University of Illinois, Urbana Champaign
CS 598DK Special Topics in Cryptography

Instructor: Dakshita Khurana
Scribe: Shreyas Gandlur and Elizaveta Kalinina
Date: September 20, 2019

LECTURE
8

Zero Knowledge and Fiat-Shamir for NIZK

In previous lectures, we have been introduced to commitment schemes and simulation proofs. In this lecture, we will focus on using these tools to build zero-knowledge proofs. *Zero-knowledge proofs* are proof systems in which a prover proves some statement without revealing any other information. We will formalize this notion later on in the lecture. In particular, this lecture will cover:

- Review commitment schemes
- Define the properties of interactive proofs
- Formalize the notion of zero-knowledge through the simulation paradigm
- Prove that every language in $\mathcal{NP}$ has a zero-knowledge proof by constructing a zero-knowledge proof scheme for graph 3-coloring
- Describe the Fiat-Shamir technique by which non-interactive zero-knowledge proofs (NIZKPs) can be constructed

8.1 Review of Commitment Schemes

Before jumping into interactive and zero-knowledge proofs, we first review commitment schemes, which were covered in Lecture 3. A commitment scheme allows someone to commit to a value without revealing it. Intuitively, we want to guarantee that (1) the commitment doesn't reveal anything about what the value is and (2) that someone committing a value cannot change the value they commit to. Formally, it is defined as follows:

Definition 8.1. (commitment scheme) A *commitment scheme* is a set of probabilistic, polynomial time algorithms

$$C(1^\kappa; m \in \{0,1\}^\kappa; r) \to c$$
$$D(1^\kappa; c; r) \to m \in \{0,1\}^\kappa \text{ or } \perp$$

satisfying the following three properties:

- *Correctness:* $\forall m \in \{0,1\}^\kappa, \forall r, D(1^\kappa, C(1^\kappa; m; r); r) \to m$
- *Computationally Hiding:* $\forall (m_0, m_1) \in \{0,1\}^\kappa \times \{0,1\}^\kappa, C(1^\kappa; m_0; r) \overset{c}{\approx} C(1^\kappa; m_1; r)$
- *Perfectly Binding:* $\forall c, r_0, r_1, D(1^\kappa; c; r_0) = m_0 \neq \perp, D(1^\kappa; c; r_1) = m_1 \neq \perp \implies m_0 = m_1$

8.2 Interactive Turing Machines

When talking about interactive proofs, we will consider a pair of interactive Turing Machines. Informally, these are just Turing Machines with additional tapes used to communicate with each other.

Definition 8.2. (interactive Turing machine) An *interactive Turing machine (ITM)* is a deterministic multi-tape Turing Machine with a read-only input tape, a read-only random tape, a write-only output tape, a read-and-write work tape, a read-only communication tape, a write-only communication tape, and a single-cell read-and-write switch tape. A more complete definition can be found in [**?**, Section 4.2.1]

In this lecture, we will always consider a pair of ITMs such that the read-only communication tape of each coincides with the write-only communication tape of the other.

We use the following notation for ITMs: Let A and B be ITMs. We denote the output of A after an interaction with B by $\text{output}_A(\langle A(x), B(x)\rangle)$. Here, x is an input to both A and B.

8.3 Interactive Proofs

First, what is a proof? In this class, we are concerned with proving $\mathcal{NP}$-statements:

Definition 8.3. (proof system, informally) Consider $L \in \mathcal{NP}$. A *proof* is any method by which one can verify whether a string x is in L. We require a proof system for the language L to satisfy the following two properties:

- *Completeness:* A proof system is complete if, for every $x \in L$, there exists a proof that causes anyone verifying the proof to accept it.
- *Soundness:* A proof system is sound if, for every $x \notin L$, a false proof cannot be generated. That is, no one verifying a proof will accept the statement $x \in L$, if in fact this is false.

We frequently relax the definitions of completeness and soundness to their negligible-error or computational equivalents. Now, we look at interactive proof systems. An interactive proof is a challenge-response game between two parties, formally interactive Turing machines, called the prover and the Verifier. Since we deal with languages in $\mathcal{NP}$, we will always require verifiers to be probabilistic, polynomial-time. In some instances, we will deal with computationally unbounded provers. Formally, this looks as follows:

DEFINITION 8.4. A pair of interactive Turing machines (P, V) is an *interactive proof system for a language* L if machine V is probabilistic, polynomial-time, and the following two conditions hold:

- *Completeness:* For every $x \in L$, $\Pr[\text{output}_V(\langle P(x,w), V(x)\rangle) = 1] = 1 - negl(\kappa)$
- *Soundness:* For every interactive machine P^* and every $x \notin L$, $\Pr[\text{output}_V(\langle P^*(x), V(x)\rangle) = 1] = negl(\kappa)$

In both properties, the probability is taken over the random coins input to P and V.

REMARK 8.5. Any language in $\mathcal{NP}$ has a trivial interactive proof protocol. Consider $L \in \mathcal{NP}$ and let R_L be an $\mathcal{NP}$-relation for L. Consider a string x. Recall that by definition of $\mathcal{NP}$, if $x \in L$, there exists a witness w such that $(x,w) \in R_L$. If $x \notin L$, then for all possible v, $(x,v) \notin R_L$. Also, recall that R_L is an efficiently computable relation. Therefore, a trivial interactive proof protocol is for the prover to simply send w to the verifier. The verifier can efficiently check that $(x,w) \in R_L$.

EXAMPLE 8.6. What types of statements might we wish to prove? One possibility is to prove that a string x is a commitment to the zero-string. The specific language in $\mathcal{NP}$ is:

$$L = \{c \mid \exists \text{ r such that } c = Commit(1^\kappa; 0; r)\}$$

The statement to prove would be that $x \in L$.

8.4 Zero-Knowledge Proofs

We now consider proofs in zero-knowledge. Informally, a zero-knowledge proof protocol is a proof system with the additional constraint that the only thing the verifier learns is that statement being proved is true. To illustrate the difference between this and interactive protocols, consider the remark related to languages in $\mathcal{NP}$. A trivial interactive proof protocol would simply have the prover send the witness to the verifier; however, this would not satisfy a zero-knowledge property.

REMARK 8.7. Here's one way we can use zero-knowledge proofs. Suppose we can prove statements in zero-knowledge. Then we can compile any protocol secure against semi-honest adversaries into a protocol secure against malicious adversaries. At a high level, this would consist of each participant providing a zero-knowledge proof for each step, showing that each step of a protocol was performed correctly.

We now formalize this zero-knowledge property by using the simulation paradigm.

DEFINITION 8.8. (Zero-Knowledge w/ Black Box Simulation) Let (P, V) be an interactive proof system for language L. (P, V) is *black-box zero-knowledge* if there exists a PPT simulator S such that for every PPT (possibly adversarial) ITM V^*, it holds that:

$$\text{View}_{V^*}\langle P(x,w), V^*(x)\rangle \overset{c}{\approx} \text{View}_{V^*}\langle S(x)^{V^*(x)}, V^*(x)\rangle$$

Here, the notation $S(x)^{V^*(x)}$ means that S has black-box oracle access to V^*.

At first glance, this definition seems pretty amazing - the simulator can generate a view for the verifier that is indistinguishable from any view generated by interacting with the protocol, *without even knowing the witness*. This seems to imply that any prover could simply cheat by running the simulator. However, the reason that a simulator can generate a transcript based simply off the public input is that S is given the additional power of *rewinding*. This is power that a prover would not have, so the fact that a simulator exists does not mean that a prover can cheat. In addition, because a simulator generates this transcript using only what a verifier sees, a verifier does not learn anything beyond what it already knows.

REMARK 8.9. We can define zero-knowledge w/ non-black box simulation as well - that is, instead of just oracle access, simulators have access to the code of verifiers as well. The difference is in the order of quantifiers: in black box simulation, the order is $\exists S \forall V^*$, whereas with non-black box simulation, the order is $\forall V^* \exists S_{V^*}$. In particular, with non-black box simulation, there can be a different simulator for every verifier. In this class, we will only deal with zero-knowledge w/ black box simulation.

REMARK 8.10. For any language $L \in \mathcal{BPP}$, there exists a trivial zero-knowledge proof to verify whether $x \in L$: the verifier just runs the $\mathcal{BPP}$-decision machine on x and outputs the machine's answer. This is because, by definition, statements in $\mathcal{BPP}$ can be verified in probabilistic, polynomial time. Therefore, we are interested in languages not in $\mathcal{BPP}$.

8.5 Graph 3-Coloring

We now give an example of a zero-knowledge proof protocol by considering the language 3COL.

DEFINITION 8.11. A graph is 3-colorable if each vertex in the graph can be assigned a color $\in R, G, B$ such that no two connected vertices have the same color.

Two other facts about 3-colorings of graphs are the following:

1. If there exists a valid 3-coloring of a graph G, if we randomly permute the colors, we still get a valid 3-coloring of the graph.
2. If a graph G does not have a 3-coloring, then there is no way to color G with 3 colors.

REMARK 8.12. In order to prove the first statement, if we have a graph $X = (V', E')$ that has a valid 3-coloring W, where $\forall\ v, u \in V$ and $(v, c), (u, d) \in W$, if $(u, v) \in E$ then $c \neq d$. If we take each color in W and map it to a different color in W, say all of the vertices colored R become G, all the ones colored G become B and all the ones colored B become R, then the relationships between all the vertices remain the same, so no two connected vertices would share a color.

REMARK 8.13. The second statement is self-evident. If there were a way to color the graph using 3 colors such that no two connected vertices shared the same color, then the graph would be 3-colorable.

Now for the construction of the proof. In one iteration of the protocol, the prover can cheat. We, however, want the verifier to detect the cheating with some non-negligible probability.

The protocol should also not reveal any information about the coloring.
The prover P in this case wants to prove to the verifier V that they know a three coloring of a graph. One iteration of the protocol can be done in three steps:

1. P first creates a coloring of the graph using a random permutation of colors. P then sends a commitment of this coloring to V, in the form of $\forall v_i \in V'$, Commit(1_κ, i, color(v_i), r_i).

2. The verifier then chooses edge $e \in E'$ as a challenge and sends it to P.

3. P sends the color and randomness for the two vertices corresponding to the edge that V selected. V can then decommit the commitments for these vertices sent in step one. V then verifies that they match the colors that P sent and that the colors are in fact different - if these do not hold, then V detects cheating by the prover.

Notice that since only one edge gets checked out of the graph, a cheating prover may not be caught most of the time in one iteration. However if P is dishonest, then there is at least one edge where both vertices share a color. This implies that a cheating prover can be caught with probability $\geq \frac{1}{|E'|}$. So $\Pr[V \text{ accepts} \mid x \notin L] \leq 1 - \frac{1}{|E'|}$.

We want $\Pr[V \text{ accepts} \mid x \notin L]$ to be negligible. We achieve this using a technique called *soundness amplification* that works as follows: run the above protocol sequentially n times. Then the probability of accepting becomes $\leq (1 - \frac{1}{|E|})^n$. When $n = \max(|E'|, \kappa)$, the probability of a cheating prover successfully convincing the verifier becomes negligible.

Remark 8.14. How can we be sure that the verifier does not learn more and more about the 3-coloring with each iteration? Here's some intuition: at the beginning of each iteration, P randomly permutes the colors of their coloring. As we have seen, a permutation of the colors is still a proper 3-coloring. So, in each iteration, the verifier only learns that two vertices sharing an edge have different colors: but this is required by the definition of a 3-coloring, so the verifier learns nothing more than that the prover might know a proper 3-coloring. We now this intuition by constructing a simulator.

8.6 Simulator for Graph 3-Coloring against a Malicious Adversary

Let us construct a simulator for the protocol above with n sequential iterations. The goal of the simulator is to always make the verifier output 1, regardless of whether the graph is actually in 3COL. The simulator pretends to be the prover and works as follows: it follows the same protocol as the prover, except that, in the first round, it only sets one edge with different colors and chooses null commitments for the rest of the vertices. The simulator then sends V commitments to these vertices. The challenge is sent by V in the same way as before, where V picks an arbitrary edge in the graph. Here the simulator does one of two things:

1. If the edge that V requests is the same as the edge that the simulator set to have different colors for the vertices, then the simulator sends V the colors and the randomness used for the commitments.

2. If the edge is not the same edge, then the simulator rewinds the prover to the beginning of the current iteration and tries again.

For each iteration, the simulator will guess the correct edge in $\approx |E|^2$ tries. Since there are a polynomial number of iterations, and each iteration is reset a polynomial number of times, the simulator runs in polynomial time, as needed.

We notice that, in the view of the verifier, a transcript interacting with the prover and a transcript interacting with the simulator will look indistinguishable - because of the hiding property of commitments, the commitments received from P and from S look indistinguishable. After that, the only other information that the verifier receives is the colors and randomness associated with the commitments for the edge that the verifier chooses. This looks the same whether the verifier interacts with the simulator or the prover. Since each transcript interacting with the prover and with the simulator look indistinguishable, and there are a polynomial number of transcripts (one for each iteration), the entire execution is indistinguishable. Therefore, we have shown that the above protocol satisfies the zero-knowledge definition.

REMARK 8.15. 3-coloring is an example of an $\mathcal{NP}$-complete problem. Since there exists a zero-knowledge proof protocol for 3-coloring, we can prove every statement in $\mathcal{NP}$ in zero-knowledge. There are some subtleties that need to be addressed - the full proof of this can be found in [**?**, Section 7.3]

8.7 Reducing Round Complexity

Suppose we wish to reduce the round complexity of the above protocol (which runs in $3n$ rounds) by running each iteration in parallel. Consider a protocol that works in three rounds, where n different iterations of the protocol are run in parallel. We immediately run into a problem with this, however: this cannot be simulated in polynomial time. The intuition for this is as follows: in the first round, the prover sends its commitments and in the second round, the verifier sends a list of edges. However, the verifier might have chosen the edgelist it sends in the second round as a function of the commitments the prover sends in the first round. This means that after the simulator rewinds, it is as if the entire protocol is starting again from scratch. This implies that the expected number of rewinds is exponential, and that we cannot construct a PPT simulator, implying that we cannot prove the zero-knowledge property.

OPEN PROBLEM 8.16. Can we construct a 3-round protocol for malicious verifier black-box zero-knowledge? Most people believe that we cannot, but this is still a big open problem in the field.

We can attempt to solve the issue above by adding a fourth round to the protocol - at the very beginning of the protocol, the verifier commits to the edges that it wants to use as its challenge. In round two, the prover sends the commitments - this implies that the verifier cannot choose its challenge based on the commitments the prover sends. Then in round three, the verifier decommits to its challenge, and in round four, the prover decommits to the vertices associated with each edge. This *does* resolve the issue from above. However, the issue now is that the commitments bind the verifier to only one set of edges - an unbounded

prover can now decommit to all of these edges and the soundness of the protocol no longer holds.

8.8 Fiat-Shamir Paradigm

We now show a technique that can be used to extend interactive zero-knowledge proof protocols into non-interactive zero-knowledge proof protocols. We assume the existence of a random oracle, which works by choosing a random output for every input. A random oracle, once queried with a previously queried input, returns the same output. Here we introduce the idea of a Random Oracle, a truly random function, that can be used to make zero-knowledge proofs non-interactive. A Random Oracle works in such a way that for any input it gives a random output, however given the same input twice, it would return the same output. Here is pseudocode for how a random oracle functions:

Algorithm 1: Pseudocode for random oracle on input x

```
T ← new table;
if x in T then
    return T[x];
else
    v ←$ {0,1}^κ;
    T[x] ← v;
    return T[x];
```

This non-interactive protocol is outlined as follows:

1. The prover P can commit to all of the colors for the vertices
2. P passes these commitments as the input to the random oracle. It then treats the output as the challenge list of edges
3. P can then publicly outputs these commitments, as well as the randomness and colors associated with the edges received from the random oracle challenge.
4. V can verify everything by himself without any interaction. Since the random oracle outputs the same value when provided the same input, V can input the commitments that P publicly output into the random oracle. Using the challenge output by the random oracle, V can then use the randomness and colors that P publicly output and verify that the decommits match and everything checks out.

Remark 8.17. Random oracles cannot be efficiently represented and therefore cannot possibly exist in the real world. In practice, we might use hash functions in place of random oracles to achieve the same result. The implicit underlying assumption when using a hash function in place of a random oracle is that a hash function "behaves like" a random oracle. Please see additional reading for benefits and pitfalls of this approach.

8.9 Conclusion

In this lecture, we saw the properties that define zero-knowledge proofs, and saw an example of a simulator construction. We also discussed the Fiat-Shamir approach to converting interactive protocols into non-interactive ones. Next class, we will finish up our discussion of non-interactive zero-knowledge proofs before going into post-quantum cryptography.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.