Cryptography and Security Protocols **IST-UL, 2024/2025**

Interactive proof systems and zero-knowledge proofs

Lecturer: João Ribeiro

Introduction

For most of the course so far we have been focusing on secure communication. But cryptography is about much more than this. In these notes we will rethink the notion of "proof".

These notes are heavily based on lecture notes of Noah Stephens-Davidowitz (see this and this) and Vinod Vaikuntanathan (see this and this). I claim no originality, but to add a tiny bit of breadth we work through interactive proofs for graph isomorphism instead of quadratic residuosity. You should go ahead and read those notes instead!

1 What is a proof?

We have all had plenty of contact with proofs throughout our education. In this course I have spent many hours "proving" theorems in front of you, aiming to convince you of their veracity. In the final exam you will be writing your own proofs, attempting to convince me of the veracity of some claim.

So, an initial attempt to define a proof could be something like this: a proof is something uttered by someone (the "prover") aiming to convince someone else (the "verifier") of the veracity of some statement. To formalize what kind of statements we care about proving in a general manner, we can imagine that there is some set $L \subseteq \{0,1\}^*$ (usually called a *language*), collecting all objects satisfying some relevant property), and the prover's goal is to convince the verifier that $x \in L$. This is especially interesting if we allow the prover more computational power than the verifier. We will consider the case where the prover is computationally unbounded and the verifier is computationally bounded (in particular, runs in polynomial time).

As we shall see, there is much more to "proofs" than just this. Exploring the notion of "proof" further leads to very nice mathematics and applications.

1.1 The complexity class NP

Our discussion of proofs begins in complexity theory. You have probably seen the complexity class NP before, which stands for "Non-Deterministic Polynomial Time" (it's not the best abbreviation...). The reason why we bring this up is that, intuitively, NP is the class of languages $L \subseteq \{0,1\}^*$ that admit short, deterministic, and efficiently-checkable proofs.

More precisely, we may define NP as the class of all languages $L \subseteq \{0,1\}^*$ for which there exists a deterministic polynomial-time algorithm $\mathcal{V}$ with the following properties:

1. If $x \in L$, then there exists $w \in \{0,1\}^*$ of length $|w| = O(|x|^C)$ for some constant $C > 0$ such that $\mathcal{V}(x,w) = 1$. We may see w as the witness/proof for the statement "$x \in L$".
2. If $x \notin L$, then for any w we have $\mathcal{V}(x,w) = 0$. In other words, if $x \notin L$ then a prover cannot convince the verifier of the opposite statement.

There are many natural languages in NP. First, all languages L such that $x \in L$ can be decided in polynomial time (the class P) are also in NP. These languages have trivial proofs (the empty string), because the computationally bounded verifier can just convince themselves that the statement is true. The most important open problem in all of theoretical computer science is whether P = NP. Most of cryptography requires at the very least that P ≠ NP (actually more than that).

Here are some examples of languages that are in NP but not known to be in P:

1. **Quadratic residuosity:** The language QR of all pairs of integers (N, x) with $x \in \mathbb{Z}_N^*$ such that x is a *quadratic residue modulo* N, i.e., there is $z \in \mathbb{Z}_N^*$ such that $x = z^2 \pmod N$.

   The proof that $(N, x) \in$ QR can be just its square root z such that $x = z^2 \pmod N$.

2. **Graph isomorphism:**[1] The language GI of all pairs of graphs[2] (G_0, G_1) that are *isomorphic*, i.e, there exists a permutation $\pi : V_0 \to V_1$, where V_i is the vertex set of G_i, such that u and v are adjacent in G_0 if and only if $\pi(u)$ and $\pi(v)$ are adjacent in G_1.

   The proof that $(G_0, G_1) \in$ GI can be just the isomorphism π.

3. **Graph 3-coloring:** The language 3-COL of all graphs G that admit a 3-coloring, i.e., for which there exists an assignment $\phi : V \to \{0,1,2\}$, where V is the vertex set of G, such that $\phi(u) \neq \phi(v)$ whenever u and v are adjacent (that is, no two adjacent vertices share the same color).

   The proof that $G \in$ 3-COL can be just the color assignment ϕ.

1.2 The complexity class IP

The class NP captures short proofs that are non-interactive and can be checked deterministically. What happens if we relax these assumptions, allowing for rounds of interaction between prover and verifier and the use of randomness (and a small probability of error) on the verifier's side?

Fix a language L and some $x \in \{0,1\}^*$. Consider an *interactive proof system*, where a computationally-unbounded prover $\mathcal{P}(x)$ and a PPT verifier $\mathcal{V}(x)$ sequentially trade messages. The prover's goal is to convince $\mathcal{V}$ that $x \in L$. At the end of the interaction $\mathcal{V}$ outputs a bit b, with $b = 1$ if $\mathcal{V}$ believes

[1] Surprisingly, even though we do not know whether GI ∈ P, we know that it is decidable in *quasi-polynomial time*. This is a fairly recent groundbreaking result of Babai [Bab16].

[2] We will assume that graphs are represented by their adjacency matrices, and the vertices of a graph with n vertices are labelled 1 to n.

that $x \in L$ and $b = 0$ otherwise. We denote the interactive proof system on input x by $\langle \mathcal{P}(x), \mathcal{V}(x) \rangle$, and the verifier's final output by $\mathsf{out}_{\mathcal{V}} \langle \mathcal{P}(x), \mathcal{V}(x) \rangle$.

Of course, for this interaction to be interesting we need some basic properties. First, if indeed $x \in L$, then the prover should be able to convince the verifier (*completeness*). Second, if $x \notin L$, then no (potentially dishonest) prover should be able to convince the verifier that $x \in L$, except with small probability (*soundness*). We denote by IP the class of languages that admit complete and sound interactive proofs. More precisely, we have the following definition.

Definition 1 *A language L is in IP if there exist a PPT verifier $\mathcal{V}$ and a computationally-unbounded prover $\mathcal{P}$ with the following properties:*

1. ***Perfect completeness:*** *For any $x \in L$,*

$$\Pr[\mathsf{out}_{\mathcal{V}} \langle \mathcal{P}(x), \mathcal{V}(x) \rangle = 1] = 1.$$

2. ***Soundness:*** *For any $x \notin L$ and every computationally-unbounded prover $\mathcal{P}^*$,*

$$\Pr[\mathsf{out}_{\mathcal{V}} \langle \mathcal{P}^*(x), \mathcal{V}(x) \rangle = 1] \leq 1/2.$$

It is natural to wonder why we choose to bound the soundness error by $1/2$. That seems pretty large. However, the choice of this upper bound does not matter, so long as it is at most $1 - 1/\mathrm{poly}(|x|)$. This is because then we can always reduce the soundness error (making it exponentially small in $|x|$, even) by sequentially repeating the interactive proof with fresh randomness a polynomial number of times. You will prove this in the problem set.

2 An interactive proof for graph *non*-isomorphism

Of course, all languages in NP have simple (even non-interactive proofs), and so $\mathsf{NP} \subseteq \mathsf{IP}$. In fact, IP is (under a reasonable conjecture) much more powerful than NP. A groundbreaking series of works showed that IP actually corresponds to the languages decidable by polynomial-*space* algorithms (for more on this, see the "Recommended reading" section later on).

Recall that graph isomorphism (the language GI) is in NP, and so is trivially in IP. But what about its complement, graph *non*-isomorphism? This is the language GNI of pairs of graphs (G_0, G_1) that are *not* isomorphic.

We do not know whether $\mathsf{GNI} \in \mathsf{NP}$. However, we will analyze a beautiful interactive proof for GNI, placing it in IP.

On input (G_0, G_1), the prover $\mathcal{P}$ and verifier $\mathcal{V}$ proceed as follows. Recall that for simplicity we assume that if both graphs have n vertices then their vertex sets are $[n] = \{1, \ldots, n\}$ (if the graphs do not have the same number of vertices then they are trivially non-isomorphic).

1. $\mathcal{V}$ samples $b \leftarrow \{0, 1\}$ and a permutation $\pi : [n] \to [n]$ uniformly at random. $\mathcal{V}$ sends $G = \pi(G_b)$ to $\mathcal{P}$, where $\pi(G_b)$ is the graph obtained by permuting the vertices of G_b according to π.

2. If G is isomorphic to G_0, then $\mathcal{P}$ sends $b' = 0$ to $\mathcal{V}$. Else, if G is isomorphic to G_1, then $\mathcal{P}$ sends $b' = 1$ to $\mathcal{V}$.
3. $\mathcal{V}$ outputs 1 if and only if $b' = b$.

We now analyze the completeness and soundness of this interactive proof:

1. **Perfect completeness:** Suppose that (G_0, G_1) are non-isomorphic. Note that $G = \pi(G_b)$ is isomorphic to G_b, and so is not isomorphic to G_{1-b}. Since the prover is computationally unbounded, it can verify whether G is isomorphic to G_0 or G_1 (by brute-force searching for a permutation) and respond with the appropriate b'. We are then guaranteed that $b' = b$ with probability 1.
2. **Soundness:** Suppose that (G_0, G_1) are isomorphic. Since π is a uniformly random permutation, the distribution of the graph $G = \pi(G_b)$ is independent of b. This means that no prover $\mathcal{P}^*$ can guess b with probability better than $1/2$.

3 A convoluted interactive proof for graph isomorphism?

The interactive proof for GNI above has the curious property that, although it clearly convinces the verifier with high probability that G_0 and G_1 are non-isomorphic, it does not tell the verifier *why* G_0 and G_1 are not isomorphic. In particular, it seems that after this interaction the verifier is unable to prove to someone else that G_0 and G_1 are not isomorphic, although he knows that this is true! We call this (for now intuitive) property *zero-knowledge*. Contrast this with the "NP proofs" for QR and GI, whose proofs explain why an input x is in the language and are definitely not zero-knowledge.

We will now see a similar protocol for graph isomorphism. On input (G_0, G_1) with n vertices each, the prover $\mathcal{P}$ and verifier $\mathcal{V}$ proceed as follows. Let $\pi^1 : [n] \to [n]$ be a permutation such that $G_0 = \pi^1(G_1)$ and π^0 the identity, so that $\pi^0(G_0) = G_0$.

1. $\mathcal{V}$ samples a uniformly random permutation $\pi^* : [n] \to [n]$ and sends $G = \pi^*(G_0)$ to $\mathcal{V}$.
2. $\mathcal{V}$ samples $b \leftarrow \{0, 1\}$ and sends b to $\mathcal{P}$.
3. $\mathcal{P}$ sends $\sigma = \pi^* \circ \pi^b$ to $\mathcal{V}$, where $\circ$ denotes composition of functions.
4. $\mathcal{V}$ outputs 1 if and only if $\sigma(G_b) = G$.

In other words, the protocol works as follows: $\mathcal{P}$ sends a random permutation G of G_0 to $\mathcal{V}$. If $\mathcal{P}$ receives $b = 0$, then $\mathcal{P}$ has to tell $\mathcal{V}$ how to go from G_0 to G. Otherwise, $\mathcal{P}$ has to tell $\mathcal{V}$ how to go from G_1 to G.

We can ask why there is even a need for $\mathcal{V}$ to make the random choice b. If $\mathcal{P}$ sends G, a permutation of G_0, to $\mathcal{V}$, then isn't $\mathcal{V}$ better off always sending $b = 1$ (i.e., asking for a permutation from G_1 to G)? When $b = 0$, $\mathcal{P}$ can always respond correctly even if G_0 and G_1 are not isomorphic! To see why we need this random choice b, note that the protocol must be sound against *dishonest provers*. For

example, if $\mathcal{V}$ always sends $b = 1$, then a dishonest prover $\mathcal{P}^*$ that deviates from the protocol and chooses G to be a random permutation of G_1 always wins even if G_0 and G_1 are not isomorphic.

Let's establish the completeness and soundness of this protocol.

Completeness. Suppose that G_0 and G_1 are isomorphic. If $b = 0$, then $\sigma(G_b) = \pi^*(G_0) = G$. If $b = 1$, then $\sigma(G_b) = \pi^* \circ \pi^1(G_1) = \pi^*(G_0) = G$.

Soundness. Suppose that G_0 and G_1 are isomorphic. Let $\mathcal{P}^*$ be an arbitrary (potentially dishonest) prover. We will show that $\mathsf{out}_{\mathcal{V}}\langle \mathcal{P}^*(G_0, G_1), \mathcal{V}(G_0, G_1)\rangle = 1$ with probability at most $1/2$. Suppose that $\mathcal{P}^*$ sends a graph G to $\mathcal{V}$ in the first step of the protocol. Note that $\mathcal{P}^*$ chooses G before seeing b, and that G is isomorphic to *at most one of* G_0 *and* G_1. There are three cases:

- G is isomorphic to G_0: if $\mathcal{V}$ sends $b = 1$ then there is no permutation σ such that $\sigma(G_1) = G$ (otherwise G_0 and G_1 would be isomorphic).
- G is isomorphic to G_1: if $\mathcal{V}$ sends $b = 0$ then there is no permutation σ such that $\sigma(G_0) = G$.
- G is not isomorphic to G_0 nor G_1: then there is no valid permutation σ for any choice of b.

In any of these cases $\mathcal{P}^*$ wins with probability at most $1/2$ (actually, probability 0 in the third case).

Is it zero-knowledge? We have not formally defined the zero-knowledge property yet, so we proceed for now with intuition only. Let's try to understand the "verifier's view".

Suppose that G_0 and G_1 are isomorphic. Note that during the protocol the verifier sees (G, b, σ). If $b = 0$, then $G = \sigma(G_0)$, with σ a uniformly random permutation. If $b = 1$, then $G = \sigma(G_1)$, again with σ a uniformly random permutation. So this means that we can perfectly "simulate" the verifier's view *based only on* (G_0, G_1) as follows: sample $b \leftarrow \{0, 1\}$ and a uniformly random permutation σ. Then, set $G = \sigma(G_b)$ and output (G, b, σ). This means that whatever $\mathcal{V}$ learned after running the protocol, he could have simulated on his own from things he already knew beforehand (the graphs G_0 and G_1)!

4 Formalizing the zero-knowledge property

Let's define the zero-knowledge property more formally. As hinted above, a protocol is zero-knowledge if "the verifier learns nothing from a protocol execution that he did not already know before". We formalize this through the *simulation paradigm*: the fact that the verifier learns nothing that he did not already know before is captured by the existence of an efficient algorithm (the *simulator*) that given only the initial input x perfectly simulates the verifier's full view in the protocol. In particular, in the special case of NP languages, the simulator does not know a witness w that allows one to efficiently check that $x \in L$. This means that a zero-knowledge proof reveals no

information about a witness for language membership that the verifier could not obtain efficiently on his own.

Also, note that a dishonest verifier may deviate from the protocol in an attempt to learn some information that he did not know before. Therefore, it also makes sense to consider the zero-knowledge with respect to *arbitrary verifiers* (in fact, this is closer to the right notion of zero-knowledge).

We proceed to formally define what we call *perfect zero-knowledge* with respect to honest verifiers and arbitrary (i.e., potentially dishonest) verifiers. We will use the notation $\mathsf{view}_{\mathcal{V}}\langle \mathcal{P}(x), \mathcal{V}(x)\rangle$ to denote the verifier's view in an interactive protocol – this consists of the verifier's input, the messages he receives and sends, and the randomness he uses during the protocol.

Definition 2 (Honest-verifier perfect zero-knowledge) *We say that an interactive protocol between a prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ for a language L is* honest-verifier perfectly zero-knowledge *if there exists a PPT simulator $\mathcal{S}$ such that for any $x \in L$ the random variables $\mathcal{S}(x)$ and $\mathsf{view}_{\mathcal{V}}\langle \mathcal{P}(x), \mathcal{V}(x)\rangle$ are identically distributed.*

Definition 3 (Perfect zero-knowledge) *We say that an interactive protocol between a prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ for a language L is* perfectly zero-knowledge *if* for any PPT verifier $\mathcal{V}^*$ *there exists a PPT simulator $\mathcal{S}$ such that for any $x \in L$ the random variables $\mathcal{S}(x)$ and $\mathsf{view}_{\mathcal{V}^*}\langle \mathcal{P}(x), \mathcal{V}^*(x)\rangle$ are identically distributed.*

The notion of zero-knowledge was first introduced by Goldwasser, Micali, and Rackoff [GMR89]. Rumor goes that this work was actually not initially well-received by the community! This work was rejected from several conferences before being accepted at STOC 1985.

5 Perfect zero-knowledge for graph isomorphism

We will now show that the interactive proof for GI defined above is perfectly zero-knowledge. We begin by showing that it is honest-verifier perfectly zero-knowledge.

5.1 Honest-verifier perfect zero-knowledge

Theorem 1 GI *has an honest-verifier perfectly zero-knowledge interactive proof.*

Proof: Recall that the honest verifier's view is (G, b, σ), where $b \leftarrow \{0,1\}$ and $G = \sigma(G_b)$ with σ a uniformly random permutation. We have actually already discussed the simulator $\mathcal{S}(G_0, G_1)$ above:

1. Sample $b \leftarrow \{0,1\}$.
2. Sample σ a uniformly random permutation and compute $G = \sigma(G_b)$.
3. Output (G, b, σ).

It is clear that the simulator output and the verifier's view are identically distributed, which establishes the desired property. ■

5.2 Perfect zero-knowledge

Theorem 2 GI *has a perfectly zero-knowledge interactive proof.*

Proof: To establish perfect zero-knowledge we need to design a simulator for an arbitrary PPT verifier $\mathcal{V}^*$ that can deviate from the protocol. Looking at the protocol, the only choice that $\mathcal{V}^*$ can make is about the value of b. Our previous simulator may not work here, because $\mathcal{V}^*$ may choose b adversarially as a function of the graph G sent by $\mathcal{P}$ in the first message.

The natural approach is for the simulator $\mathcal{S}(G_0, G_1)$ to run $\mathcal{V}^*(G_0, G_1)$ to get the value of b, instead of sampling $b \leftarrow \{0,1\}$. However, in order to do this, $\mathcal{S}$ must first feed G to $\mathcal{V}^*$. This is a problem because our previous simulator first sampled b and then constructed G as a function of b.

We can get around this barrier by using a common technique called *rewinding*. Namely, the simulator will first produce a guess $b' \leftarrow \{0,1\}$ for the bit b that $\mathcal{V}^*$ will choose. Because b' is sampled independently of everything else, we have $b' = b$ with probability $1/2$. If $\mathcal{V}^*$ indeed chooses $b = b'$, then we are in good shape. Otherwise, we "rewind" the simulation and restart from the beginning, hoping that this time we correctly guess b.

More precisely, the simulator $\mathcal{S}(G_0, G_1)$ works as follows:

1. Sample $b' \leftarrow \{0,1\}$, a uniformly random permutation σ, and set $G = \sigma(G_{b'})$.
2. Send G to $\mathcal{V}^*(G_0, G_1)$. Let b be the next message sent by $\mathcal{V}^*$.
3. If $b \neq b'$, return to Step 1.
4. If $b = b'$, output (G, b, σ).

We argue about the running time and output distribution of the simulator. Because each attempt at simulating succeeds with probability $1/2$, the whole process runs in *expected* polynomial time.[3] Furthermore, as before, it is easy to see that it produces an output identically distributed to the verifier's view conditioned on $b = b'$. ■

6 Zero-knowledge proofs for all of NP?

We saw above that graph isomorphism has a nice perfectly zero-knowledge proof. Can we design such proofs for a wide range of languages, ideally systematically?

[3] Here we will slightly relax the notion of "PPT simulator" and allow it to run in expected polynomial time. In other words, the expected value of its running time (taken over the randomness of the algorithm) is polynomial in the input length.

It is easy to see that every language in P has a (trivial) zero-knowledge interactive proof system (why?). What about languages in NP? Do all of them have zero-knowledge interactive proof systems? It turns out that the answer is yes! This seminal result was proved by Goldreich, Micali, and Wigderson [GMW91]. There is a catch, however: we must relax our notion of zero-knowledge. And this relaxation is "necessary", in the sense that if all languages for NP have perfectly zero-knowledge proofs then something unreasonable happens (the polynomial hierarchy collapses) [AH87, For87].

6.1 Relaxed notions of zero-knowledge

Our notion of perfect zero-knowledge requires that the verifier's view be simulated *perfectly* and efficiently. We may relax this notion by allowing for imperfect simulation. One reasonable option is the following relaxation.

Definition 4 (Statistical zero-knowledge) *We say that an interactive protocol between a prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ for a language L is* statistically zero-knowledge *if* for any PPT verifier $\mathcal{V}^*$ *there exists a PPT simulator $\mathcal{S}$ and a negligible function ε such that for any $x \in L$ and any computationally-unbounded distinguisher $\mathcal{D}$ it holds that*

$$|\Pr[\mathcal{D}(x, \mathcal{S}(x)) = 1] - \Pr[\mathcal{D}(x, \mathsf{view}_{\mathcal{V}^*}\langle \mathcal{P}(x), \mathcal{V}^*(x)\rangle) = 1]| \leq \varepsilon(|x|).$$

If you recall the notion of *statistical distance* (a.k.a. *total variation distance*) from the problem sets, we can rephrase the notion of statistical zero-knowledge as requiring that the statistical distance between the verifier's view and the simulator's output is negligible in $|x|$.

Some additional relevant observations are in order. First, there are two separate "adversaries" in the definition. There is the dishonest verifier, who may deviate from the protocol to try to extract more information from the prover, and the distinguisher that attempts to distinguish between the verifier's full view and the simulated view. Second, we explicitly give x as an input to $\mathcal{D}$. This is technically not needed since the verifier's view includes x already. We opted for this to emphasize the dependence on x.

It turns out that we must further relax the zero-knowledge property we are aiming for. Intuitively, we will only require zero-knowledge against efficient distinguishers.

Definition 5 (Computational zero-knowledge) *We say that an interactive protocol between a prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ for a language L is* computationally zero-knowledge *if* for any PPT verifier $\mathcal{V}^*$ *there exists a PPT simulator $\mathcal{S}$ and a negligible function ε such that for any $x \in L$ and any PPT distinguisher $\mathcal{D}$ it holds that*

$$|\Pr[\mathcal{D}(x, \mathcal{S}(x)) = 1] - \Pr[\mathcal{D}(x, \mathsf{view}_{\mathcal{V}^*}\langle \mathcal{P}(x), \mathcal{V}^*(x)\rangle) = 1]| \leq \varepsilon(|x|).$$

Note that this is the same as requiring that the verifier's view and the simulator's output be computationally indistinguishable.

6.2 Computational zero-knowledge for graph 3-coloring

In order to show that every language in NP has a computational zero-knowledge interactive proof system, it suffices to show that this holds for some NP-*complete* language. Recall from your complexity/theory of computation course that a language L is NP-complete if any $L' \in \mathsf{NP}$ can be efficiently "reduced" to it. More precisely, L is NP-complete if $L \in \mathsf{NP}$ and for every $L' \in \mathsf{NP}$ there exists a deterministic polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}(x) \in L$ if and only if $x \in L'$.

NP-complete languages are the hardest languages in NP. In particular, we strongly believe that such languages L are not efficiently decidable, and so we believe that it is computationally infeasible to efficiently find a valid witness for any given $x \in L$.

Graph 3-coloring is one of the canonical examples of an NP-complete language. A *3-coloring* of an undirected, unweighted graph $G = (V, E)$ is a coloring of the vertices of G using 3 colors so that no two adjacent vertices share the same color. More precisely, a 3-coloring is an assignment $\phi : V \to \{0, 1, 2\}$ (with 0, 1, and 2 being the three colors) with the property that $\phi(u) \neq \phi(v)$ whenever $(u, v) \in E$. Then, we define the language of 3-colorable graphs

$$\mathsf{3COL} = \{G : \text{the graph } G \text{ has a 3-coloring}\}.$$

It is easy to see that $\mathsf{3COL} \in \mathsf{NP}$ (the valid 3-coloring ϕ serves as a short, deterministic, efficient proof), and it was shown to be NP-complete by Lovász in 1973 (if you are curious, see these notes and also this discussion by Chvátal). Note that having 3 colors does matter! Deciding whether a graph is 2-colorable can be done efficiently.

We will give a computational zero-knowledge interactive proof system for 3COL.

Our proof system will use a perfectly binding and computationally hiding *commitment scheme*, which we already covered in lecture 5 when constructing coin tossing protocols. We recall here the intuition. Suppose that Alice wants to commit to some message x to Bob, before revealing it. On input x and randomness r, the commitment scheme outputs a commitment $\mathsf{Com}(x, r)$. Alice sends $\mathsf{Com}(x, r)$ to Bob. To open the commitment, Alice simply reveals (x, r), and Bob can check that the commitment is valid by deterministically computing $\mathsf{Com}(x, r)$. The perfect binding property states that for any $x' \neq x$ and any r' it holds that $\mathsf{Com}(x', r') \neq \mathsf{Com}(x, r)$, i.e., Alice cannot trick Bob into thinking that she committed to a different message x'. The computational hiding property states that a PPT Bob cannot distinguish between commitments to two distinct messages $x \neq x'$. More precisely, for any PPT adversary $\mathcal{A}$ there exists a negligible function $\varepsilon(n)$ such that for any distinct $x \neq x'$ and $r, r' \leftarrow \{0,1\}^n$,

$$|\Pr[\mathcal{A}(1^n, \mathsf{Com}(x, r)) = 1] - \Pr[\mathcal{A}(1^n, \mathsf{Com}(x', r')) = 1]| \leq \varepsilon(n).$$

In Lecture 5 we constructed commitment schemes for 1-bit messages. But it is easy to extend such a commitment scheme to longer messages by separately commiting to each bit of the message. 2-bit messages are already sufficient for us.

Theorem 3 *Assume that OWFs exist. Then, every language in* NP *has a computationally zero-knowledge interactive proof system.*

In other words, all statements with efficiently checkable non-interactive proofs have zero-knowledge interactive proofs.

We prove Theorem 3 by analyzing the following protocol. Let Com be a perfectly binding and computationally hiding commitment scheme for 2-bit messages, with n-bit randomness. For simplicity, we assume that $n = |V|$.

The protocol works as follows, where the prover $\mathcal{P}$ and verifier $\mathcal{V}$ receive a 3-colorable graph $G = (V, E)$ as input:

1. The prover $\mathcal{P}$ finds a 3-coloring ϕ for G. Then, he samples a uniformly random permutation $\pi : \{0,1,2\} \to \{0,1,2\}$ and defines $\phi' = \pi \circ \phi$. Note that ϕ' is still a valid 3-coloring of G, and that for any edge $(u, v) \in E$ the corresponding colors $(\phi'(u), \phi'(v))$ are uniformly distributed in $\{0,1,2\} \times \{0,1,2\}$ conditioned on $\phi'(u) \neq \phi'(v)$.

   For each vertex $v \in V$, the prover samples randomness $r_v \leftarrow \{0,1\}^n$ and computes the commitment $c_v = \mathsf{Com}(\phi'(v), r_v)$. Then, $\mathcal{P}$ sends c_v for all $v \in V$ to the verifier. Intuitively, $\mathcal{P}$ commits to a randomly permuted 3-coloring of G.

2. The verifier $\mathcal{V}$ samples a random edge $(i, j) \leftarrow E$ and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ replies with $\phi'(i), \phi'(j), r_i, r_j$.

4. $\mathcal{V}$ verifies that $\phi'(i), \phi'(j) \in \{0,1,2\}$ (i.e., they are valid colors), that $\phi'(i) \neq \phi'(j)$ (i.e., that they are distinct colors), and verifies the commitments by checking $\mathsf{Com}(\phi'(i), r_i) = c_i$ and $\mathsf{Com}(\phi'(j), r_j) = c_j$. If all these checks hold, then $\mathcal{V}$ outputs 1. Otherwise, he outputs 0.

We begin by establishing completeness and soundness.

Lemma 1 *This protocol is complete and has soundness* $1 - 1/|E|$.

Proof: If G is 3-colorable then indeed $\phi'(i) \neq \phi'(j)$ since $\phi(i) = \phi(j)$, and so it is clear that the verifier always accepts.

Suppose that G is not 3-colorable and consider an arbitrary (potentially dishonest) prover $\mathcal{P}^*$. Suppose that $\mathcal{P}^*$ sends commitments to some coloring ϕ of G. Since G is not 3-colorable, we know that there exists an edge (u, v) such that $\phi(u) = \phi(v)$. The probability that $\mathcal{V}$ chooses to check this edge is $1/|E|$. In this case, either $\mathcal{P}^*$ refuses to reveal the openings of the commitments, or the commitments fail to verify, or the commitments are valid and $\mathcal{V}$ sees that $\phi(u) = \phi(v)$. In all of these cases $\mathcal{V}$ outputs 0 and rejects.

Therefore, this protocol has soundness $1 - 1/|E|$, which can be made at most $1/2$ by sequentially repeating the protocol $|E|$ times. ■

It now remains to show computational zero-knowledge. For the sake of simplicity, we begin by proving computational zero-knowledge against an honest verifier $\mathcal{V}$.

Lemma 2 *This protocol is computationally zero-knowledge against an honest-verifier.*

Proof: Note that the view of $\mathcal{V}$ on input G consists of

$$G, (c_v)_{v \in V}, (i,j), (\alpha_i, \alpha_j, r_i, r_j),$$

where the c_v's are commitments to a randomly permuted 3-coloring of G, (i,j) is a uniformly random edge, (α_i, α_j) the corresponding uniformly random colors subject to $\alpha_i \neq \alpha_j$, and r_i, r_j the corresponding openings of the commitments.

Consider the following simulator $\mathcal{S}$ that receives a 3-colorable graph G as input:

1. Sample $(i,j) \leftarrow E$ and $(\alpha_i, \alpha_j) \leftarrow \{0,1,2\} \times \{0,1,2\}$ conditioned on $\alpha_i \neq \alpha_j$. Sample $r_i, r_j \leftarrow \{0,1\}^n$ and compute the commitments $c_i = \mathsf{Com}(\alpha_i, r_i)$ and $c_j = \mathsf{Com}(\alpha_j, r_j)$. For all other vertices $v \in V \setminus \{i,j\}$, sample $r_v \leftarrow \{0,1\}^n$ and set $c_v = \mathsf{Com}(0, r_v)$.
2. Output $G, (c_v)_{v \in V}, (i,j), (\alpha_i, \alpha_j, r_i, r_j)$.

We must show that $\mathcal{S}(G)$ is computationally indistinguishable from the honest verifier's view. The only difference between the two distributions is that in $\mathcal{S}(G)$ the commitments c_v for $v \in V \setminus \{i,j\}$ are commitments of 0, and not of the appropriate coloring $\phi'(v)$ of G. So computational indistinguishability follows from the computational hiding property of Com and a simple hybrid argument where we change the commitments to 0 to commitments to $\phi'(v)$ one-by-one. ■

We now prove computational zero-knowledge against arbitrary verifiers.

Lemma 3 *The protocol above is computationally zero-knowledge.*

Proof: Fix an arbitrary 3-colorable graph G and an arbitrary PPT verifier $\mathcal{V}^*$. The idea is similar to what we saw before for graph isomorphism. We try to guess which edge the verifier $\mathcal{V}^*$ will choose, and if we get it wrong then we rewind the simulation. Otherwise, we continue with the "honest-verifier simulation".

More precisely, consider the simulator $\mathcal{S}$ which on input G behaves as follows:

1. Sample $(i,j) \leftarrow E$ and $(\alpha_i, \alpha_j) \leftarrow \{0,1,2\} \times \{0,1,2\}$ conditioned on $\alpha_i \neq \alpha_j$. Sample $r_i, r_j \leftarrow \{0,1\}^n$ and compute the commitments $c_i = \mathsf{Com}(\alpha_i, r_i)$ and $c_j = \mathsf{Com}(\alpha_j, r_j)$. For all other vertices $v \in V \setminus \{i,j\}$, sample $r_v \leftarrow \{0,1\}^n$ and set $c_v = \mathsf{Com}(0, r_v)$. Send $(c_v)_{v \in V}$ to $\mathcal{V}^*(G)$.
2. Suppose that $\mathcal{V}^*$ replies with $(i', j') \in E$.[4] If $(i', j') \neq (i, j)$, then go back to Step 1 and repeat. Otherwise, output
$$G, (c_v)_{v \in V}, (i,j), (\alpha_i, \alpha_j, r_i, r_j).$$

We must show that $\mathcal{S}(G)$ is computationally indistinguishable from the view of $\mathcal{V}^*$, and also that $\mathcal{S}(G)$ runs in expected polynomial time. This is a bit more subtle than it seems, because $\mathcal{V}^*$ chooses

[4] Note that $\mathcal{V}^*$ could in principle reply with an invalid pair (i', j'). E.g., this may not even be an edge! But we will ignore these edge cases which can be addressed but are boring.

(i', j') as a function of the commitments $(c_v)_{v \in V}$, which depend on (i, j). Therefore, naively, it could be the case that $\mathcal{V}^*$ always chooses $(i', j') \neq (i, j)$, and so the simulator would never terminate. We will show that this cannot happen due to the hiding property of Com.

Consider the "modified simulator" $\widetilde{\mathcal{S}}$ that gets as input G *and* a valid 3-coloring ϕ of G and behaves as follows:

1. Sample $(i, j) \leftarrow E$ and $(\alpha_i, \alpha_j) \leftarrow \{0,1,2\} \times \{0,1,2\}$ conditioned on $\alpha_i \neq \alpha_j$. Sample $r_i, r_j \leftarrow \{0,1\}^n$ and compute the commitments $c_i = \mathsf{Com}(\alpha_i, r_i)$ and $c_j = \mathsf{Com}(\alpha_j, r_j)$. Let $\pi : \{0,1,2\} \to \{0,1,2\}$ be the unique permutation such that $\pi \circ \phi(i) = \alpha_i$ and $\pi \circ \phi(j) = \alpha_j$ and set $\phi' = \pi \circ \phi$. For all other vertices $v \in V \setminus \{i, j\}$, sample $r_v \leftarrow \{0,1\}^n$ and set $c_v = \mathsf{Com}(\phi'(v), r_v)$. Send $(c_v)_{v \in V}$ to $\mathcal{V}^*(G)$.

2. Suppose that $\mathcal{V}^*$ replies with $(i', j') \in E$.[5] If $(i', j') \neq (i, j)$, then go back to Step 1 and repeat. Otherwise, output
$$G, (c_v)_{v \in V}, (i, j), (\alpha_i, \alpha_j, r_i, r_j).$$

Note that the sampling of the commitments $(c_v)_{v \in V}$ in $\widetilde{\mathcal{S}}(G, \phi)$ is *independent of the choice of* (i, j). Therefore, $\mathcal{V}^*$ must choose (i', j') independently of (i, j), and so the simulator terminates in any given iteration with probability $1/|E|$. In particular, this means that $\widetilde{\mathcal{S}}(G, \phi)$ runs in expected polynomial time, since $|E|$ is polynomial in the description size of G. Furthermore, once $\widetilde{\mathcal{S}}(G, \phi)$ terminates, its output is identically distributed to the view of $\mathcal{V}^*$ in the protocol.

It remains to show that each iteration of $\mathcal{S}(G)$ terminates with probability not much lower than $1/|E|$. First, note that we can replace the commitments corresponding to vertices $v \in V \setminus \{i, j\}$ sent by $\widetilde{\mathcal{S}}(G, \phi)$ in the first message to $\mathcal{V}^*$ by commitments of 0 by a hybrid argument, using the computational hiding property of Com. After these replacements, the two simulators behave in exactly the same way. Therefore, since an iteration of $\mathcal{S}(G)$ is computationally indistinguishable from an iteration of $\widetilde{\mathcal{S}}(G, \phi)$, it follows that an iteration of $\mathcal{S}(G)$ must terminate with probability at least $\frac{1}{|E|} - \varepsilon(|V|)$, for some negligible function ε. Since $\frac{1}{|E|} - \varepsilon(|V|)$ is still inversely polynomial in the description size of G, it follows that $\mathcal{S}(G)$ terminates in expected polynomial time as well.[6] ■

6.3 What does this mean, really?

We saw that all languages in NP have computationally zero-knowledge interactive proofs. Even more than that, these interactive proofs have efficient provers, provided that the prover receives the witness as input. Here are a few examples of things this allows you to do:

- You generate an RSA public-key/secret-key pair and give someone else the public key. You can prove to them that the public key was correctly generated without revealing any information about the secret key.

[5] Note that $\mathcal{V}^*$ could in principle reply with an invalid pair (i', j'). E.g., this may not even be an edge! But we will ignore these edge cases which can be addressed but are boring.

[6] Some things about this hybrid argument are being swept under the rug. See page 8 of these notes.

- You can show someone else a public-key encryption of your bank account balance and prove to them that you have more money than them, without revealing any extra information.
- More generally, you can give your friend a target output y and the public-key encryption of an input x and prove to them that $f(x) = y$ for a given (efficiently computable) function f, without revealing any extra information about x.

7 Recommended reading

As mentioned above, the zero-knowledge property was first introduced by Goldwasser, Micali, and Rackoff [GMR89]. Not long after, Goldreich, Micali, and Wigderson [GMW91] gave zero-knowledge proofs for all NP languages.

I already mentioned the great lecture notes of Noah Stephens-Davidowitz and Vinod Vaikuntanathan above. Besides that, the history surrounding the development of interactive proof systems is fascinating and highly impactful. For some entertaining accounts, see this article by Babai on the power of interaction, broadly construed, and this article by O'Donnell on the history of the PCP theorem, one of the crown jewels of theoretical computer science.

Zero-knowledge proofs started as a purely theoretical concept, and remained so for several years. More recently, these notions have revealed themselves to be highly impactful in practice. For an overview, see the ZKProof website.

References

[AH87] William Aiello and Johan Håstad. Perfect zero-knowledge languages can be recognized in two rounds. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 439–448, 1987.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 684–697, New York, NY, USA, 2016. Association for Computing Machinery.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 204–209, New York, NY, USA, 1987. 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. Preliminary version in STOC 1985.

[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. *J. ACM*, 38(3):690–728, July 1991. Preliminary version in FOCS 1986.