Journey to Zero-Knowledge

Davide Galassi
davxy@datawok.net

2023-12-30

Contents

1 Introduction

Zero-Knowledge Proofs (ZKP) represent a fascinating and influential concept within the realm of cryptographic protocols.

At a glance, a ZKP enables one party to demonstrate the correctness of a statement to another party without revealing any details beside the validity of the claim itself.

ZKPs find applications in various areas, including secure authentication protocols, blockchain systems, and secure computation, among others.

Another motivation is philosophical. The notion of a proof is basic to mathematics and to people in general. It is a very interesting and fascinating question whether a proof carries with it some knowledge or not.

In this paper we incrementally construct a path leading from the classic mathematical notion of proof to those that share zero knowledge. One of the goals is to demystify the subject, maintaining the necessary rigor while ensuring accessibility for a broader audience.

2 Classical Proofs

2.1 Deductive Reasoning

Deductive reasoning is a fundamental method of logical thinking used across various disciplines, from philosophy and mathematics to computer science and law.

It involves deriving specific **conclusions** from a set of general **premises** or known facts. The strength of deductive reasoning lies in its ability to guarantee the truth of the conclusion, provided the premises are true, and the reasoning process is logically correct.

One of the earliest examples of deductive reasoning can be traced back to ancient Greek philosophers, particularly Aristotle, who formalized the syllogistic reasoning. A classic example of a syllogism is:

1. All men are mortal (premise).
2. Socrates is a man (premise).
3. Therefore, Socrates is mortal (conclusion).

This example captures the essence of deductive reasoning: if the premises are true and the reasoning is valid, then the conclusion must also be true.

2.1.1 Deductive Reasoning in Mathematics

In the realm of mathematics, deductive reasoning takes a more structured form known as a mathematical proof.

A **mathematical proof** is a logical argument presented systematically to verify the truth of a mathematical *statement*. Here, deductive reasoning is used to derive conclusions from a set of *axioms* (self-evident truths) and previously established *theorems* (proven statements) by using some *inference rules* which can be applied in the specific context.

By setting the *conclusion* as the *statement* we want to prove and the *premises* as the set of *axioms* and previously proven *theorems*, we can define a proof as a finite length string encoding the set of logical derivations that incrementally drives the reader from the premises to the conclusion.

As all the proof reasoning relies on the basic properties of Boolean algebra, it is implicit that Boolean logic axioms and theorems hold as premises for any reasonable proof technique we'll analyze in this document.

2.2 Validity and Soundness

Definition. *A proof is **valid** if the conclusion logically follows the premises, regardless of whether those promises are true or not.*

Definition *A proof is **sound** if it is valid and all of its premises are true.*

Example: Sound proof.

- *Premises*:
  - A, B and C are three sets such that $A \cap B \subseteq C$;
  - $x \in B$;
- *Conclusion*: $x \notin A \setminus C$.
- *Proof*:
  - $x \notin A \setminus C = \neg(x \in A \land x \notin C) = x \notin A \lor x \in C = x \in A \to x \in C$
  - $(A \cap B \subseteq C \land x \in B \land x \in A) \to x \in C$

Example: Valid but not sound proof:

- *Premises*:
  - All prime numbers are odd (wrong premise).
  - 2 is a prime number (as it has no divisors other than 1 and itself)
- *Conclusion*: 2 is odd
- *Proof*: The conclusion follows directly from the premises.

Since the conclusion can be derived from the premises, the proof is formally correct, but as the second premise is incorrect it is not sound.

The example emphasize how we can reach incorrect conclusions even though we constructed an apparently correct proof just because of a bad premise.

2.3 Proof Systems

A **proof system** is a formal and systematic approach to construct and evaluate proofs. It typically consists of the following key components:

- **Statement (x)**: assertion that one tries to prove.
- **Proof (π)**: evidence or logical steps that should establish the validity of the statement.
- **Prover (P)**: algorithm to construct a proof for the given statement ($P(x) = \pi$).
- **Verifier (V)**: algorithm that, given both the statement and the proof, decides whether the proof is valid. This algorithm outputs 1 if the proof is correct and 0 if the proof is not correct ($V(x, \pi) = 0|1$).

Clear rules and guidelines are essential to move proof construction and checking from the ambiguous domain of natural languages to the precise realm of formal languages.

Proof verification is often modeled as a **decision problem**. The decision is about determining whether a string x, which represents a candidate solution to a given problem, is a member of a language L, which is the set of all the correct solutions to the same problem. In this framework, the proof π drives the verification algorithm towards the final verdict.

For instance, consider the *Subset Sum Problem* (SSP). The language for this problem is represented as:

$$L = \{x = (S, t) \mid S \subseteq \mathbb{N} \text{ and } \exists V \subseteq S : \text{the sum of the elements in } V \text{ equals } t\}$$

A string $x = (S, t)$ belongs to the language L if and only if we can prove that there is a subset of S which adds up to t.

Key characteristics of a proof system (P, V) for decision problems:

- **Completeness**: $x \in L$ if and only if $V(x, \pi) = 1$.
- **Soundness**: $x \notin L$ if and only if $V(x, \pi) = 0$.
- **Efficiency**: $V(x, \pi)$ runs in polynomial time with respect to length of x.

Where appropriate, though this document we'll stick to the popular convention of referring to the *prover* as Peggy and the *verifier* as Victor.

2.4 Knowledge Sharing

In a *classical* proof system a proof that some assertion is true inherently reveals *why* it is true. This aspect is deeply rooted in how the classical mathematical proof systems work: Peggy shares all the logical steps to allow Victor to independently reach the same conclusion.

Follows that the classical proof provides more knowledge than just the mere fact that the statement is true, which is what Peggy and Victor are interested in the first place, regardless of the way this is proven.

For instance, consider the task of proving knowledge of the factorization of a natural number n. Peggy could simply share the list of its prime factors $\{p_i\}$ and the relative exponents $\{e_i\}$ to allow Victor to check if $n = \prod_i p_i^{e_i}$. In the end, not only Victor is convinced about the statement, but he also gains knowledge of the factorization of n.

The information which facilitates the construction of the proof is known as the **witness**. In a classical proof system the witness is always contained within the proof. However, in certain situations, Peggy may desire to keep the witness confidential while still proving the statement.

3 Interactive Proofs

An **interactive proof** (IP) system extends the classical notion of proof conceived as a static sequence of symbols to an *interactive protocol* where Peggy incrementally convinces Victor by actively exchanging messages.

The entire sequence of messages exchanged during the protocol execution is collectively referred to as the **transcript**. Two runs of the same protocol with the same inputs can produce different transcripts depending on whether the protocol is deterministic or probabilistic.

The idea emerged in the mid 80s from the work of Goldwasser, Micali, and Rackoff [GMR85]. Their seminal contribution not only formally defined the concept of interactive proof but also presented the first definition of *zero-knowledge* proof.

3.1 Sigma Protocols

An IP system with a four[1] messages transcript is called a **sigma protocol**. The name is inspired by the Greek letter Σ, which shape mirrors the sequence of the protocol's steps:

1. *Commitment*: Peggy initiates the protocol by sharing some parameters.
2. *Challenge*: Victor issues a challenge to Peggy.
3. *Response*: Peggy replies to Victor's challenge.
4. *Result* (optional): Victor sends a message with the verification outcome.

The names of these steps are chosen to reflect their functional roles in the execution of typical proofs, particularly in the ZK context. For the moment the detailed mechanics and purpose of each step are left open.

[1] some sources describe sigma protocols with just three messages

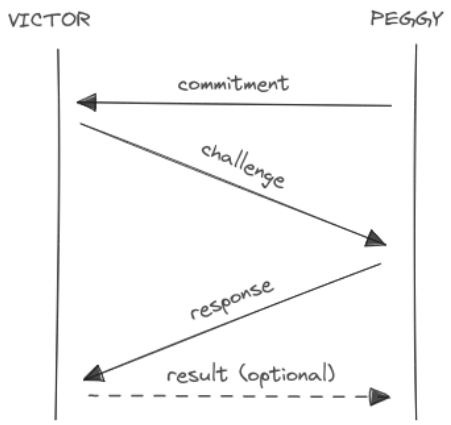

Figure 1: Sigma protocol

Sigma protocols are the most common type of protocols in the realm of interactive proofs. In fact, nearly every protocol described in this paper falls into this category.

3.2 Deterministic Interactive Proofs

A **deterministic** IP system is an interactive proof system which doesn't introduce any randomness in the protocol messages. In such a system, given the same inputs the transcript is completely predictable and reproducible.

As an example, consider the protocol where Peggy wants to prove that she knows that a number N is the product of two primes:

1. Peggy asserts she knows the factors of N.
2. Victors asks for the smaller factor.
3. Peggy sends the factor x.
4. Victor computes $y = \lceil N/x \rceil$, checks if $N = x \cdot y$ and that both x and y are primes via some deterministic algorithm.

Proposition. *Every deterministic IP can be mapped into a static classic proof and vice versa.*

- To convert a static proof into a deterministic IP, the two parties can just communicate to incrementally transfer one or more chunks of the proof. Note that a static proof is essentially a deterministic IP with a single message exchange.
- To convert a deterministic IP into a static proof, the *prover* constructs a string encapsulating the entire protocol transcript. This construction is

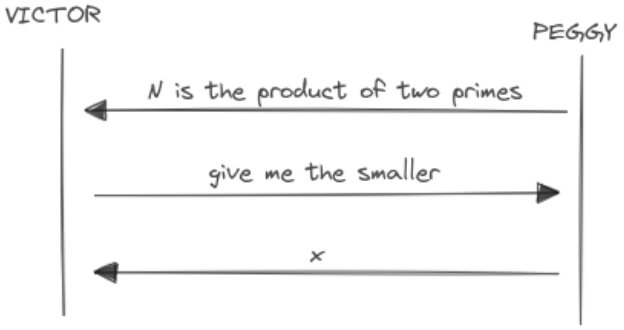

Figure 2: Multi message deterministic IP example

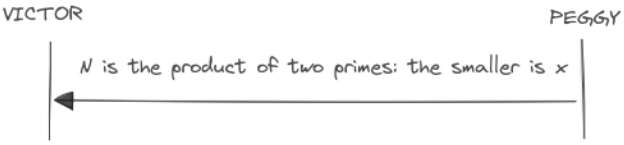

Figure 3: Single message deterministic IP example

feasible because of the deterministic nature of the transcript. Victor just needs to check if the transcript matches the expected one.

From this last result follows that deterministic IP systems are not more powerful than static proofs with respect to the set of languages that can be proven. Due to the limited interest on the deterministic IP systems, we'll save the formal definitions for the next section.

3.3 Probabilistic Interactive Proofs

In the probabilistic version of the proving system, the steps mirror those of the deterministic counterpart, but with additional elements of randomness introduced by either the prover, the verifier, or both.

The introduction of randomness into the protocol can, in some scenarios, lead to more efficient proofs or enable to prove en entire new class of languages which can't be proven using deterministic proof systems.

Definition [GMR85]. *A probabilistic IP system for a language L is a protocol (P, V) for communication between a computationally unbounded machine P and a probabilistic polynomial time machine V taking an input statement x, a message history (h_i) and randomness source(s) ($r_{p|v}$) to produce the next protocol*

message:

$$
\begin{aligned}
m_1 &= P(x, r_p, h_1 = \{\}) \\
m_2 &= V(x, r_v, h_2 = \{m_1\}) \\
m_3 &= P(x, r_p, h_3 = \{m_1, m_2\}) \\
&... \\
\end{aligned}
$$

Elaborating the definition:

- Peggy is assumed to have unbounded computational resources.
- Victor is assumed to operates within polynomial constraints relative to the size of the statement to prove.
- Given Victor's polynomial limitations, the number of messages exchanged between the two actors must also be polynomially bound.
- Both Peggy and Victor have access to a **private** random generator.

For convenience, from now on we'll refer to *probabilistic interactive proof* systems just as *interactive proof* (IP) systems.

Key characteristics of an IP system (P, V) for a language L:

- **Completeness**: If $x \in L$, then $V(x, \pi) = 1$ with high probability.
- **Soundness**: If $x \notin L$ then $V(x, \pi) = 1$ with negligible probability.
- **Efficiency**: Both the total computation time of $V(x, \pi)$ and the overall communication in (P, V) is polynomial with respect to length of x.

The probability ε of the verifier accepting a false statement is known as **soundness error** probability. If for a single protocol execution $\varepsilon < 1$, then is always possible to construct another protocol which runs the original one k times consecutively, thus exponentially reducing the error probability to an arbitrary value ε^k.

3.3.1 Proving Completeness

If Peggy knows the solution to a problem, she will eventually convince Victor as its responses will be always correct regardless of the number of protocol repetitions and the value of ε.

3.3.2 Proving Soundness

A proof of protocol soundness generally requires to construct an **extractor**, a hypothetical tool tailored for the specific protocol which allows Victor to extract some key information used by Peggy to construct the proof.

Imagine Victor being capable of rewind Peggy's execution without her knowledge. In such a case, Peggy would re-execute the protocol exactly as before,

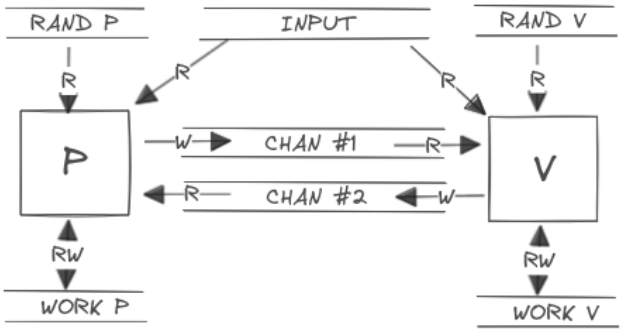

Figure 4: Prover and Verifier Interactive Turing Machines

even using the same random values as in the previous execution. From Peggy's perspective, these executions would appear indistinguishable. If under these conditions Victor is able to recover some witness or the incontrovertible evidence ($\varepsilon = 0$) about Peggy's knowledge then the protocol is sound. The rationale is that it is statistically impossible for a dishonest prover to disclose crucial information without possessing it.

It is important to note that the extractor is a theoretical construct, which should not be allowed to exist in the protocol's execution environment. For example, if the environment is the *real world*, it could be something like a time machine. In the *digital world*, it could be the capability to snapshot and restart the *prover* state at any point.

3.4 Interactive Turing Machines

Definition. *An **Interactive Turing Machine** (ITM) is a Turing machine with a read-only input tape, a read-write work tape, a read-only random tape, a read-only communication tape and a write-only communication tape.*

In an IP system, both the *prover* (P) and the *verifier* (V) are defined as ITM who share the same input tape, which generally contains the encoded assertion to be proven.

The read-only communication tape of one machine is defined to be the write-only communication tape of the other machine. This type of tape is used to exchange protocol messages.

During a protocol execution, the two machines take turns in being active. With each protocol step, the active ITM utilizes all its readable tapes as inputs for internal computations. It then writes the resulting output to the write-only communication tape (which corresponds to the read-only input of the other machine) and optionally updates its state stored within the work tape.

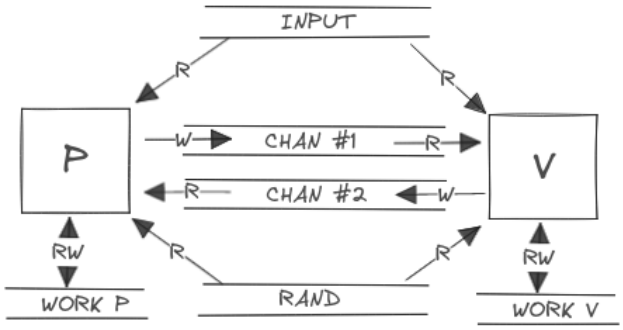

Figure 5: Arthur-Merlin protocol

3.5 Interactive Proof Computational Complexity Class

In this section, we give a quick look at the complexity class associated with IP systems. This topic is highly theoretical and falls somewhat outside the primary focus of this paper.

Definition. *The IP complexity class is the set of languages for which there exists an IP system whose proofs can be verified in polynomial time with respect to the length of the statement to prove.*

$$IP = \{L \mid L \text{ has an interactive proof } \}$$

This class can be further classified based on the IP system protocol characteristics. For instance, $IP[k]$ denotes the class of languages that can be decided by an interactive proof with k rounds.

At the beginning of the 90s, Lund, Fortnow, Karloff and Nisan [LFKN90] proved that $PH \subset IP$, indicating the significant relevance of interactive proofs as they contain the union of all complexity classes in *polynomial hierarchy* (PH), which includes NP and $co - NP$. Shortly later, Shamir [Sha92] proved that $IP = PSPACE$, which gave a complete characterization of the capabilities of interactive proofs.

The interaction between the prover and the verifier also leads to a novel and more efficient verification process, allowing the verifier to assess the proof validity without the need for the prover to share any witness (ZK proofs).

3.6 Arthur-Merlin Protocols

An **Arthur-Merlin** (AM) protocol, introduced by Babai [Bab85] in 1985, is an IP system with the additional constraint that the *prover* and the *verifier* share the same randomness source. In this context, Merlin is the *prover* and Arthur is the *verifier*.

The set of decision problems that can be decided in polynomial time by an AM protocol with k messages is called $AM[k]$. Babai proved that for all $k \geq 2$, $AM[k]$ is equivalent to $AM[2]$. This result is due to the fact that Merlin can observe Arthur randomness source during the whole protocol execution.

Goldwasser and Sipser [GS86] proved that for any language with an interactive proof protocol with private randomness (IP) also has an interactive proof with public randomness with two messages ($AM[2]$). In particular: $AM[k] \subset IP[k] \subset AM[k+2]$. And since, for $k \geq 2$, $AM[k+2] = AM[k] = AM[2]$ follows that $IP = AM[2]$.

Is worth anticipating that even though general IP systems with secret randomness source are not more powerful in terms of the range of languages they can prove, the secrecy of the randomness becomes crucial for proving statements without sharing any knowledge.

3.6.1 MA Protocols

The set of decision problems that can be verified in polynomial time using a single message AM protocol forms the MA set. MA protocols are very similar to traditional static proofs with the addition that the *prover* can use a public randomness source to construct the proof.

3.7 Examples

3.7.1 Tetrachromacy

Tetrachromacy is a condition enabling some individuals to perceive a broader spectrum of colors than the typical trichromat, who has three types of cone cells for color vision. Tetrachromats have an additional cone cell type, allowing them to see a wider range of colors.

In this scenario, Peggy claims to be tetrachromat and wants to prove it to Victor by showing that she's able to distinguish between two marbles who appear identical to Victor.

Protocol:

1. Peggy places the two marbles in from of Victor and turns her back.
2. Victor flips a coin and, based on the outcome, he may swap the position of the marbles.
3. Peggy, facing the marbles again, tells Victor whether their positions were swapped.
4. Victor accepts if Peggy's answer is correct.

The likelihood of Peggy falsely claiming to distinguish the marbles and being able to cheat (soundness error) is $\varepsilon = 1/2$.

Note that in this protocol a malicious verifier may replace one of the marbles with another, apparently equal, marble and thus use Peggy as an oracle to determine if this new marble is equal to the replaced one.

3.7.2 Quadratic Non-Residuosity

A number $y \in \mathbb{Z}_n^*$ is a quadratic residue if there exists an $x \in \mathbb{Z}_n^*$ such that $x^2 \equiv y \pmod{n}$. If no such x exists, y is a quadratic non-residue modulo n.

$$QR = \{y \mid y \in \mathbb{Z}_n^* \text{ and is a quadratic residue } \}$$
$$QNR = \{y \mid y \in \mathbb{Z}_n^* \text{ and is a quadratic non-residue } \}$$

It is considered to be a hard problem to tell if $y \in QR$ or $y \in QNR$ without knowing the factorization of y.

If Peggy wants to prove to Victor that $y \in QR$ or that $y \in QNR$ without sharing the factorization of y she requires an IP system. The following protocol, proves that $y \in QNR$ and is based on the fact that if $y \in QNR$ then $y \cdot k^2 \mod n \in QNR$ for any $k \in \mathbb{Z}_n^*$.

Protocol [GMR85]:

1. Victor selects a random $r \in \mathbb{Z}_n^*$ and flips a coin. If the coin shows *heads* he sets $t = r^2 \mod n$ else $t = y \cdot r^2 \mod n$. He sends t to Peggy.
2. Peggy, which has unrestricted computing power, finds if t is a quadratic residue and tells Victor what was his coin toss result.
3. Victor accepts if Peggy's answer is correct.

If $y \notin QNR$ then $y \in QR$ and thus, regardless of the coin toss result, $t \in QR$ as well. In this case Peggy has no way to recover the coin toss results and thus the soundness error $\varepsilon = 1/2$.

Note that if $y \in QNR$ then using this protocol a malicious verifier can use Peggy as an oracle to learn if an arbitrary number $k \in QR$ or not. As we'll see in the next sections we can prove both $y \in QR$ or $y \in QNR$ without sharing any information.

4 Zero-Knowledge Proofs

Now that we defined the interactive class of proof system is finally time to better discuss the core topic of this paper: *the quantity of knowledge required to validate a statement.*

The concept of Zero-Knowledge Proofs (ZKP) was first rigorously defined in the 80s by Goldwasser, Micali and Rackoff [GMR85]. Before their work, most of the effort on IP systems area focused on the *soundness* of the protocols. That is, the sole conceived weakness was a malicious prover attempting to trick the

verifier into approving a false statement. What Goldwasser, Micali and Rackoff did was to turn the problem into: *what if instead the verifier is malicious?*.

The specific concern they raised was about **information leakage**. Concretely, how much extra information is Victor going to learn during the execution of the protocol beyond the mere fact that the statement is true.

Definition. *A proof system for a language L is **zero-knowledge** if, for all $x \in L$, Peggy reveals to Victor a single bit of information: the fact that $x \in L$.*

At first glance, one might assume that the definition forbids sharing only the specific details regarding the how and why the statement being proven is true. However, the criteria in ZKP are much more stringent as it encompasses any piece of information that Victor cannot independently compute, including facts that are not related to the proof.

The definition holds true even when Victor is not honest, bounded by his polynomial-time capabilities.

Key attributes of a ZKP system (P, V) for a language L:

- **Completeness**: If $x \in L$ then $V(x, \pi) = 1$ with high probability.
- **Soundness**: If $x \notin L$ then $V(x, \pi) = 1$ with negligible probability.
- **Efficiency**: The total computation time of $V(x, \pi)$ and total communication in (P, V) is polynomial with respect to length of x.
- **Zero-knowledgeness**: The proof does not reveal any additional information other than the fact that the statement is true.

The proofs of completeness and soundness follow methodologies analogous to those employed in probabilistic interactive proof systems.

4.1 Proving Zero-Knowledgeness

The proof is based on the construction of a **simulator** and the basic idea is similar to the one used to prove *soundness* with the *extractor*.

A *simulator* is a hypothetical tool which allows Peggy to convince Victor that a statement is true, and thus about the knowledge of some key information, when she doesn't possess any knowledge.

The intuition here is that if Peggy can consistently convince Victor of the statement's truth without possessing any knowledge, then the protocol itself can't reveal any information to Victor. In essence, Victor cannot discern whether Peggy truly holds any knowledge based on the protocol's execution.

Similarly to the *extractor*, the implementation of the *simulator* is based on rewinding Victor's execution in order to gain some advantage without him noticing. From Victor's perspective, the outputs produced by the *simulator* are statistically indistinguishable from any genuine protocol execution.

4.2 Probability Distributions Distinguishability

Distinguishability refers to the ability of a polynomially computationally bounded *Turing machine* to distinguish between two random variables.

More generically, consider two families of random variables, $\{P(x)\}$ and $\{S(x)\}$, defined over a language L. Indistinguishability of these two families is about the capacity of a judge to tell if a given sample originated from $P(x)$ or $S(x)$ for some $x \in L$.

The judge decision-making process is influenced by two factors:

- The size of the sample.
- The time available to decide.

Based on these parameters, $P(x)$ and $S(x)$ can be classified as:

- *Equal*: if the decision is random regardless of time and sample size.
- *Statistically indistinguishable*: if the decision becomes random when given infinite time and samples with polynomial size with respect to $|x|$.
- *Computationally indistinguishable*: if the decision becomes random when both time and samples size are polynomially bounded by $|x|$.

In practice, given the verifier (the judge) polynomial bounds, practical ZK proofs are mostly concerned with computational indistinguishability.

In ZKP protocols, indistinguishability is relevant to ensure that during the protocol execution no information is leaked by the prover. The property is carefully evaluated when analyzing the protocol's *zero-knowledgeness* as the verifier should not be able to distinguish between the prover and the simulator messages.

4.3 Additional Considerations

Any ZK protocol must be run in an environment where the construction of either an *extractor* or a *simulator* is infeasible.

If Victor is able to construct an *extractor* then it will be able to extract knowledge from the proof, and thus undermine *zero-knowledgeness*. Conversely, if Peggy is able to construct a *simulator* she will be able to forge valid proofs without any knowledge, and thus undermine *soundness*.

Since re-playing a protocol compromises its fundamental properties, using a recording of the protocol execution to convince a third party about the validity of a proof doesn't have any value. An observer has no way to tell if the recorded execution is genuine or if the protocol steps were manipulated, which in practice requires rewinding the execution.

Follows that in the context of a ZKP system the prover is able to convince only the verifier who is actively engaged in the execution of the protocol.

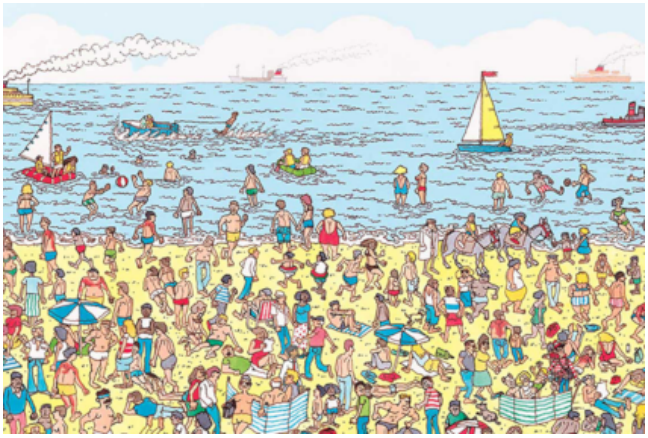

Figure 6: A typical "Where is Waldo" illustration

5 Simple ZK Protocols

While *real-world* ZK proofs often rely on complex mathematical structures and cryptographic techniques, there are intuitive examples that effectively convey the same core principles.

Except where specifically indicated, the discussed protocols are both sound and zero-knowledge as it is feasible (and often straightforward) to construct both an *extractor* and a *simulator* for them.

5.1 Where is Waldo

"Where is Waldo" is a famous kid's puzzle where, given a very detailed illustration with many different characters the goal is to find Waldo, the main character.

Peggy asserts she knows where Waldo is and wants to convince Victor without revealing any additional information.

Protocol [NNR98]:

1. Peggy covers the illustration with a large sheet of paper (bigger that the one with the illustration) which has a little hole in the center, positioned exactly over Waldo's face.
2. Victor is convinced if he sees Waldo through the hole.

Even though extremely simple, this protocol doesn't adequately address soundness concerns. How can Victor be sure the covered illustration is the original one?

The following extended protocol is designed to be sound:

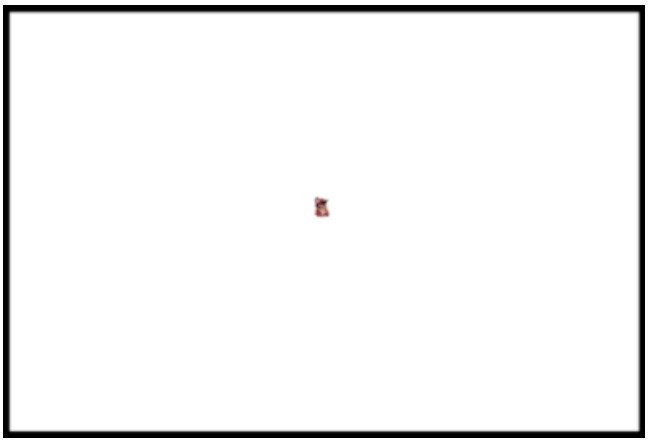

Figure 7: Waldo's face is shown through the hole in the overlay sheet

1. Peggy covers the illustration with a sheet of paper large enough to conceal the relative position of the illustration and with a hole hole positioned over Waldo's face. Then she covers this last with another sheet of equal size but without any hole.
2. Victor flips a coin, and depending on the outcome, asks Peggy to either remove both layers to reveal the illustration or just the top layer to see Waldo's face through the hole.
3. Peggy complies with the challenge.
4. Victor accepts or rejects based on the evidence.

Peggy committed the illustration position in the initial step so she can't change it. Since she doesn't know if Victor will ask her to reveal the illustration or the face of Waldo, she must be prepared to satisfy either of these potential challenges. A similar principle will be used in most of the subsequent protocols.

In one run, soundness error is $\varepsilon = 1/2$, which means that Peggy has a 50% chance to cheat.

5.2 Ali Baba Cave

The story is about Ali Baba, a guy who knows the magic spell to open a secret door in a cave. The cave has a single entrance and splits into two paths, which reconnect at the end through the magic door. Ali Baba wants to prove his knowledge of the spell without revealing it.

Protocol [QQQ$^+$89]:

1. Ali Baba enters the cave and randomly takes one of the two paths, while Victor waits outside.

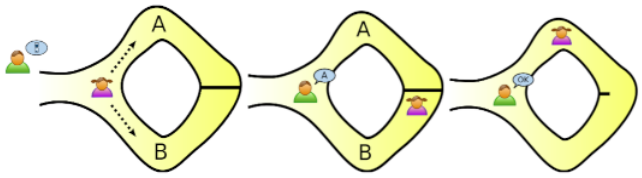

Figure 8: Ali Baba Cave ZK proof illustration

2. Victor enters the cave, goes to the bisection, flips a coin, and based on the outcome, asks Ali Baba to come out from a specific path.
3. Ali Baba complies with the Victor's request, using the magic door if necessary.
4. Victor accepts or rejects the proof if he sees Peggy coming out from the expected path.

For one run, soundness error is $\varepsilon = 1/2$.

Is worth noting that if both parties enter the cave together, Victor can observe Ali Baba taking one path and exiting from another, confirming his knowledge in one single run ($\varepsilon = 0$). However, this approach is not consistent with the strict definition of ZK proof, which should convince only Victor. The possibility of Victor recording the event would extend the proof's validity verification to anyone.

6 Intermediate ZK Proofs

6.1 Sudoku

Given a Sudoku puzzle instance, Peggy wants to convince Victor that she knows the solution without revealing it.

Protocol [GNPR09]: [2]

1. Peggy places three cards on each cell of the Sudoku grid. For pre-filled cells she places three cards with the assigned value, face-up. For other cells, she places the cards according to the solution, face-down.
2. Victor randomly selects one of the three cards from each cell across every row, column and subgrid, creating 27 groups of cards.
3. Peggy shuffles each group independently, and gives the shuffled groups to Victor.
4. Victor checks that each group contains all numbers from 1 to 9.

[2] Sudoku protocol demo: `https://www.wisdom.weizmann.ac.il/~naor/PAPERS/SUDOKU_DEMO`

Soundness error for this protocol is $\varepsilon = 1/9$ (refer to [GNPR09, pp. 9] for a proof).

6.2 Graph 3-Coloring

The generic *graph N-coloring* problem is about deciding if a given graph vertices can be colored with N different colors such that no two adjacent vertices share the same color.

Given a graph, Peggy wants to convince Victor that she knows the solution to the 3-coloring problem for it.

Protocol [GMW91]: [3]

1. Peggy draws the graph, assigns to the solution the colors randomly and covers each vertex with a hat.
2. Victor randomly selects two adjacent vertices and asks Peggy to reveal their colors.
3. Peggy reveals the colors of the selected vertices.
4. Victor accepts the proof if the vertices have different colors.

Let E be the number of edges in the graph, since Victor checks only one out of the E possible ones, the soundness error probability is $\varepsilon = (E-1)/E$. Even though this value can be reduced arbitrarily by repeating the protocol, the error approaches 1 quite fast with the number of edges and thus can be very expensive to be performed in practice.

For example, if $E = 1000$ and the *verifier* wants $\varepsilon < 0.1$ then the protocol should be iterated for k rounds where $(999/1000)^k < 1/10$ and thus $k > 2301$.

6.3 Proofs for all NP

While the protocols provided in this section might seem limited in their direct application, both are ZK proofs of knowledge of solutions for NP**-complete** problems.

The implication is profound: **any** problem in NP class can theoretically be converted into an instance of the 3-coloring problem, and thus a ZKP exists for every problem in NP.

The typical method for such transformation begins with reformulating the NP problem into a *Boolean circuit*. This circuit is designed to generate a *true* output if and only if the input represents a correct solution to the original NP problem. Subsequently, this Boolean circuit is converted into a graph. The construction of this graph ensures that finding a valid 3-coloring correlates directly with solving the original NP problem.

[3] Graph 3-coloring protocol demo: `http://web.mit.edu/~ezyang/Public/graph/svg.html`

While theoretically feasible, this approach is often not practical. The transformation process can be computationally expensive, not to mention the high soundness error of the graph 3-coloring protocol. Therefore, in practice, where possible specialized and more efficient approaches are employed for specific NP problems.

7 More Advanced ZK Protocols

7.1 Graph Isomorphism

Two graphs G_0 and G_1 are isomorphic if there exists a bijective mapping $f : G_0 \to G_1$ such that for any edge (v, w) in G_0 there is a corresponding edge $(f(v), f(w))$ in G_1.

The problem about determining if two graphs are isomorphic is known to be in NP, but at the current state of knowledge, not NP-complete.

Peggy wants to prove to Victor that G_0 and G_1 are isomorphic without revealing the specific mapping f such that $G_1 = f(G_0)$. That is, that the (G_0, G_1) couple belongs to the language:

$$GI = \{(G_0, G_1) \mid G_0 \text{ and } G_1 \text{ are isomorphic } \}$$

Protocol [GMW91]:

1. Peggy selects a random bit $p \in \{0, 1\}$, a random permutation π_x and sends $H = \pi_x(G_p)$ (permutation of G_0 or G_1) to Victor.
2. Victor selects a random bit $v \in \{0, 1\}$ and sends it to Peggy.
3. Peggy sends the permutation π_y such that $\pi_y(H) = G_v$.
4. Victor checks if π_y gives the expected result.

The protocol has soundness error $\varepsilon = 1/2$.

1. *Soundness* proof: Victor constructs an *extractor* by sending $v = 0$, to get π_{y_0} which maps H to G_0. Then, he re-execute the protocol from step 2 (challenge) by sending $v = 1$, to get π_{y_1} which maps H to G_1. He recovers the isomorphism f as $\pi = \pi_{y_1} \cdot \pi_{y_0}$.
2. *Zero-knowledgeness* proof: Peggy constructs a *simulator* which sends $H = \pi_x(G_0)$. If Victor sends $v = 1$ then Peggy re-executes the protocol by sending $H = \pi_x(G_1)$. She responds to the challenge with π_x.

7.2 Graph Non-Isomorphism

The *Graph Non-Isomorphism* problem is the complement of the *Graph Isomorphism* one, thus falls in the $co - NP$ complexity class.

The problem is about checking if a pair (G_0, G_1) belongs to the language:

$$GNI = \{(G_0, G_1) \mid G_0 \text{ and } G_1 \text{ are not isomorphic } \}$$

This problem is of particular interest since, unlike GI language which, if we ignore the ZK, can be solved using a traditional proof system by sharing the mapping f from G_0 to G_1, the GNI problem based on our current knowledge can't be solved without an IP system.

Protocol [GMW91]:

1. Victor selects a random bit $a \in \{0, 1\}$, a random permutation π and sends $H = \pi(G_a)$ to Peggy.
2. Peggy using its unbounded computational power determines whether H is a permutation of G_0 or G_1 (can't be of both as they are not isomorphic). Thus sends to Victor the bit b
3. Victor accepts the proof if $a = b$.

The protocol has soundness error $\varepsilon = 1/2$.

Note that this protocol doesn't make use of a commitment and indeed its ZK property is a bit flawed. Victor can send any random H and use Peggy as an oracle to gain knowledge if H is a permutation of one of the two graphs. The way to fix this is to require first Victor to prove to Peggy that he knows an isomorphism between his query graph H and one of the two input graphs. This is done using a parallel version of the GI proof protocol [GMW91, section 2.3].

7.3 Quadratic Residue

A *quadratic residue* modulo n is an integer x such that there exists an integer w where $w^2 \equiv x \pmod{n}$.

Peggy wants to prove that $x \in \mathbb{Z}_n^*$ is an element of the language:

$$QR = \{x \mid x \text{ is a quadratic residue}\}$$

All the operations are assumend to be reduced modulo n.

Protocol [GMR85]:

1. Peggy chooses a random $r \in \mathbb{Z}_n^*$ and sends $y = r^2$ to Victor.
2. Victor tosses a coin, chooses $b \in \{0, 1\}$ and sends it to Peggy.
3. If $b = 0$ then Peggy sends $z = r$ else she sends $z = r \cdot w$ to Victor.
4. Victor accepts if:
   (a) $b = 0$ and $z^2 = y$, or
   (b) $b = 1$ and $z^2 = x \cdot y$

The protocol the soundness error is $\varepsilon = 1/2$.

1. *Soundness* proof. Victor constructs an *extractor* which rewinds the protocol execution to send to Peggy both 1 and 0 for the same run. It will thus acquire both r and $r \cdot w$ which allows recovering $w = r^{-1} \cdot (r \cdot w)$.
2. *Zero-knowledgeness* proof. Peggy constructs a *simulator* such that if Victor's challenge is 1, then she rewinds the protocol execution to commit $y = r^2 \cdot x^{-1}$ and to send as the challenge response $z = r$. In this way $x \cdot y = x \cdot (r^2 \cdot x^{-1}) = r^2 = z^2$ satisfies Victor's check.

As for GI, we can prove the complement of the QR language, known as QNR. You can find more information for this protocol in the [GMR89, paragraph 6] paper.

8 Cryptographic ZK Protocols

We finally reached the section where we can apply what we've seen so far to analyze some widespread cryptographic protocols.

The context is the realm of public key cryptography that relies on the hardness of solving the discrete logarithm problem in a cyclic group.

8.1 Schnorr's Protocol

Given a cyclic group G with a generator g of prime order p, Peggy wants to prove to Victor her knowledge of the discrete logarithm $x \in \mathbb{Z}_p^*$ for some group element $y = g^x \in G$ without revealing any additional information.

Protocol [Sch91]:

1. Peggy selects a random $k \in \mathbb{Z}_p^*$ and sends $r = g^k$ to Victor.
2. Victor selects a random $c \in \mathbb{Z}_p^*$ and sends it to Peggy.
3. Peggy computes $s = x \cdot c + k \mod p$ and sends it to Victor.
4. Victor accepts if $g^s = y^c \cdot r$.

Security considerations:

- Peggy can't cheat because constructing a valid s requires knowledge of x. The only scenario where she might successfully cheat is if she's able to predict the challenge c before committing to r. In such a case, she can construct $r = g^s \cdot y^{-c} \mod p$ for any chosen s.
- Victor can't cheat because to extract the value of x from s he must compute $x = (s - k) \cdot c^{-1}$. However, this requires him to solve the discrete logarithm problem for r in order to compute k.
- The protocol has soundness error $\varepsilon = 1/|\mathbb{Z}_p^*|$, thus given a reasonably big prime p a single protocol run is sufficient.

8.1.1 Soundness Proof

The *extractor* rewinds Peggy's execution to the challenge step after she already responded to the challenge c_1 with s_1. By presenting a different challenge c_2 the extractor can induce Peggy to generate a different s_2 using the same k:

$$\begin{aligned} s_1 &= x \cdot c_1 + k \mod p \\ s_2 &= x \cdot c_2 + k \mod p \\ s_1 - s_2 &= x \cdot (c_1 - c_2) \mod p \\ x &= (s_1 - s_2) \cdot (c_1 - c_2)^{-1} \mod p \end{aligned}$$

The soundness proof highlights a crucial prerequisite for the protocol. Peggy must **never reuse the same value for** k in two different runs of the protocol. Reusing k easily leads to the disclosure of her secret.

8.1.2 Zero-Knowledgeness Proof

The *simulator* rewinds Victor's execution before the commitment phase after he shared the challenge c. She can now convince him without knowing the secret by committing to a value r computed as:

$$r = g^s \cdot y^{-c} \mod p$$

This convinces Victor, for any arbitrary s, as the equation $g^s = y^c \cdot r$ holds true.

Is worth noting that the *zero-knowledgeness* proof assumes Victor to be honest ($HVZK$[4]), which in this case means that c is not chosen in function of r. If instead $c = f(r)$ then our definition of *simulator* is ineffective.

Although certain ZKP systems can prove *zero-knowledgeness* property even in the presence of a malicious verifier, this minor theoretical limitation in Schnorr's protocol is not a concern for practical applications.

8.2 Non-Interactive Schnorr's Protocol

Our discussion so far has emphasized the importance of interactivity to prove certain problems. In the *real world*, this remains predominantly true. However, there is an *imaginary world* where this limitation can be circumvented.

Converting Schnorr's protocol into a **non-interactive proof** initially seems infeasible due to its fundamental reliance on the *verifier*'s randomly chosen challenge. Yet, this is not true in the *imaginary world*.

In the 80s, Fiat and Shamir [FS87] introduced a technique, known as the **Fiat-Shamir heuristic**, to transform an interactive protocol into a non-interactive proof within an imaginary environment known as the **random oracle model** (ROM). Within this model we can replace the *verifier*'s random challenge with

[4]Honest Verifier Zero-Knowledge

the output of a cryptographically secure hash function H seeded by both the problem input and the prover's commitment.

Protocol [Sch91]:

1. Peggy picks a random $k \in \mathbb{Z}_p^*$ and computes $r = g^k$.
2. Peggy computes the challenge $c = H(r) \in \mathbb{Z}_p^*$.
3. Peggy computes $s = x \cdot c + k \mod p$.

An arbitrary verifier accepts the proof if $g^s = y^c \cdot r$.

The implications of using the *Fiat-Shamir heuristic* are significant, fundamentally altering the assumptions used to prove *soundness* and *zero-knowledgeness* of ZK protocols.

Of course, since we are already working in a hypothetical environment we can also imagine to work with a programmable random oracle to artificially restore the properties:

- Restoring *soundness*: in a standard settings, the *extractor* depends on receiving two different responses s_1 and s_2 for the same commitment r, by submitting different challenges. However, this approach doesn't work when using the *Fiat-Shamir heuristic* as $c = H(r)$. In the ROM, the proof holds if the extractor programs the oracle to return a different value for c in the two different extractor's runs.
- Restoring *zero-knowledgeness*: similarly, the *simulator* depends on predicting the challenge c before generating the commitment r. This doesn't work with *Fiat-Shamir heuristic* as $c = H(r)$ and r should be generated as $r = g^s \cdot y^{-c} \mod p$. In the ROM, the proof holds if the simulator programs the oracle to return a fixed value for the required commitment r.

While the concept of a programmable oracle aids in validating the protocol within the ROM, in practical applications, the random oracle is typically realized through a non-programmable, cryptographically secure hash function.

To summarize, although the ROM assumption is controversal[5], it has been effectively used to demonstrate the security of various real-world cryptographic primitives. The essential practical security requirement is that the prover must not be able to predict or control the hash output.

[5] https://blog.cryptographyengineering.com/2011/09/29/what-is-random-oracle-model-and-why-3

8.2.1 Schnorr Signature

The non-interactive Schnorr's protocol can be easily transformed into a signature scheme by binding a message m to the challenge c:

$$c = H(r, m)$$

9 Conclusions

The evolution from *classical* proofs to *zero-knowledge* proofs highlights a significant shift in problem-solving techniques, illustrating how complex solutions can be verified without sharing sensitive information.

Tracing back to the groundbreaking work of Goldwasser, Micali, and Rackoff in the 1980s, ZK proofs have transcended their initial theoretical boundaries, emerging as important tools in the verification of information in today's technology-centric world, finding applications spanning from blockchain technology to secure cloud computing.

In recent years, ZK proofs have further evolved from being simple proofs of knowledge to complex proofs of arbitrary computation, as exemplified by technologies like *zk-SNARK*s (zero-knowledge Succinct Non-Interactive Argument of Knowledge) and *zk-STARK*s (zero-knowledge Scalable Transparent Argument of Knowledge).

Looking forward, the potential of ZK proofs seems boundless. While academic research is continually pushing the frontiers with a continuous stream of innovation and proposals[6], open-industry initiatives like ZKProof[7] aim to develop a shared set of standards for ZK protocols to ensure interoperability and security. These collective efforts are vital in transitioning ZK proofs from niche cryptographic tools to more widely adopted technologies.

References

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, New York, NY, USA, 1985. Association for Computing Machinery.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology*, CRYPTO '86, pages 186–194, Berlin, Heidelberg, 1987. IACR, Springer-Verlag.

[6] `https://zkp.science/`
[7] `https://zkproof.org`

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[GNPR09] Ronen Gradwohl, Moni Naor, Benny Pinkas, and Guy N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles. *Theory or Computing Systems*, 44(2):245–268, feb 2009.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 59–68, New York, NY, USA, 1986. Association for Computing Machinery.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, volume 1, pages 2–10. IEEE, 1990.

[NNR98] Moni Naor, Yael Naor, and Omer Reingold. Applied kid cryptography or how to convince your children you are not cheating. In *Proceedings of Eurocrypt '94*. IACR, 1998.

[QQQ+89] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 628–631, Berlin, Heidelberg, 1989. IACR, Springer-Verlag.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sha92] Adi Shamir. Ip = pspace. *Journal of the ACM*, 39(4):869–877, oct 1992.