# Lecture 2

Gidon Rosalki
2026-04-29

## 1. Introduction

We have discussed ZK, and the following:

1. Interaction is **required** (as in, it must be an interactive proof)
2. It is impossible to do perfect ZK for every language, since this would mean that $P = NP$
3. It is possible to do computational ZK for every language in NP
4. There are in fact methods to do non-interactive ZK (in direct contradiction to the first point)

## 2. Non-Interactive ZK

We can make use of the existence of cryptographic hash functions, such as SHA256, or SHA3. Consider the QR proof example, between $P$ and $V$. To make this non interactive, we will have the prover send the sequence $(r^2, b, z_b)$ an incredibly large number of times, where with cryptographic hash function $\mathcal{O}$:

$$b \leftarrow \mathcal{O}(r^2)$$

$$z_b = \begin{cases} r & \text{if } \ b = 0 \\ xr & \text{if } \ b = 1 \end{cases}$$

The idea in the interactive protocol is that one must commit to $r^2$ before discovering $b$. The same is done here, thanks to the cryptographic hash function. Additionally, the verifier may feel confident in the proof, since they have received an incredibly large number of examples, as required earlier. This is a method to create non-interactive ZKP, which is the Fiat-Shamir method (yes, both Israeli).

## 3. ZK For All NP

We showed two languages with perfect ZK proofs. Can we show this for all NP languages?
*Theorem* [Fortnow'89, Aiello-Hastad'87] No, unless bizarre stuff happens in complexity theory (technically: the polynomial hierarchy collapses.)

However, we will instead show:
*Theorem* [Goldreich-Micali-Wigderson'87] Assuming one-way functions exist, all of NP has computational zero-knowledge proofs.
This tells us that everything can be proven (in the sense of Euclid) in zero knowledge.

Let us consider the 3-Colouring problem. This problem is NP-Complete, meaning that *every* other problem in NP may be reduced to it.

### 3.1. Commitment Schemes

These are based off one way functions / PRGs. Here we have a sender, and receiver, where the sender commits to a bit $b$, which has been cryptographically obscured, and may then send a key to open the cryptographic box to reveal $b$ to the receiver. Commitment schemes should enable

- **Hiding**: The locked box should completely hide $b$
- **Binding**: The sender should not be able to open to $1 - b$

We also have the following features:

1. **Completeness**: R always accepts in an honest execution.
2. **Computational Hiding**: For every possible malicious PPT $R^*$:

$$\text{view}_{R^*}(S(0), R^*) \underset{c}{\approx} \text{view}_{R^*}(S(1), R^*)$$

3. **Perfect Binding**: For every possible malicious $S^*$, let COM be the receiver's output in an execution of $(S^*, R)$. There is no pair of decommitments $(\text{DEC}_0, \text{DEC}_1)$ such that $R$ accepts both of $(\text{COM}, 0, \text{DEC}_0)$ and $(\text{COM}, 1, \text{DEC}_1)$

## 3.2. ZK For 3COL

$P$ comes up with a random permutation of the colours: $\rho : V \to \{R, G, B\}$ which it commits to $V$. $V$ then requests a random edge $(i, j)$, and $P$ responds with the opening of $\rho(i), \rho(j)$. The verifier then checks the openings, that the results are all in $\{R, G, B\}$, and that the two nodes are distinct from each other. Behold! Using commitments, we have a ZKP for 3COL.

**Completeness**: Exercise for the reader (trivial)

**Soundness**: If the graph is not 3COL, in every 3-colouring (to which $P$ commits), there is some edge whose end-points have the same colour. $V$ will catch this, and reject with probability $\geq \frac{1}{|E|}$. We can then repeat $|E| \cdot \lambda$ times to get the verifier to accept with probability

$$\leq \left(1 - \frac{1}{|E|}\right)^{|E| \cdot \lambda} \leq 2^{-\lambda}$$

**Zero Knowledge**: We will construct the simulator $S$ as follows:

1. First pick a random edge $(i^*, j^*)$. Colour those vertices with different, random colours, and all other vertices red.
2. Feed the commitments of the colours to $V^*$, and get edge $(i, j)$
3. If $(i, j) \neq (i^*, j^*)$, go back and repeat
4. If $(i, j) = (i^*, j^*)$, output the commitments and opening $r_i, r_j$ as the simulated transcript

*Theorem* Assuming the commitment is hiding, $S$ runs in polynomial time
*Proof* Trivial

*Theorem* When $S$ outputs a view, it is computationally indistinguishable from the view of $V^*$ in a real execution *Proof* Let us construct Hybrid 0, and Hybrid 1 from the above simulator, as follows. Hybrid 0 is just the above simulator, but Hybrid 1 changes the first step to "permute a legal colouring, and colour all vertices correctly". We will now claim that these 2 hybrids are computationally indistinguishable, assuming that the commitment scheme is computationally hiding.
This proof is by contradiction. We show a reduction that breaks the hiding property of the commitment scheme, assuming there is a distinguisher between hybrids 0 and 1. Hybrid 1 is not a simulator, and the real view of $V^*$ is Hybrid 2, where we

1. Permute a legal colouring, and colour all vertices correctly
2. Feed the commitments of the colours to $V^*$, and get edge $(i, j)$
3. Output the commitments and opening $r_i, r_j$ as the simulated transcript

We will claim that hybrids 1 and 2 are identical. This is true since 1 samples from the same distribution as 2, and with probability $1 - \frac{1}{|E|}$ decides to throw it away, and re-sample.

Put it together and what do you get? The 3COL problem is ZK.

Some examples of NP assertions:

- My public key is well formed
- Encrypted bitcoin: "I have enough money to pay you"
- Running programs on encrypted inputs: Given Enc $(x)$, and $y$, prove that $y = \text{PROG}(x)$
    - This s more generally a tool to enforce honest behaviour, without revealing information

A real world use was that a few months ago, Google published a ZKP that they have created an algorithmic optimisation to Shor's algorithm (quantum algorithm for finding prime factors in polynomial time) that instead of requiring hundreds of thousands of qbits, only requires 10s of thousands.

# 4. Proofs of Knowledge

Until now, we have had a prover, and a verifier, where the prover demonstrates to the verifier that some value $x$ is in the language $L$, without revealing the method to demonstrate this $w$. This is effectively the boolean $x \in L$. What if we do not want to simply prove that $x \in L$, but also that $P$ knows $w$? This is a *proof of knowledge*: $P$ knows $w$ s.t. $(x, w) \in R_L$.

How do we formalise this? For example, we have shown how to prove that there exist $p, q : pq = N$, but how do we prove that we know $p$ and $q$? To do this, we will introduce the **Extractor**. To formalise this properly, we will say that we want to prove knowledge of the Discrete Logarithm $y = g^x \bmod p$. For a generator $g$, there is always an $x$, so the original ZK is not helpful, but here we want to prove that we *know* the value of $x$. Let us do this as follows:

$$\begin{aligned} p &= 2q + 1 \\ y &= g^x \bmod p \\ \longrightarrow z &= g^r \bmod p \\ \longleftarrow c &\leftarrow \{0, 1\} \\ \longrightarrow s &= r + cx \bmod q \\ (g^{r+cx} &= g^r \cdot (g^x)^c) \end{aligned}$$

The verifier then accepts **if and only if** $g^s = z \cdot y^c$. The extractor does not really exist. We are imagining it for the sake of the proof. The above protocol takes place between Alice, and the Extractor. After we get $s = r + cx \bmod q$, we then return to after we received $g^r$, and request for $1 - c$. Once it finishes, thanks to two equations with two unknowns:

$$\begin{aligned} s &= r + cx \\ s &= r + (1 - c)x \end{aligned}$$

The Extractor may compute $x$.

If we consider this, we state to $P$ that $y = g^x \bmod p$.

$$\begin{aligned} y &= g^x \bmod p \\ \longrightarrow z &= g^r \bmod p \text{ Extractor runs } P^* \text{ to get a } z \\ \longleftarrow c &\leftarrow \{0, 1\} \text{ Runs with } c = 0 \text{ for } s_0 \text{ rewinds, and reruns with } c = 1 \text{ for } s_1 \\ \longrightarrow s_i &= r + c_i x \bmod q \\ g^{s_0} = z \wedge g^{s_1} &= zy \text{ with probability } \frac{1}{p(n)} \\ g^{s_1 - s_0} &= y \end{aligned}$$

So $s_1 - s_0$ is the discrete log of $y$. Note that this game is purely theoretical, and would never be run since it leaks the secret $x$. We just need to demonstrate that given the setup, such a game is theoretically possible, since then we have demonstrated a proof of knowledge.

*Theorem* [Goldreich-Micali-Wigderson'87] Assuming one way functions exist, all of NP has computational zero knowledge proof of knowledge.

## 4.1. Reducing the Soundness Error

The 3COL protocol has a large soundness error of $1 - \frac{1}{|E|}$ (this is the probability that $V$ accepts even though $G \notin$ 3COL).

*Theorem* Sequential Repetition reduces soundness error for interactive proofs (and preserves the ZK property).

This has the problem of requiring lots of rounds.

*Theorem* Parallel Repetition reduces soundness error for interactive proofs It is also honest-verifier ZK.

Problem with parallel repetition is that there may not be independence between the commitments, which were sent at the same time.

*Theorem* [Goldreich-Krawczyk'90] There exist ZK proofs whose parallel repetition is NOT (malicious verifier) zero knowledge.
This is unfortunate, since it means that when using parallel repetition, there are ZK protocols that are not ZK for parallel repetition. However, this counterexample is quite contrived. What about for the "natural protocols", such as for the 3-Colouring protocol from before?

*Theorem* [Holmgren-Lombardi-Rothblum'21] Parallel Repetition of the (Goldreich-Micali-Wigderson) 3COL protocol is not zero-knowledge.
In summary, shit. Our example earlier is not in fact ZK. However, there is another protocol, that assumes discrete logarithms are hard, that is.
*Theorem* [Goldreich-Kahan'95] There is a constant-round ZK proof system for 3COL (with exponentially small soundness error), assuming discrete logarithms are hard (more generally, assuming the existence of collision-resistant hash functions).