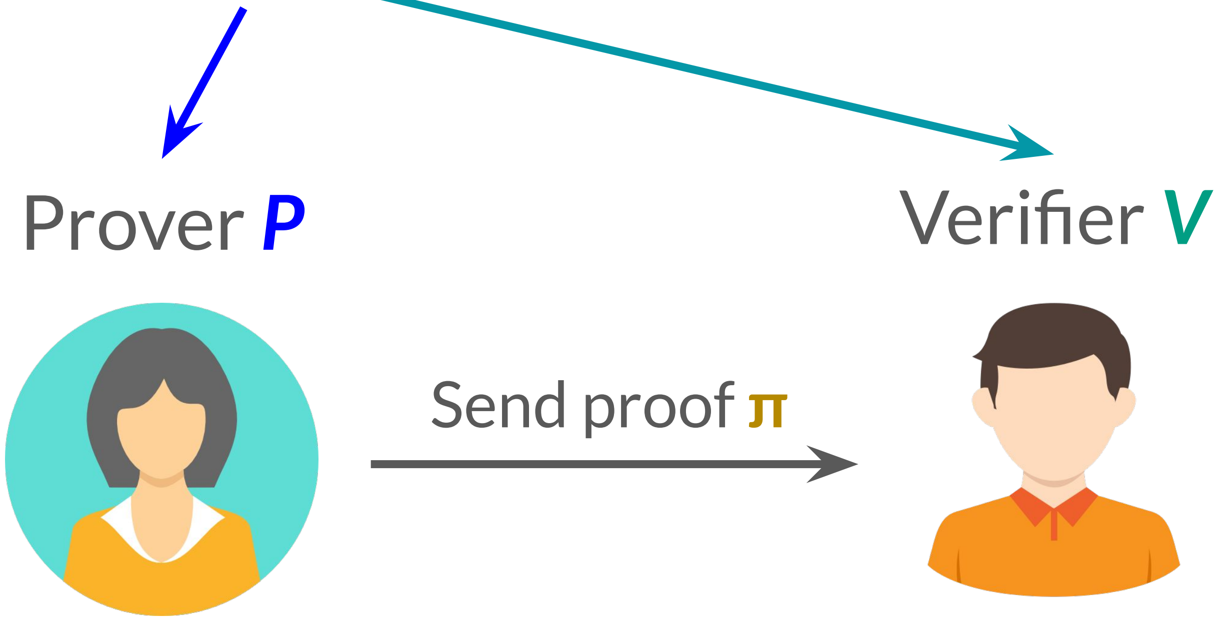
Note

This is well beyond the scope of this talk, but we will give a superficial overview nonetheless!

zk-SNARK

Preprocessing NARK

$$(\sigma_p, \sigma_v) \leftarrow \texttt{Setup}(1^\lambda, \mathcal{R})$$



$\pi \leftarrow \texttt{Prove}(\sigma_p, \mathbb{x}, \mathbb{w})$

$\texttt{Verify}(\sigma_v, \pi, \mathbb{x})?$

SNARK

Succinct **N**on-Interactive **Ar**gument of **K**nowledge

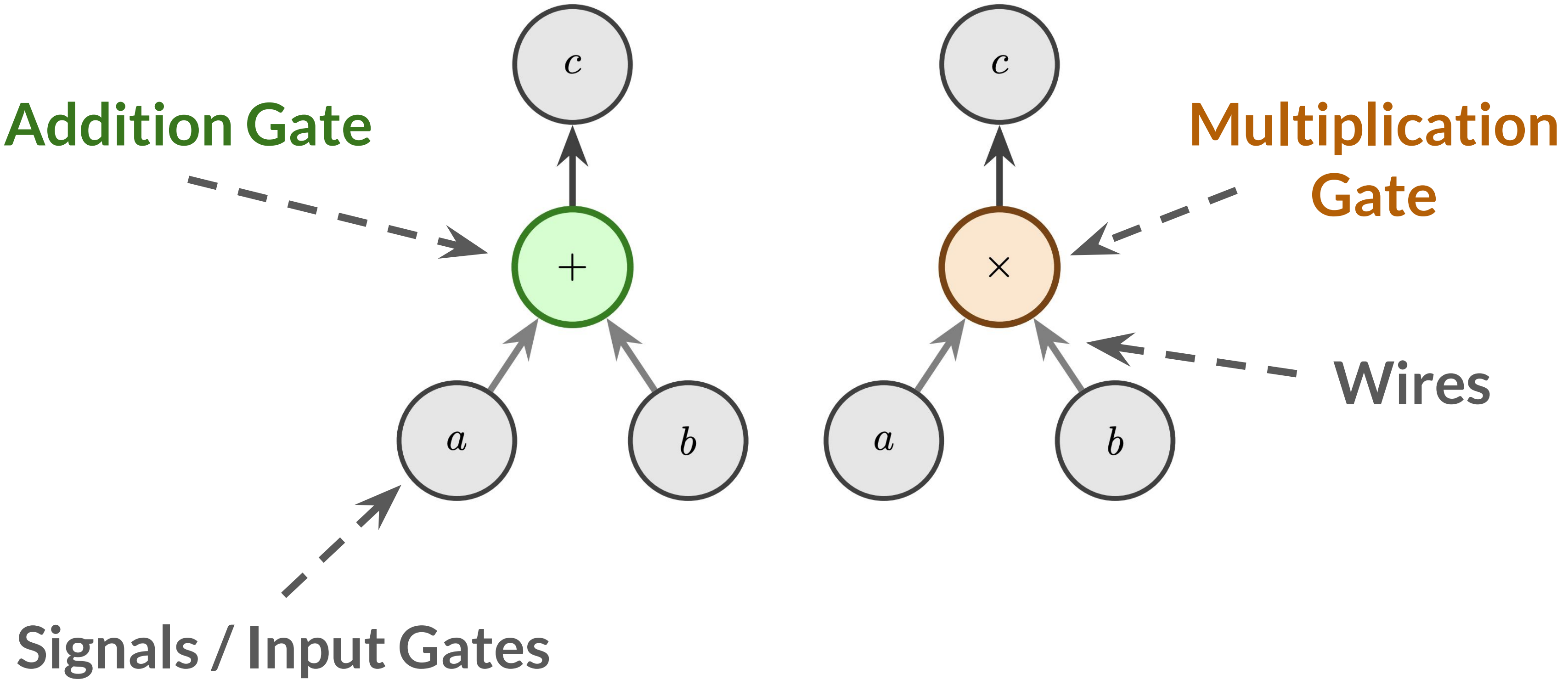
Definition

Suppose the "size" of a relation is $|C|$. (*strong*) **SNARK** is a NARK with **logarithmic** verifier and proof size:

$$\texttt{len}(\pi) = O_\lambda(\log |C|), \quad \texttt{time}(V) = O_\lambda(|x|, \log |C|)$$

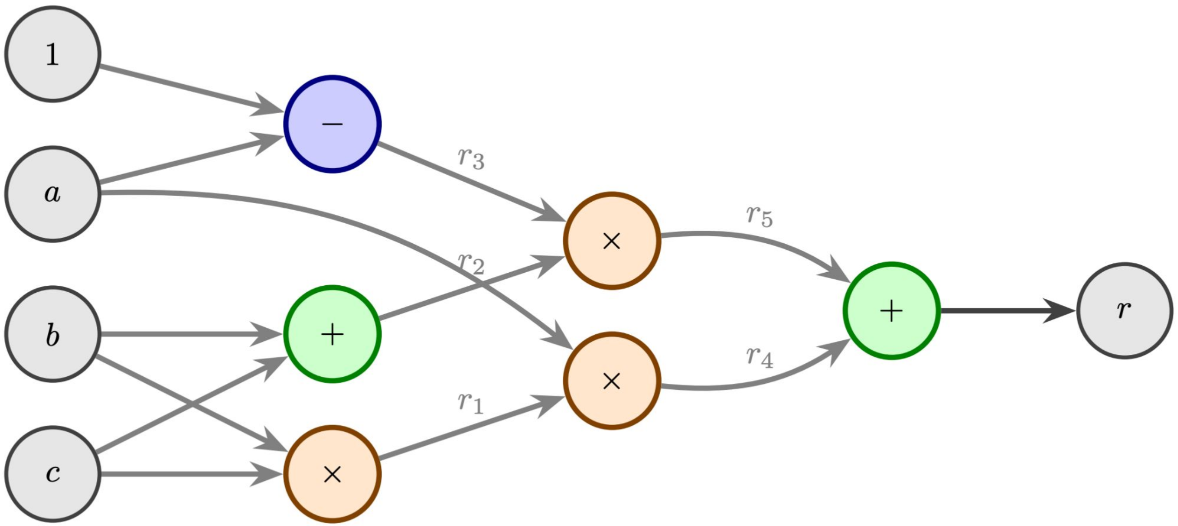
Note. SNARK is not necessarily ZK! If that is the case, the SNARK is naturally called the **zk-SNARK**.

How to measure size? Arithmetic Circuits



How to measure size? Arithmetic Circuits

Connect gates
and wires to get an
arithmetic circuit
C(***x***, ***w***)



Fact. Any NP relation's verifier can be implemented using some arithmetical circuit **C** over finite field F_p (and F_2 in particular).
Relation size = |**C**| = # of **gates** in **C**.

How Circuits are written in practice

What do you think this program computes? (written in *Circom*)

```
template ???() {
    signal input in;
    signal output out;
    signal inv;
    inv <-- in != 0 ? 1/in : 0;
    out <== -in * inv + 1;
    in * out === 0;
}
```

How Circuits are written in practice

Obviously, checking whether the element is 0! :)

```
template IsZero() {
    signal input in;
    signal output out;
    signal inv;
    inv <-- in != 0 ? 1/in : 0;
    out <== -in * inv + 1;
    in * out === 0;
}
```

Key Idea: it is not a language of execution, but verification!

Why writing circuits is weird?

- Operator === imposes constraint.
- Operator == checks equality of constant variables.
- Operator <-- assigns the value to variable *off-circuit*.
- Only addition/subtraction/multiplication are allowed.
- No comparison operators.
- Only multiplication of two variables is allowed.
- No variable-sized loops!
- All variables are finite field elements.
- No classes, generics, interfaces, or any syntax sugar!

...and if you mess something up, your system might be completely insecure!

The Trivial SNARK is not SNARK

(a) *P* sends ***w*** to *V*.

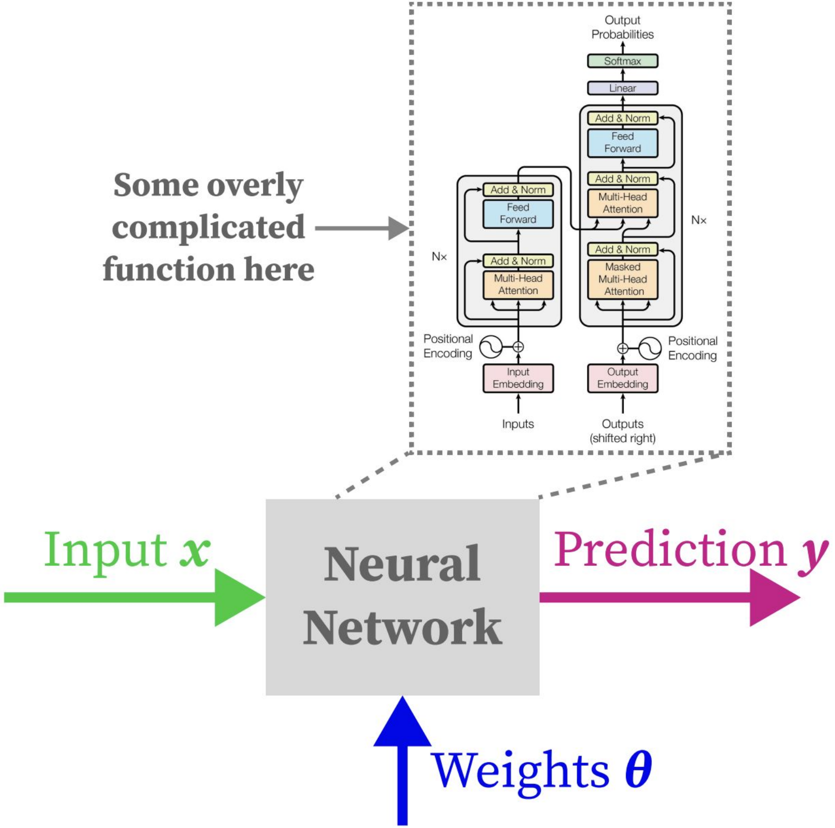
(b) *V* checks whether **C**(**x**, ***w***) = 0 and accepts if so.

Fun observation: this is a totally valid NARK.

However, this is not zk-NARK nor SNARK!

1. ***w*** might be secret: this is clearly violated.
2. ***w*** might be too-large: *V* has no time to *read* it!
3. |*C*| might be too-large: *V* has no time to *compute*!

Example: Zero-Knowledge Machine Learning



Goal: for the given $\boldsymbol{x}$, $\boldsymbol{y}$, weights $\boldsymbol{\theta}$, and model F, prove that:

$$\boldsymbol{y} = F(\boldsymbol{x}; \boldsymbol{\theta})$$

Zero-Knowledge Machine Learning

User *U*

OpenAI

x = "Windows or Linux?"

y = "Linux of course!"

$y = F(x; \theta)$

F

Problems:

1. How can we be sure that y was indeed computed by x using F?
2. How can U do *(1)* without running F and knowing θ?

Zero-Knowledge Machine Learning

💡 **Answer**: use zk-SNARKs! ⓪ $(\sigma_p, \sigma_v) \leftarrow$ `Setup`(F)

σ_v σ_p

① **x** = "Windows or Linux?"

③ **y** = "Linux of course!" + **π**

② $\pi \leftarrow \texttt{Prove}(\sigma_p, \mathbb{x} = (x, y), \mathbb{w} = \theta)$

④ $\texttt{Verify}(\sigma_v, \pi, \mathbb{x} = (x, y))$? ← – – **this is MUCH faster than computing *F*!**

🤯 **Fact:** Using SOTA proving systems, you can verify proof π in constant-time ***O(1)***! (relates not only to Machine Learning)

this is how π looks in practice - - - - - - ➤

proof.json


```
{
"proof": {
"pi_a": [
"47058017115654770468371195107739881730919574172707669183674412442920479800 64",
"14008115995489042379593199896964816349631620264393830590521359762731205641 67",
"1"
],
"pi_b": [
[
"12538508168416900299033726521685163817792614632620657244409429354131980454661",
"10914283679966848917795247355212516197618338956682374874239005506750384424444"
],
[
"11504632457518572930719312464170675169899321263873993433191427524966381618623",
"15524163713890313070296837080299781036987071183397727452907670321368057103914"
],
[
"1",
"0"
]
],
"pi_c": [
"260996700533282086084038116247679709285710726946875725263543647585988798998",
"142784280692542509392927046961757487190318591660754511827073317135139694032 99",
"1"
],
"protocol": "groth16",
"curve": "bn128"
},
"publicSignals": {
"r": "18"
}
}
```


Problem: typically, proving times are *very bad*! (or **V** is slow: $O(\sqrt{n})$)

See our solution to this problem.

Other applications

I have not mentioned numerous other applications:

- Scaling blockchain infrastructure (zk-rollups).
- Zero-Knowledge Virtual Machines (zkVM).
- Confidential assets.
- Identity protocols (proof-of-passport-validity, proof-of-humanity by scanning iris).
- ...

ZKP in the Wild

Why zk-SNARKs should exist?

How can ***V*** be ensured by ***P*** with time less than linear?

Lemma

(*Schwartz-Zippel Lemma*). For any multivariate polynomial $f \in F[T_1, \ldots, T_n]$, the following holds:

$$\Pr_{(r_1,\ldots,r_v)\leftarrow_\$ \mathbf{S}}[f(r_1,\ldots,r_v)=0] \le \frac{\deg f}{|\mathbf{S}|}, \quad \mathbf{S} \subseteq \mathbb{F}^v$$

(the statement is trivial for univariate polynomials)

Corollary. Checking equality of two polynomials can be done by picking a random point and comparing evaluations!

Why zk-SNARKs should exist?

💡 **Idea.** ***P*** can "*encode*" the arithmetic circuit instance into large polynomials and ***V*** can "*ask to open*" values of polynomials at random points. Then, ***V*** checks relations between these polynomials to ensure correctness.

Example

Suppose ***P*** wants to convince ***V*** that $f \in F[T]$ vanishes over certain subset Ω of size k over finite field F. Note that in such case:

$$f(T) = q(T)Z_\Omega(T),\ Z_\Omega(T) = \prod_{u \in \Omega}(T - u)$$

Zero-Test "SNARK"

Prover ***P***

"Verifier" ***V***

② Send com(***f***), com(***q***) →

③ Query point ***r*** ↔

① $q(T) \leftarrow f(T)/Z_\Omega(T)$

④ Learns ***q***(***r***), ***f***(***r***) and accepts iff ***f***(***r***) = ***q***(***r***)Z_Ω(***r***)

Note

P time: Quasilinear.

V time: O(log ***k***) + 2 queries.

Modern Protocols: Poly-IOP

(*Polynomial Interactive Oracle Proofs*). ***P*** gives oracles to ***V*** to query polynomials (example on the previous slide).

OK, I believe it is time to introduce ***awesome*** namings used for cryptographic protocols. The most famous Poly-IOP is:

PlonK'19 (**P**ermutations over **L**agrange-bases for ***O**ecumenical*🤨 **N**on-interactive arguments of **K**nowledge)

Improvements:
(not all are Poly-IOPs)

UltraPlonK **TurboPlonK** **aPlonK** **HyperPlonK** **Honk** **Goblin PlonK**

Modern Protocols: SumCheck-based

(*Sumcheck-based approaches/Multilinear IOP*). Proofs are based on effective IP for the following equation:

$$\sum_{b_0\in\{0,1\}} \sum_{b_1\in\{0,1\}} \cdots \sum_{b_v\in\{0,1\}} f(b_0,\dots,b_v) = H, \; f \in \mathbb{F}[T_1,\dots,T_v]$$

Very effective and simple!

GKR'08

Spartan'19 (there is a SuperSpartan'23 as well!)

zkGPT'25 (used for zkML)

Modern Protocols: Vector IOPs

(*Vector IOPs*). Proofs are typically based on Merkle Tree commitments and Error-Correction Codes.

Transparent setups, security based on hash collision-resistance and security of **FRI’18**. Oh, the name…

Fast **R**eed-Solomon… **I**nteractive Oracle Proof of Proximity

zk-STARK’18
Orion’22

Modern Protocols: Vector IOPs

They are so important that there is even a recent 1 million $ prize for solving ECC proximity gaps conjectures!

The Proximity Prize

initiative by
ethereum foundation

$1,000,000

in prizes to prove (or disprove!) Reed-Solomon proximity gaps conjectures—more info soonTM

An initiative by the Ethereum Foundation to advance the foundations of modern zkVMs.

Link
and
this one

Modern Protocols: EC-based

Pairing-based. ***Examples***: Pinocchio'13, Groth'16, Pari'24.

Based on the bilinear pairing defined over elliptic curve using some algebraic geometry construction.

$$e(\pi_A, \pi_B) = e(g_1^{\alpha}, g_2^{\beta}) e(\pi_{\mathrm{IC}}, g_2^{\gamma}) e(\pi_C, g_2^{\delta})$$

DL-based. Discrete-log based zk-SNARKs work over arbitrary groups. They have slow verifiers, but succinct proofs. Unfortunately, all are non-quantum-resistant

Examples: **Bulletproofs**'17, **Bulletproofs+**'20, **Bulletproofs++**'22.

Any Questions?

Resources

–

As was requested after the lecture, here are some resources to study cryptography and zero-knowledge!

My personal favourite about applied cryptography in general: "***A Graduate Course in Applied Cryptography***" by Dan Boneh and Victor Shoup:

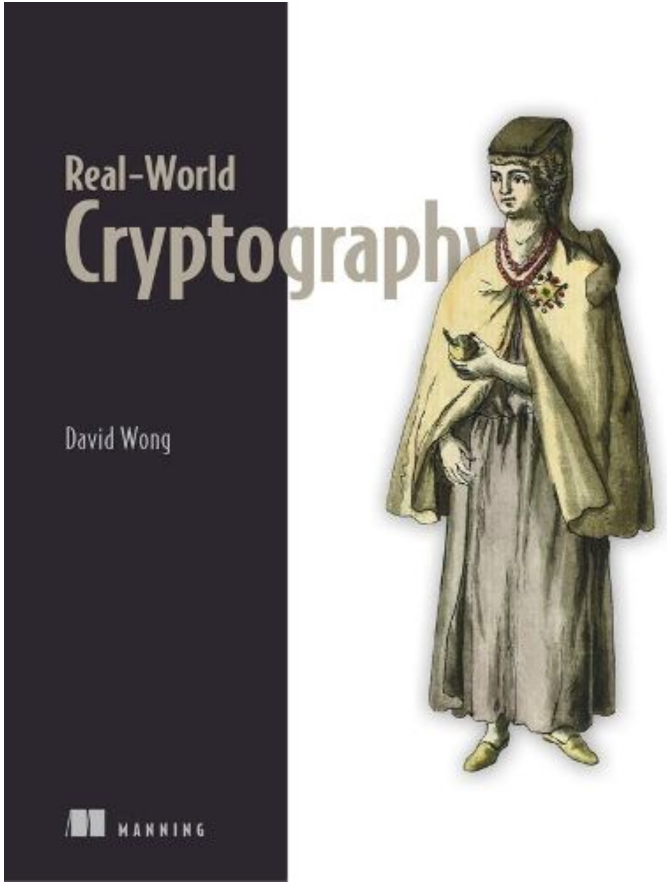
https://toc.cryptobook.us/

Warning: The book is hard, but it is worth it!

Resources

Very starter-friendly book:

"***Real-World Cryptography***" by David Wong

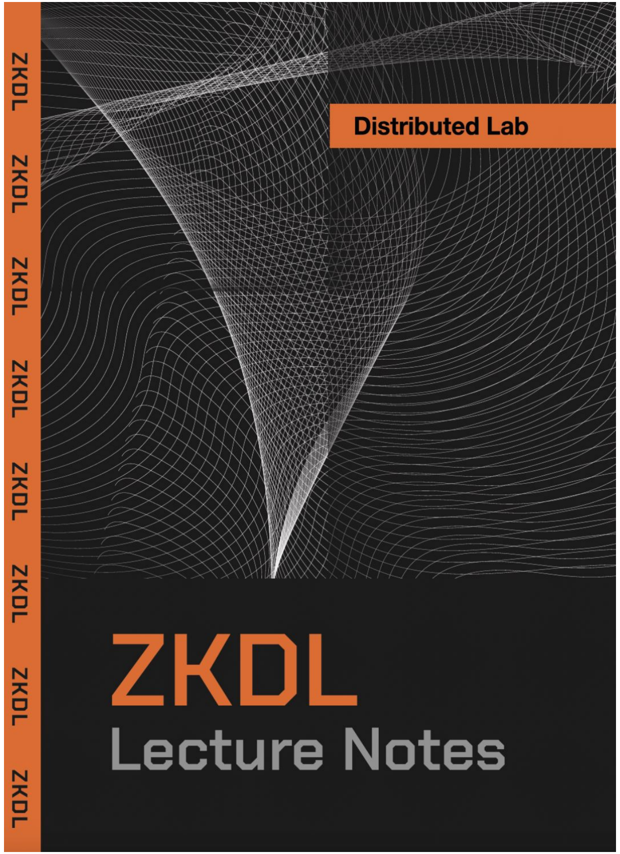


Resources

–

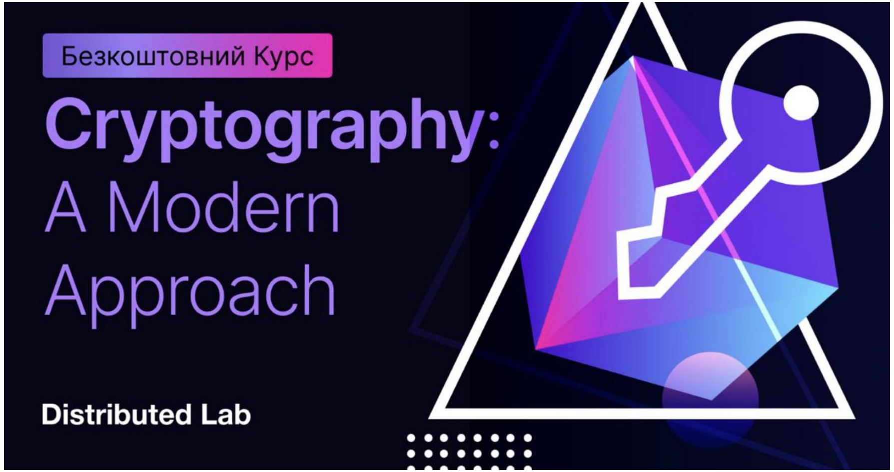
"*ZKDL Lecture Notes*" by Distributed Lab!
I am the main editor of the book, so if you have any questions – reach out to me!

https://zkdl-camp.github.io/



Resources

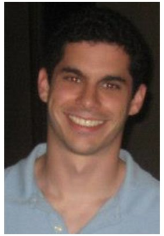
"*Cryptography: A Modern Approach*" by Distributed Lab.
This is our course about general cryptography that, more or less, contains all modern constructions and topics in Cryptography: https://github.com/distributed-lab/crypto-lectures



Resources

"***ZKMOOC***": one of the best courses in zero-knowledge proofs organized by top cryptographers: Dan Boneh, Shafi Goldwasser, Justin Thaler etc.: https://rdi.berkeley.edu/zk-learning/

Instructors

				
Dan Boneh	Shafi Goldwasser	Dawn Song	Justin Thaler	Yupeng Zhang
Stanford	UC Berkeley	UC Berkeley	Georgetown University	Texas A&M University

Resources

“**Alin Tomescu’s Website**”: although a lot of blogs are still in progress, many of them are awesome: see Groth16 or Spartan blogs! https://alinush.github.io/