Zero knowledge proofs

The basics

András Szabolcsi

Properties of zero-knowledge proof

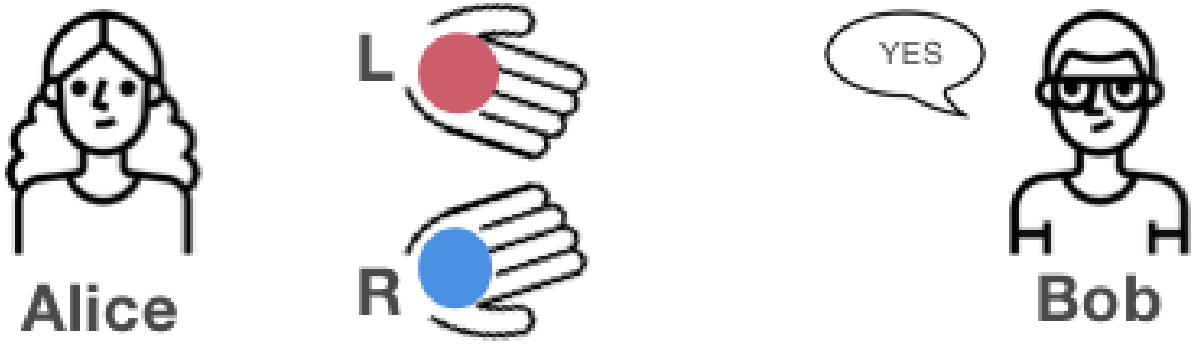
- Soundness — everything that is provable is true
  Simply: Alice's proof systems are truthful and do not let her cheat.
- Completeness — everything that is true has a proof.
  Simply: Alice's proof systems convince Bob that she found Waldo.
- Zero-Knowledge — only the statement being proven is revealed.
  Simply: Alice's proof systems prove her victory to Bob, without revealing her knowledge

Types

- Interactive
- Non-Interactive
- Proof of Knowledge (POK) – Simple (or purpose)
- (Statistical ZK-proofs)
- (Bulletproofs)
- Sigma protocols – Generic (ZKP Systems)

Interactive

- In a colorblind world, how can Bob convince Alice about he can see colors?
- Statement: Bob: I can see colors
- Setup: Alice hold a red and a blue ball
- * Challenge: Alice secretly may or may not exchange balls between her hands, then show to Bob.
- * Response: Bob should tell if she exchanged or not
- Repeat challenge-response as many time as you want. 1-(½)^n



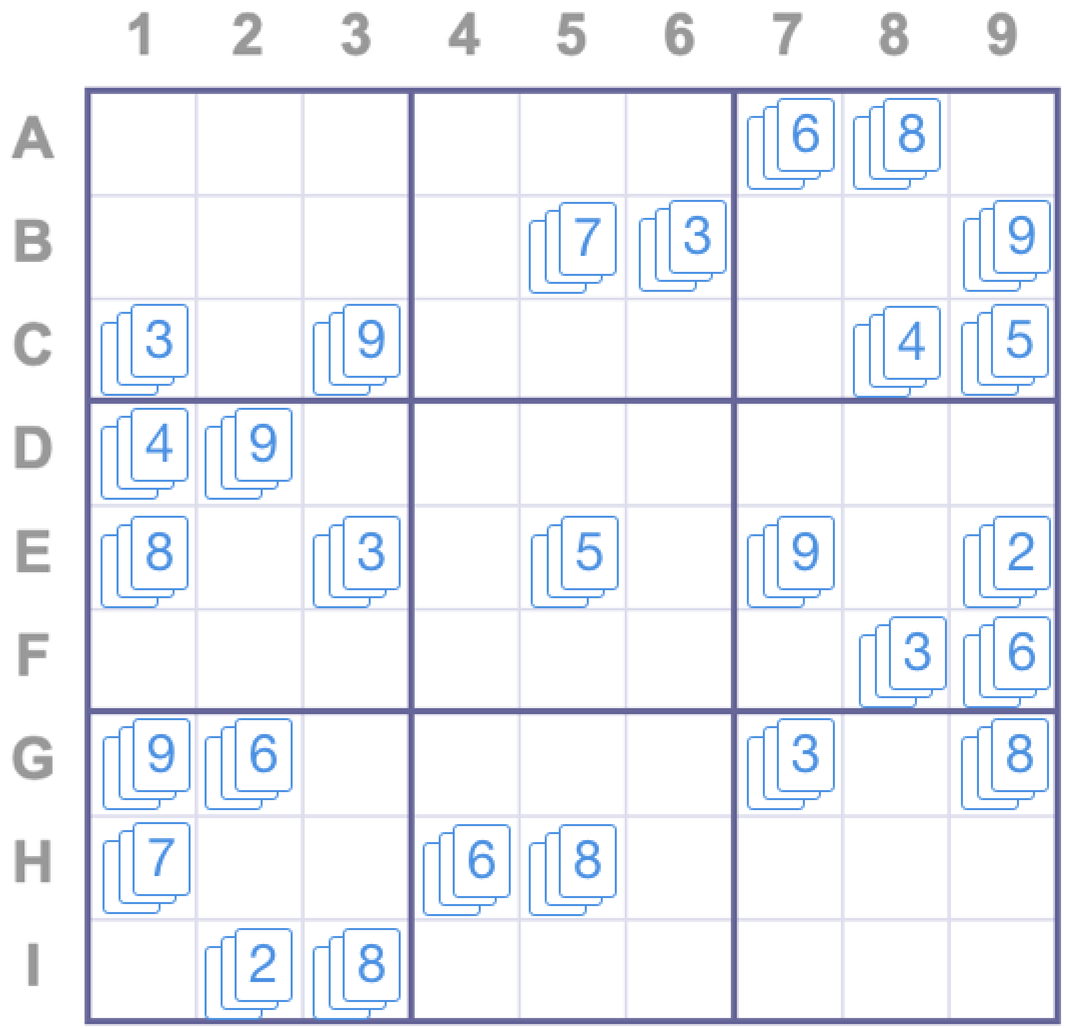
Non-interactive

- Doesn’t require the former challenge-response dynamic between Alice and Bob.
- Alice wants to prove to that she has solved a Sudoku puzzle they have not been able to solve.
- Alice builds a tamper-proof machine that executes the proof to Bob and friends. Alice’s machine follows a specific, *publicly verifiable* protocol with the following logic.

	1	2	3	4	5	6	7	8	9
A							6	8	
B					7	3			9
C	3		9					4	5
D	4	9							
E	8		3		5		9		2
F								3	6
G	9	6					3		8
H	7			6	8				
I		2	8						

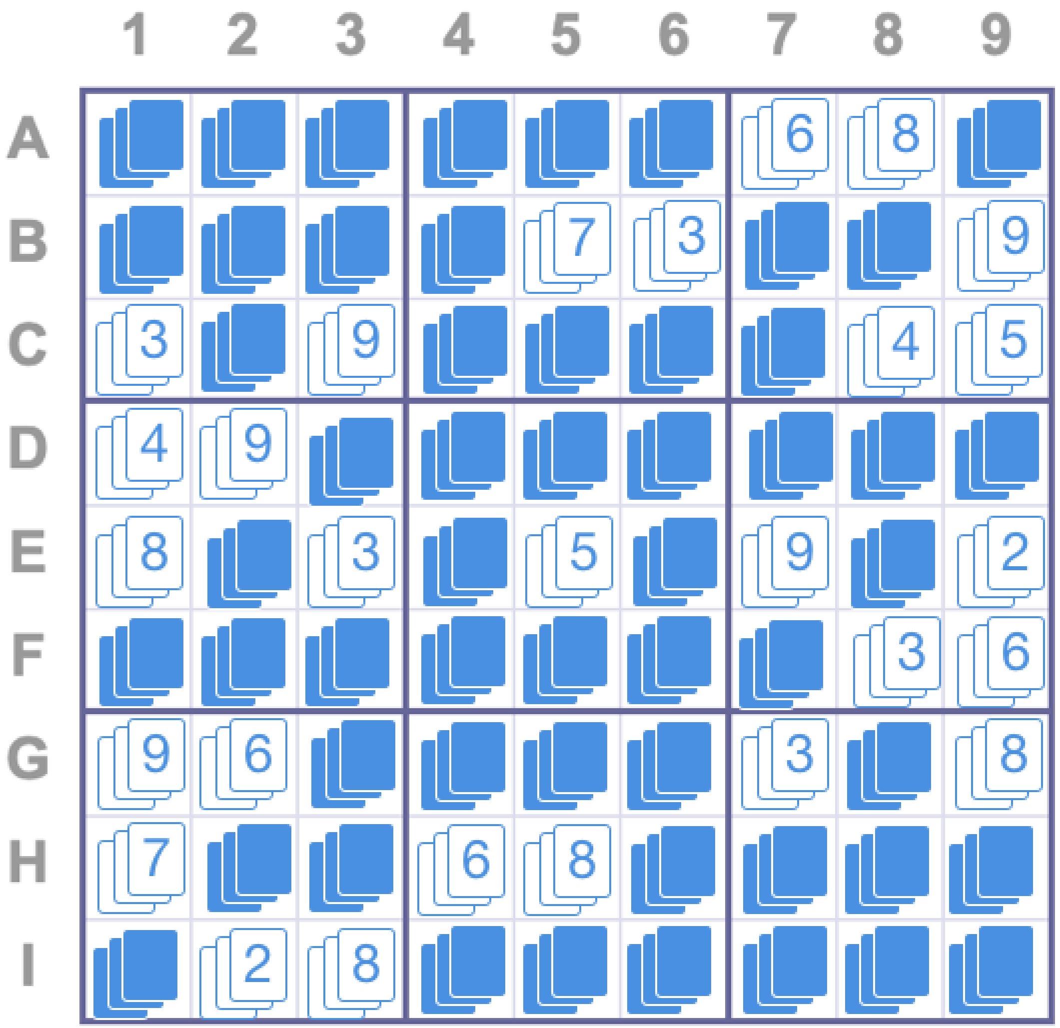
NZKP example - 1. step

- Reproduces the original, unsolved puzzle in the machine. For each cell with an existing value, it automatically lays three face-up cards with the corresponding number, e.g. cell C3 has 3 number 9 cards.



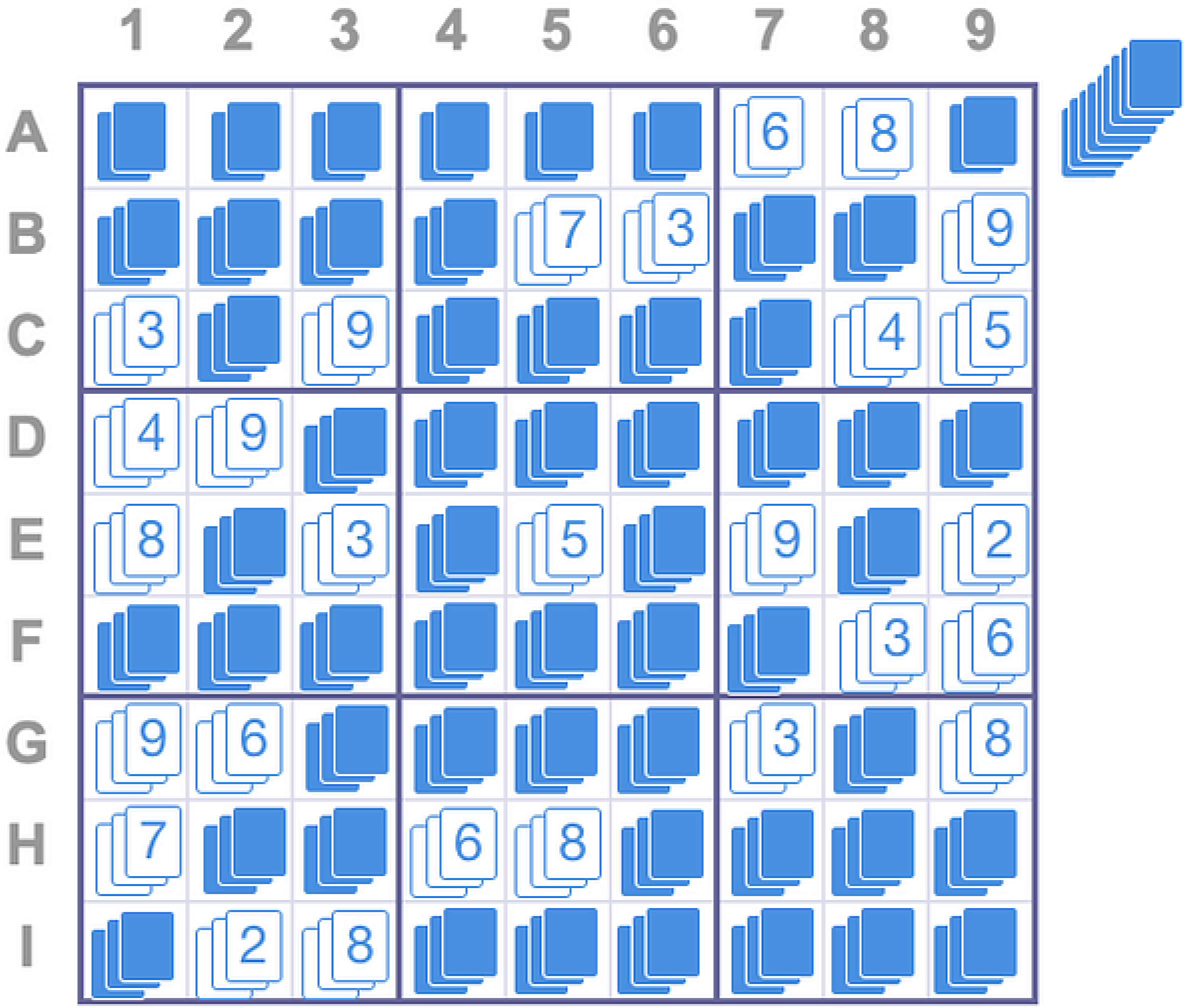
NZKP example - 2. step

- Encodes her solution by having the machine lay her answers face down on the grid. Of course, the machine prevents Bob from simply flipping over the cards in their cells.



NZKP example - 3. step

- Bob can now interact with the machine. Starting with each row, Bob randomly chooses one card in each cell, from the top, the middle, or the bottom.



NZKP example - 4. step

- The machine takes the chosen cards and makes a face down, 9-card-packet for each row.
- This action is repeated for each column as well.
- Finally, the remaining cards are sorted into one packet for each 3x3 grid.
- In total, the machine makes 27 packets.
- Then the machine randomly shuffles the cards in each packet, before giving the packets back to Bob.
- Bob flips the cards over and verifies that each packet contains the numbers 1 through 9 without any numbers missing or duplicated.

Simple ZKPs

- Usually built only for one purpose.
  For example, I want to prove that I have/know a secret.
- Accumulators
- Ring signatures
- Proof of Knowledge
- Usually EC or Lattice-based

Let some math

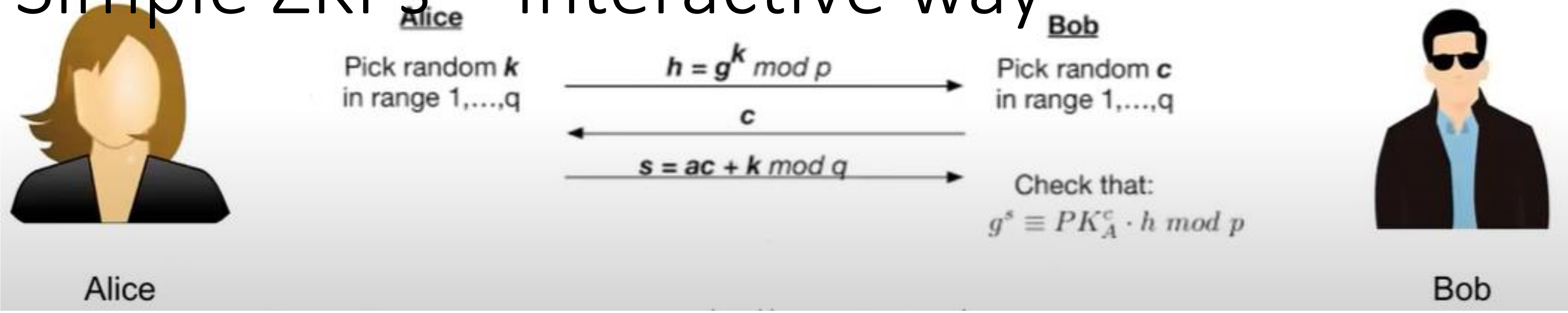
- Let “a” be a secret key, in the world of elliptic cryptography, it can be a large random number, or any secret information
- The corresponding public key is

$$PK_a = g^a \ (mod\ p)$$

- Some properties

$$g^{(ac)} = (g^a)^c = PK^c \ (mod\ p)$$
$$g^{(a+c)} = g^a \cdot g^c = PK_a \cdot PK_c (mod\ p)$$

Simple ZKPs – Interactive way



- a and $PK_a = g^a \ (mod\ p)$
- Choose random k
  k and $PK_k = g^k \ (mod\ p)$ and sends PK_k to Bob (commitment)
- Knows c
- Calculates: $s = ac + k \ (mod\ p)$
  Sends s to Bob

- Knows: PK_a (statement: Alice knows a)
- Knows PK_k
- Choose random c and sends to Alice (challenge)
- Knows s
- $g^s = PK_a^c \cdot PK_k \ (mod\ p)$
- $g^s = (g^a)^c \cdot g^k = g^{(ac+k)} \ (mod\ p)$

Simple ZKPs – Non-interactive way

- Alice:
  - Her secret is “a” and $PK_k = \ g^a \ (mod\ p)$
  - Pick a random number v and $PK_v = g^v$ (Commitment)
  - Calculate her own challenge $c = Hash(g||a||PK_k||PK_v)$ (Challenge)
  - Calculate $r = v - c * a$
  - Sends PK_v, c, r to Bob
- Bob:
  - Calculate $V_{verify}, V_{verify} = g^r \cdot (g^a)^c$
  - If $V_{verify} = PK_v$ then it ok

Simple ZKPs - Accumulators

- use a BL12 curve, has two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$
- Create a random secret is "sk" and $PK = sk \cdot G_1$
- $a_0 = G_1$
- To add y_1, we add to the accumulator with:
  - $a_1 = (y_1 + sk) \cdot G_1$
- To add y_2, we add to the accumulator with:
  - $a_2 = (y_2 + sk) \cdot a_1 = (y_2 + sk) \cdot (y_1 + sk) \cdot G_1$
- To remove y_1 from a_2:
  - $a_3 = \frac{1}{y_1 + sk} \cdot a_2 = (y_2 + sk) \cdot G_1$

Simple ZKPs – Accumulators – WHY??

- Let“sk” is a secret, $PK = sk \cdot G_2$ and “a” be the accumulator value
- “w” is a witness proof that a given value (y_1) is in the accumulator
- $w = \frac{1}{y_1+sk} \cdot a$
- Check: $e(w, y_1 \cdot G_2 + PK) \cdot e(-a, G_2) = 1$ is it true?
  - $e\left(\frac{a}{y_1+sk}, y_1 \cdot G_2 + PK\right) \cdot e(-a, G_2) = 1$
  - $e\left(\frac{a}{y_1+sk}, y_1 \cdot G_2\right) \cdot e\left(\frac{a}{y_1+sk}, sk \cdot G_2\right) \cdot e(-a, G_2) = 1$
  - $e\left(\frac{a \cdot y_1}{y_1+sk}, G_2\right) \cdot e\left(\frac{a \cdot sk}{y_1+sk}, G_2\right) \cdot e(-a, G_2) = 1$
  - $e\left(\frac{a \cdot (y_1+sk)}{y_1+sk}, G_2\right) \cdot e(-a, G_2) = 1$

zk-SNARK

- **Z**ero-**K**nowledge **S**uccinct **N**on-Interactive **Ar**gument of **K**nowledge
- Pros:
  - Small proof size
  - Fast
  - Generic
  - Trustlessly verified by anyone
- Cons:
  - Trusted setup
  - Susceptibility to quantum computing attacks (ECC)
  - Setup and proof generation is computationally-intensive process (Time and/or space complexity)

zk-STARK

- **Z**ero-**K**nowledge **S**calable **T**ransparent **Ar**gument of **K**nowledge
- Pros:
  - No need for a trusted setup
  - Fastest (Scalable)
  - Higher security (considered resistant to quantum computing attacks)
- Cons:
  - Large proof size
  - Lower adaptation

ZKP Systems

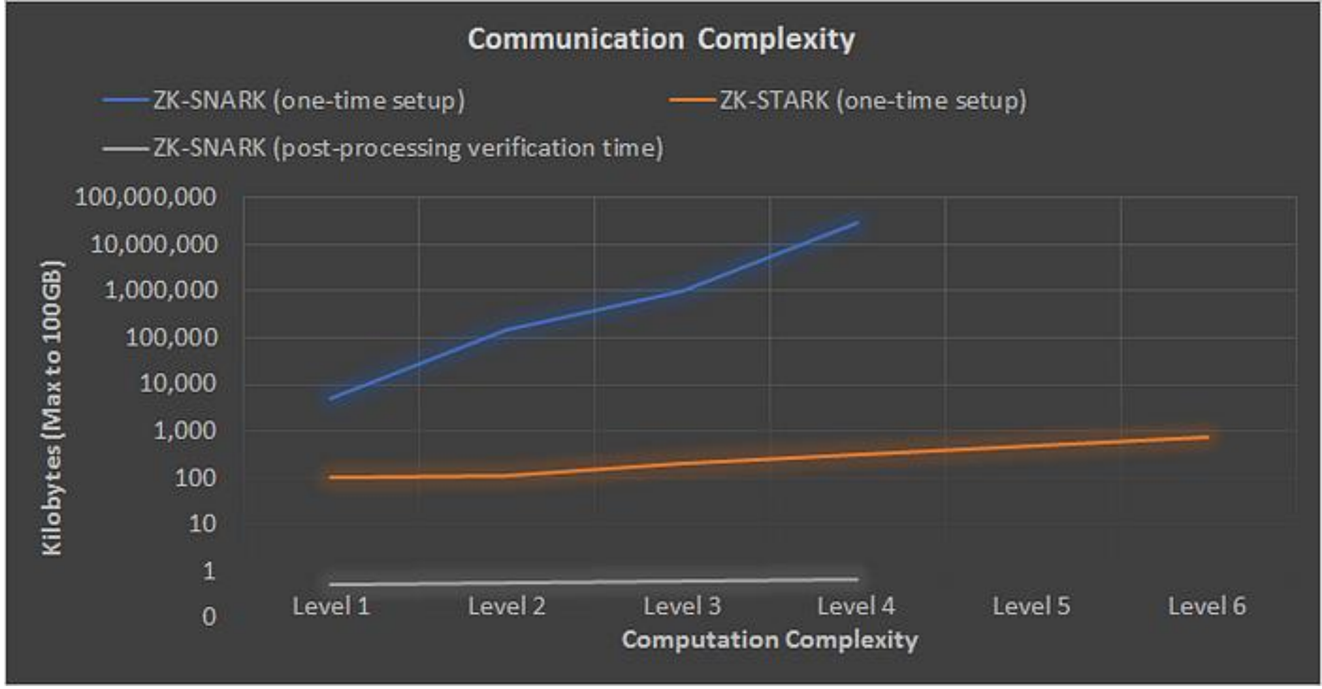
- zk-SNARK
- zk-STARK

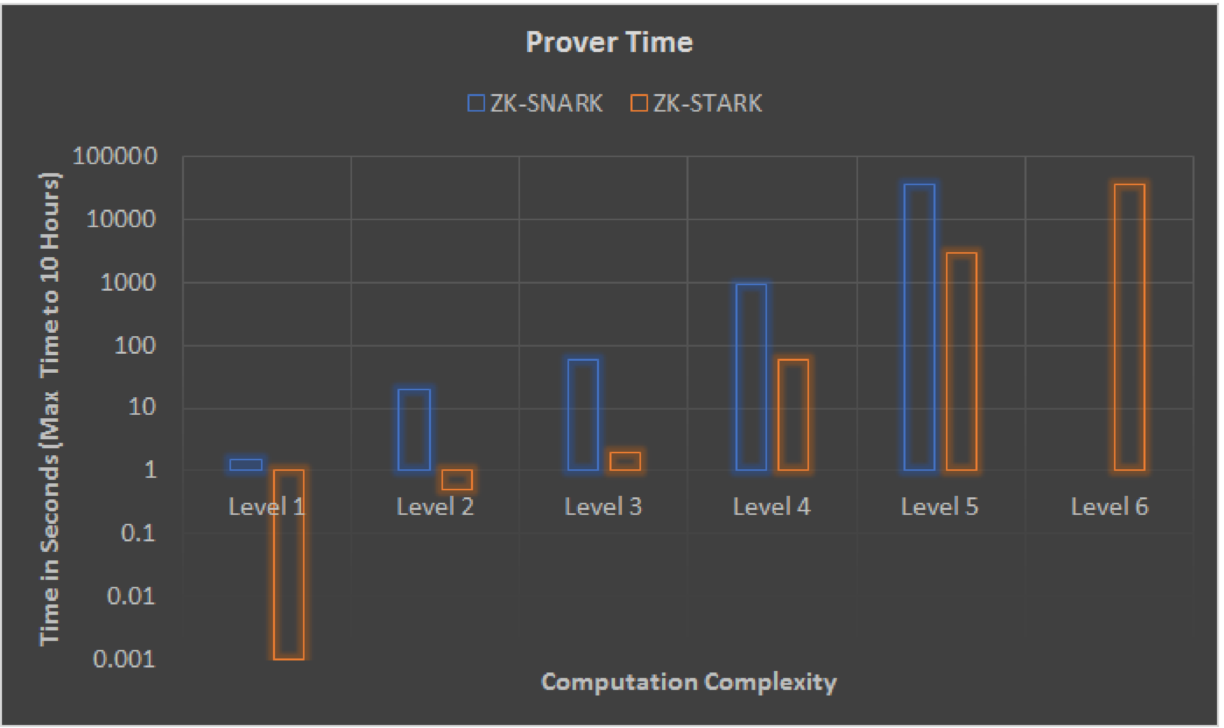
Zero-knowledge proof (ZKP) systems

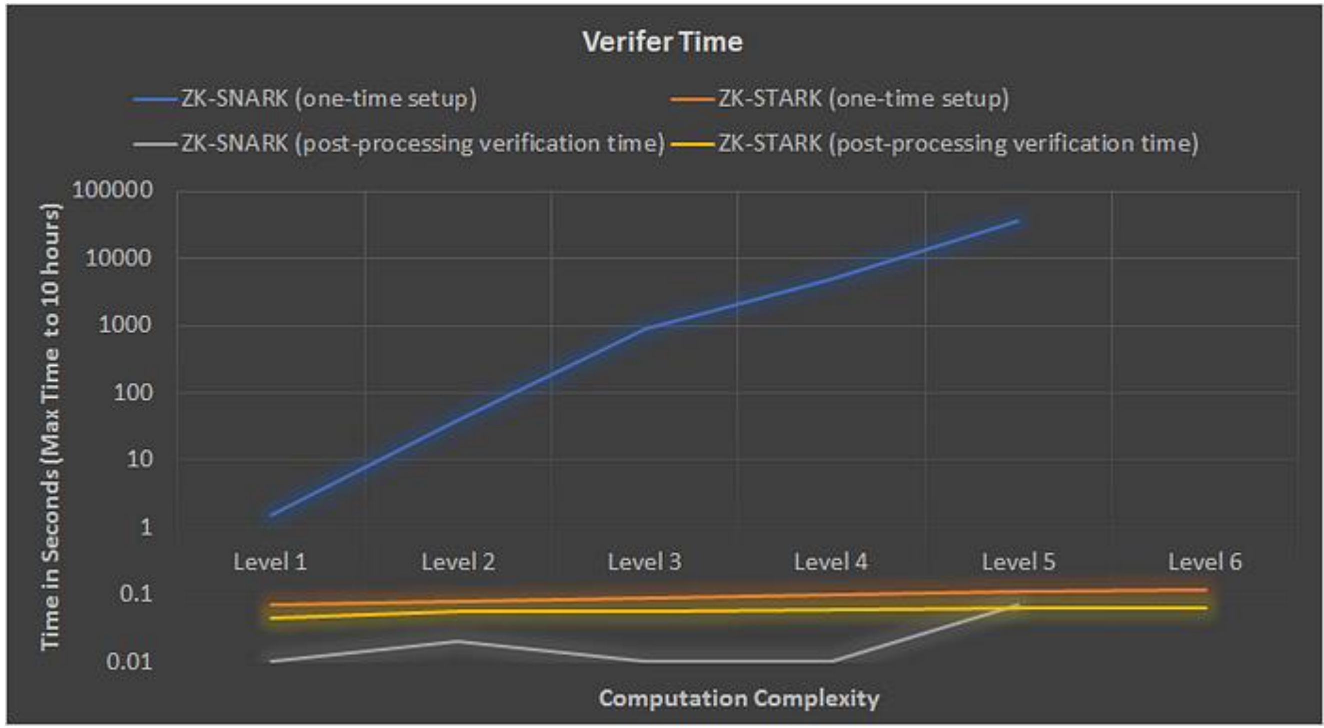
ZKP System	Publication year	Protocol	Transparent	Universal	Plausibly Post-Quantum Secure	Programming Paradigm
Pinocchio[36]	2013	zk-SNARK	No	No	No	Procedural
Geppetto[37]	2015	zk-SNARK	No	No	No	Procedural
TinyRAM[38]	2013	zk-SNARK	No	No	No	Procedural
Buffet[39]	2015	zk-SNARK	No	No	No	Procedural
ZoKrates[40]	2018	zk-SNARK	No	No	No	Procedural
xJsnark[41]	2018	zk-SNARK	No	No	No	Procedural
vRAM[42]	2018	zk-SNARG	No	Yes	No	Assembly
vnTinyRAM[43]	2014	zk-SNARK	No	Yes	No	Procedural
MIRAGE[44]	2020	zk-SNARK	No	Yes	No	Arithmetic Circuits
Sonic[45]	2019	zk-SNARK	No	Yes	No	Arithmetic Circuits
Marlin[46]	2020	zk-SNARK	No	Yes	No	Arithmetic Circuits
PLONK[47]	2019	zk-SNARK	No	Yes	No	Arithmetic Circuits
SuperSonic[48]	2020	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Bulletproofs[24]	2018	Bulletproofs	Yes	Yes	No	Arithmetic Circuits
Hyrax[49]	2018	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Halo[50]	2019	zk-SNARK	Yes	Yes	No	Arithmetic Circuits
Virgo[51]	2020	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
Ligero[52]	2017	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
Aurora[53]	2019	zk-SNARK	Yes	Yes	Yes	Arithmetic Circuits
zk-STARK[54]	2019	zk-STARK	Yes	Yes	Yes	Assembly
Zilch[35]	2021	zk-STARK	Yes	Yes	Yes	Object-Oriented

SNARK vs STARK

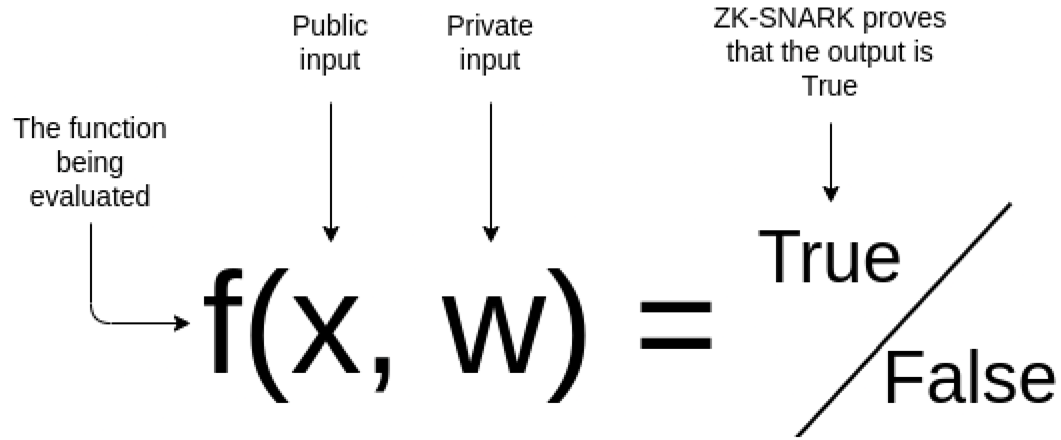
Levels are referred to arithmetic circuit complexity







SNARK - Deep dive?



Given a function ***f(x)***, and public output ***y***, using zkSNARK, one can generate a proof to demonstrate the knowledge of a solution ***s***, without revealing the value of ***s***.

Given: ***f(x) = y***

Produce ***s***, such as that ***f(s) = y***

SNARK consumes the “code” of the function ***f(x)*** and public input ***y*** as input, and produces the zk proof as the output.