Introduction to Cryptography 04/10/2018

Lecture 19: Zero-Knowledge Proofs I

Instructor: Vipul Goyal *Scribe: Josh Ackerman*

1 Introduction

The notion of an interactive protocol is central to modern cryptography. In this lecture we explore a specific kind of interaction, *zero-knowledge protocols*, where a prover P tries to convince a verifier V of the validity of a statement, without leaking any information to V.

For example P might want to convince V of a statement like

$$\exists x, y \in \mathbb{Z} \text{ s.t. } x^3y + xy + 1 = 0$$

or,

$$\exists x_1, .., x_n \in \{0,1\}^n \text{ s.t. } \Phi(x_1, .., x_n) = 1$$

where Φ is a boolean CNF formula. In a traditional setting, a proof of one of these statements could simply consist of a record of values satisfy the equation. However, in the context of zero-knowledge proofs we require that no such information, i.e., the witness itself, is leaked in the exchange. Although the idea that one could (in a relaxed sense of the word) prove a statement without revealing any information is quite counterintuitive, we will eventually describe such a zero-knowledge protocol the graph isomorphism problem (described in section 3).

2 Definition

In the following definition we take P to be a probabilistic polynomial time algorithm structured as follows,

```
On Input (x, w, next verifier message):
    if next verifier message is empty then
        return next verifier message
    else
        return first prover message.
```

Think of x as an object which P is trying to show is in a language $\mathcal{L}$, and w as the witness proving that $x \in \mathcal{L}$. Similarly V is works as,

```
On Input (x, next prover message):
    if protocol is not over then
        return next verifier message
    else
        return ACCEPT or REJECT.
```

With these two definitions in mind, we can now formally define zero-knowledge proofs.

Definition 1. *(Zero-Knowledge Proof). Suppose $\mathcal{L}$ is a language. A zero-knowledge protocol is an interaction between two probabilistic polynomial time algorithms P and V with P trying to convince V that $x \in \mathcal{L}$ and satisfying properties (i), (ii), and (iii) described below.*

(i) *Completeness.* If $x \in \mathcal{L}$, w is the correct witness, and the protocol is honestly executed, then V outputs `ACCEPT`.

(ii) *Soundness.* Informally, we wish to capture the idea that no cheating prover can successfully lie to an honest verifier (except with small probability). Formally, if $x \notin \mathcal{L}$, for every probabilistic polynomial time algorithm algorithm $\hat{P}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that,

$$\mathbf{Pr}\big[\hat{P} \text{ convinces } V \text{ that } x \in \mathcal{L}\big] \leq \mathsf{negl}(\cdot).$$

(iii) *Zero-knowledge.* Intuitively, this definition represents the idea that P does not leak any information by necessitating that V can perform the protocol "alone", and if V can perform the protocol alone, then V does not learn any additional information from its interaction with P. This intuition is formalized by necessitating that there is a standalone simulator S which can produce a transcript which is indistinguishable from that of P and V.

Formally, an interaction is considered to be zero-knowledge if for all $x \in \mathcal{L}$ there is a probabilistic polynomial time algorithm S which can output a transcript τ' such that $\tau \approx_c \tau'$, where τ is the distribution of the original interaction transcript. The algorithm S is often called a simulator.

After reading property (iii) one might wonder, if such a simulator S must exist, then what purpose does the prover serve to V the first place? The short answer is that the interaction itself in the protocol is of pragmatic significance.

To illustrate this claim, imagine you (Victor) have a friend named Peggy who claims she can always make a fair coin land on heads. If you engaged in an interactive protocol with her, you might ask her to flip a coin 100 times and show you the result each time. If each time she shows you a head, you would be convinced that she is telling you the truth as there is only a 2^{-100} chance she is not. On the other hand, the output of a simulation of the protocol can be thought of a video of her performing the 100 coin flips, and showing the result to the camera each time. In this case you would be skeptical, since she may have modified the video in some manner such as cutting out the scenes where the coin lands on tails.

Finally, it is is worth mentioning there are other formalizations of what it means to be zero-knowledge. Property (iii) can more concretely be characterized as *computational zero-knowledge*. Besides computational zero-knowledge proofs, one might also wish to study *perfect zero-knowledge* where the distributions produced by the interaction between P and V are exactly the same, or *statistical zero-knowledge*, where the two distributions are *statistically close*.

3 Graph Isomorphism

Informally, two graphs are isomorphic if their edge sets encode the same adjacency information, i.e., you can permute the vertex labels of one of the graphs so that it is the same as the other graph. The formal definition of graph isomorphism is presented on the next page.

Definition 2. *Let $V(G), E(G)$ denote the vertex set and edge set of G respectively. Then, a pair of of graphs (G_0, G_1) is isomorphic (denoted $G_0 \simeq G_1$) if there exists a permutation $\pi : V(G_0) \mapsto V(G_1)$ such that $\forall x, y \in \overline{V(G_0), xy} \in E(G_0)$ if and only if $\pi(x)\pi(y) \in E(G_1)$ [1]. The permutation π is called an isomorphism.*

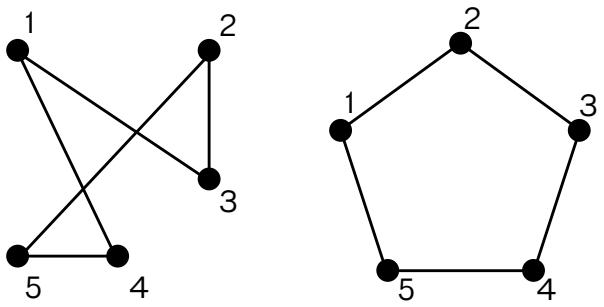

Figure 1: Two isomorphic graphs.

4 Zero-Knowledge Protocol for Graph Isomorphism

Let G_0 and G_1 be graphs on n vertices and define S_n to be the set of permutations of n elements. On input a pair of graphs (G_0, G_1), known to both parties, the protocol proceeds as follows,

Prover: Sample $\sigma \xleftarrow{\$} S_n$ and send $H := \sigma(G_0)$ to the verifier.

Verifier: Pick $ch \in \{0, 1\}$, and send ch to the prover.

Prover: If $ch = 0$ then send $\varphi := \sigma$ to the verifier, and otherwise send $\varphi := \sigma \circ \pi^{-1}$.

Verifier: Output `ACCEPT` if and only $H = \varphi(G_{ch})$.

Intuitively, the verifier asks the prover to either show $G_0 \simeq \sigma(G_0)$ or $G_1 \simeq \sigma(G_0)$ which if $G_0 \simeq G_1$ the prover should be able to do (by sending the right permutation to the verifier). We now show that this protocol satisfies completeness, soundness of $\frac{1}{2}$, and zero-knowledge. In a future lecture we will demonstrate how to use repetition to create a protocol which actually satisfies the definition of soundness in the sense of property (ii).

Theorem 1. *The above protocol satisfies completeness, soundness $\frac{1}{2}$, and zero-knowledge.*

Proof.

Completeness. To show this protocol is complete, we need to argue that when the prover has the correct permutation π and the verifier is honest, then the verifier will end by returning `ACCEPT`. So assume that π is a witness to the isomorphism of G_0 and G_1, i.e., $\pi(G_0) = G_1$. We will examine

[1] Formally the edge set of a graph E is a subset of $\binom{V}{2}$ so it would be more precise to write $\{x, y\} \in E$, but it is a bit clumsy, so often people write $xy \in E$ instead.

the cases when the verifier sends $ch = 0$ and $ch = 1$.

Case 1 ($ch = 0$). As discussed earlier, when the verifier sends $ch = 0$, he is asking the prover to show that $H \simeq \varphi(G_0)$. Recall that $H := \sigma(G_0)$, and that when the verifier sends $ch = 0$ the prover returns $\varphi := \sigma$. Certainly, $\sigma(G_0) \simeq \sigma(G_0)$, so the verifier will output `ACCEPT`.

Case 2 ($ch = 1$). Again the verifier asks the prover to show that $H \simeq \varphi(G_1)$. The prover does this by sending the verifier $\varphi := \sigma \circ \pi^{-1}$ as a witness. Since

$$H := \sigma(G_0)$$

and

$$\varphi(G_1) = \sigma \circ \pi^{-1}(G_1) = \sigma(G_0)$$

the verifier will find that $H \simeq \varphi(G_1)$, and output `ACCEPT` as desired.

Soundness $\frac{1}{2}$. We will show a weaker form of soundness. Rather than showing that if $G_0 \not\simeq G_1$ then for every probabilistic polynomial time algorithm algorithm $\hat{P}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that,

$$\mathbf{Pr}\big[\hat{P} \text{ convinces } V \text{ that } G_0 \simeq G_1\big] \leq \mathsf{negl}(\cdot)$$

we will instead show

$$\mathbf{Pr}\big[\hat{P} \text{ convinces } V \text{ that } G_0 \simeq G_1\big] \leq \frac{1}{2}.$$

So suppose $G_0 \not\simeq G_1$. By definition we know that for any graph G' either $G' \simeq G_0$ or $G' \simeq G_1$ but not both (since $\simeq$ is transitive). Essentially, this means that the prover could pass one of the tests, but not both. More specifically, if the verifier sends $ch = 0$, then the prover sends σ, in which case the verifier will accept. However, when $ch = 1$, the prover needs to come up with a permutation φ that shows $\sigma(G_0) \simeq \varphi(G_1)$, and if $G_0 \not\simeq G_1$, then such a task is not possible, causing the verifier to reject. So since the verifier only picks $ch \in \{0, 1\}$ it follows that,

$$\mathbf{Pr}\big[\hat{P} \text{ convinces } V \text{ that } G_0 \simeq G_1\big] \leq \frac{1}{2}.$$

as desired.

Zero-knowledge. We aim to construct a simulator S which outputs a transcript which is computationally indistinguishable from an honest execution of the above protocol. First we explore an incorrect attempt which illustrates many core ideas in the correct protocol, but fails for a subtle yet critical reason. Finally, we will conclude by presenting a correct version.

Incorrect Attempt: Define S as follows,

1) Sample $\sigma \xleftarrow{\$} S_n, b \xleftarrow{\$} \{0, 1\}$, and set $H := \sigma(G_b)$.
2) Choose $ch \xleftarrow{\$} \{0, 1\}$.
3) If $ch = b$ output σ, otherwise repeat from (1).
4) Output `ACCEPT`.

Now we would need to show that the distribution τ of the real protocol is indistinguishable from the distribution τ' from S. However, one might suspect from the *incorrect attempt* label attached to the above definition of S, that there is a reason why $\tau \not\approx_c \tau'$. Upon comparing S and the original protocol, one might conjecture that the problem lies in the first step of S, since S sets $H := \sigma(G_b)$ where $\sigma \xleftarrow{\$} S_n, b \xleftarrow{\$} \{0,1\}$ as opposed to the protocol which deterministically sends $\sigma(G_0), \sigma \xleftarrow{\$} S_n$. However, this disparity turns out be inconsequential, due to the following fact which essentially corroborates that this change is not computationally noticeable.

Fact 1. *If $G_0 \simeq G_1$ then for $\sigma \xleftarrow{\$} S_n$, the distributions $\{\sigma(G_0)\}$ and $\{\sigma(G_1)\}$ are equal.*

We will give a more detailed argument for why step (1) is valid towards the end of this section. The actual source of the flaw is how S simulates the verifier sampling *ch* and sending it to the prover. In S, *ch* is sampled uniformly at random, whereas there is no such restriction in the actual protocol. So while executing the original protocol, V could be bias in how it chooses *ch*. For example, V might decide to always send the verifier $ch = 1$. If this was the case, then V always sends $ch = 1$, whereas S only sets $ch = 1$ with probability $\frac{1}{2}$. This dichotomy does in fact impact computational indistinguishability, since the transcript τ' from S is always fair when τ from the original protocol may not be. So we find $\tau \not\approx_c \tau'$.

Reflecting on how to remedy this flaw, we notice that we need to give S a more realistic way of sampling *ch* in step (2). Of course we cannot fix a deterministic strategy, or really any sort of probability distribution to pick *ch* from. This observation motivates our approach to remedying the problem which is to give S access to an arbitrary black-box verifier V^* which supplies S with the random bit for *ch*. Now using V^* we present the corrected version of S.

Correct Attempt: The correct S is as follows,

1) Sample $\sigma \xleftarrow{\$} S_n, b \xleftarrow{\$} \{0,1\}$, and set $H := \sigma(G_b)$.
2) Feed H into V^* to get *ch*.
3) If $ch = b$ output σ and, otherwise repeat from (1).
4) Output `ACCEPT`.

Since each *ch* comes from an arbitrary V^*, we correct the issue faced earlier since V^* could also reflect any sort of bias present in the original protocol. Otherwise, it should be clear that S at least superficially simulates a run of the original protocol. Now we only need to argue that $\tau' \approx_c \tau$. We remark that to show $\tau' \approx_c \tau$ it suffices to show that the distribution of the output in step (1) is indistinguishable from step (1) in the actual protocol, as the rest of S is primarily identical to the protocol, when $G_0 \simeq G_1$.

By assumption, $G_0 \simeq G_1$, and so by fact (1) it follows that $\{\sigma(G_0)\} = \{\sigma(G_1)\}$ for any $\sigma \xleftarrow{\$} S_n$. Next, note that when $\sigma \xleftarrow{\$} S_n$ that $\sigma \circ \tilde{\sigma}$ for a fixed $\tilde{\sigma} \in S_n$, still behaves as a uniformly random permutation. Subsequently,

$$\{\sigma(G_0)\} = \{(\sigma \circ \pi^{-1})(G_1)\} = \{\sigma_S(G_b)\}$$

where σ_S is the permutation chosen in step (1) of S, and $b \xleftarrow{\$} \{0,1\}$. Hence, the distributions for the output of step (1) in the original protocol and S are the same, so certainly $\tau' \approx_c \tau$. This completes the proof. ■

5 Looking Ahead

In this lecture we discussed the fundamentals of zero-knowledge proofs, and presented an example of one. In future lectures we will explore the concepts such as sequential repetition to improve the probability bounds for soundness, connections to NP, non-interactive variants of zero-knowledge proofs, and applications to cryptocurrencies.