# Lecture 2: Interactive Proofs

## 1 Overview

In this lecture, we describe the notion of *computational indistinguishability*, which is one of the most important ideas in modern cryptography. We then describe *interactive proof systems*, which are two-party protocols involving a *prover* and a *verifier*. The idea is that the prover and verifier interact by exchanging messages, and the prover's goal is to convince the verifier of some piece of information. Finally, we describe an extension of interactive proofs, called zero-knowledge proofs, in which the prover can convince the verifier that some piece of information is true—without revealing anything about the information!

## 2 Computational Indistinguishability

We introduce the notion of *computational indistinguishability*, which is one of the most important ideas in modern cryptography. Informally, we say that two probability distributions are computationally indistinguishable if they "look the same" to any computationally bounded "observer." Before diving into the formal definition of computational indistinguishability, we introduce several preliminaries.

In the last lecture, we ended with a discussion of the discrete logarithm problem. In particular, we explained that many cryptosystems can be built assuming that one of the problems related to discrete logarithms is difficult to solve. We said that a problem is difficult to solve if there is no efficient algorithm that can solve it, except with very low (negligible) probability. A bit more formally, a problem is difficult to solve if there is no efficient algorithm that can solve it, except with a probability that is negligible in a security parameter, denoted by $\lambda$.

**Definition 1 (Negligible Function).** A function $\epsilon\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive integer $n$ there exists an integer $\lambda_0$ such that for every $\lambda > \lambda_0$ we have

$$|\epsilon(\lambda)| < \frac{1}{\lambda^n}.$$

Equivalently, $\epsilon$ is negligible if for every positive polynomial $\mathrm{poly}(\cdot)$ there exists a positive integer $\lambda_0$ such that for every $\lambda > \lambda_0$ we have

$$|\epsilon(\lambda)| < \frac{1}{\mathrm{poly}(\lambda)}.$$

Put simply, $\epsilon$ is negligible if it grows slower than any positive polynomial.

*Examples of negligible functions.*
1. $f(n) = 2^{-n}$
2. $f(n) = 2^{-\sqrt{n}}$
3. $f(n) = n^{-\log n}$

*Properties of negligible functions.* Negligible functions obey certain closure properties.
1. If $\epsilon_1$ and $\epsilon_2$ are negligible functions, then the function $\epsilon_3(n) = \epsilon_1(n) + \epsilon_2(n)$ is negligible.
2. If $\epsilon_1$ is a negligible function, then for any positive polynomial $\mathrm{poly}$ the function $\epsilon_2(n) = \mathrm{poly}(n) \cdot \epsilon_1(n)$ is negligible.

Next, we give the definition for a probability distribution ensemble. The reason we deal with probability distributions is because the algorithms we consider are probabilistic; the output of a probabilistic algorithm can be described by a probability distribution over the possible outputs.

**Definition 2 (Probability Distribution Ensemble).** A *probability distribution ensemble* is a family of probability distributions $\{X_\lambda\}_{\lambda \in \mathbb{N}}$ with index set $\mathbb{N}$. In cryptography, $\lambda$ is the security parameter, which usually denotes the length of an input, and we are generally interested in probability distributions over $\{0,1\}^*$.

Having explained negligible functions and probability ensembles, we now give the definition of computational indistinguishability.

**Definition 3 (Computational Indistinguishability).** We say that two ensembles $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are *computationally indistinguishable* (abbreviated as $\{X_\lambda\}_{\lambda\in\mathbb{N}} \approx \{Y_\lambda\}_{\lambda\in\mathbb{N}}$) if for any non-uniform probabilistic polynomial time algorithm $D$ (called a *distinguisher*), there exists a negligible function $\epsilon(\cdot)$ such that $\forall\lambda \in \mathbb{N}$ we have

$$|\Pr[t \leftarrow X_\lambda, D(t) = 1] - \Pr[t \leftarrow Y_\lambda, D(t) = 1]| < \epsilon(\lambda).$$

Another way to interpret computational indistinguishability is that a polynomial time distinguisher cannot determine whether a sample is drawn from one distribution or the other with more than negligible probability over $1/2$, i.e., it cannot do much better than simply guessing.

## 3 Interactive Proof Systems

In this section, we describe the notion of interactive proof systems, in which a *prover* wishes to convince a *verifier* that a statement is true. First, we provide a brief review of some ideas from complexity theory.

A *decision problem* is a yes-or-no question on an infinite set of inputs. Inputs can be represented as strings over some alphabet, and the subset of strings for which the problem returns "yes" is called a *formal language*. The particular class of decision problems we are interested in for interactive proofs is the class of NP (nondeterministic polynomial time) decision problems.

**Definition 4 (NP).** NP is the class of languages that have polynomial time verifiers. In other words, it is the class of languages whose members all have short certificates of membership, which we call *witnesses*, that can be checked in polynomial time.

Another tool that we will use often is the Turing Machine (variations of it, to be exact). As mentioned previously, in cryptography, we are primarily concerned with probabilistic algorithms. These can be modeled by probabilistic Turing machines, which are Turing machines that can make random coin flips. When we are concerned with *efficient* probabilistic algorithms, we can model them as probabilistic polynomial time (abbreviated as p.p.t.) Turing machines. Put simply, p.p.t. Turing machines can make random coin flips and it produces output in a polynomially bounded number of steps. Of course, since both of these kinds of Turing machines are probabilistic, we can describe their outputs as probability distributions.

For interactive proof systems, we describe another extension to the Turing machine, called an Interactive Turing Machine (ITM).

**Definition 5 (Interactive Turing Machine).** An *Interactive Turing Machine* (ITM) is a Turing machine with a read-only input tape, a read-only auxiliary input tape, a read only random tape, a read/write working tape, a read-only communication tape, and a write-only communication tape.

So, an interactive protocol $(A, B)$ is simply a pair of ITMs that share communication tapes: The write-only communication tape of one ITM is the read-only communication of the other, and vice versa. In an interactive proof system, a *prover* wishes to convince a *verifier* that a statement is true. The verifier is a polynomial time bounded machine; otherwise, it could figure out the witness itself.

An interactive proof system should satisfy two properties: a prover should be able to convince a verifier of a true statement (completeness), but a malicious prover should not be able to convince a verifier of a false statement (soundness). Before giving the formal definition, we explain some notation.

**Notation.** Let $M_A = \{m_A^1, \ldots, m_A^n\}$ and $M_B = \{m_B^1, \ldots, m_B^n\}$ be vectors of strings and let $x, r_A, r_B, z_A, z_B \in \{0,1\}^*$. An execution of the protocol $(A, B)$ is described as

$$((x, z_A, r_A, M_A), (x, z_B, r_B, M_B)),$$

where ITM $A$ runs on common input $x$, auxiliary input $z_A$, and random tape $r_A$, and $m_A^i$ is the $i^{\text{th}}$ message received by $A$ (similarly for $B$).

Let $V_A = (x, z_A, r_A, M_A)$ denote the *view* of ITM $A$. Let $\text{view}_A[A_{r_A}(x, z_A) \leftrightarrow B_{r_B}(x, z_B)]$ denote $A$'s view in the execution $A_{r_A}(x, z_A) \leftrightarrow B_{r_B}(x, z_B)$. Let the tuple $(M_A, M_B)$ denote the transcript of the execution. Let $\text{out}_A(e)$ denote the output of $A$ in an execution $e$.

Let $A(x, z_A) \leftrightarrow B(x, z_B)$ (notice that the random tapes are omitted) denote the probability distribution of the random variable obtained by selecting the random tapes $r_A$ and $r_B$ randomly and independently, and then outputting $A_{r_A}(x, z_A) \leftrightarrow B_{r_B}(x, z_B)$. The corresponding probability distributions for view and out are analagously defined.

Now, the formal definition for an interactive proof is the following:

**Definition 6 (Interactive Proof).** A pair of ITMs $(P, V)$ is an interactive proof system for a language $L$ if $V$ is a p.p.t. machine and the following properties hold:

1. (Completeness) For every $x \in L$, there exists a witness string $y \in \{0,1\}^*$ such that for all auxiliary strings $z$ we have
$$\Pr[\text{out}_V[P(x, y) \leftrightarrow V(x, z)] = 1] = 1.$$
2. (Soundness) There exists a negligible function $\epsilon$ such that for all $x \notin L$, provers $P^*$, and auxiliary strings $z$ we have
$$\Pr[\text{out}_V[P^*(x) \leftrightarrow V(x, z)] = 0] > 1 - \epsilon(|x|).$$

See Section 4 in the Pass and Shelat book for more details and examples.

## 4 Zero-Knowledge Protocols

We now extend the idea of interactive proofs to zero-knowledge proofs. The idea is that the prover can convince the verifier that some statement is true without actually revealing any other information about the statement.

**Definition 7 (Honest Verifier Zero-Knowledge).** Let $(P, V)$ be an interactive proof system for the NP language $L$ with witness relation $R_L$. $(P, V)$ is honest verifier zero-knowledge if there exists a p.p.t simulator $S$ such that for every n.u. p.p.t. distinguisher $D$, there exists a negligible function $\epsilon(\cdot)$ such that for every $x \in L$, $y \in R_L(x)$, $z \in \{0,1\}^*$, $D$ distinguishes the following distributions with probability at most $\epsilon(\lambda)$.

- $\{\text{view}_V[P(x, y) \leftrightarrow V(x, z)]\}$
- $\{S(x, z)\}$

The idea here is that whatever $V$ saw in the interactive proof could have been generated by $V$ itself by running the simulator $S(x, z)$. The reason this definition is *honest verifier* zero-knowledge is because it ensures that the verifier gains no additional information as long as it follows the prescribed protocol. The following modified definition ensures that even a malicious verifier does not gain any additional information.

**Definition 8 (Zero-Knowledge).** Let $(P, V)$ be an interactive proof system for the language $L \in NP$ with witness relation $R_L$. $(P, V)$ is zero-knowledge if for every p.p.t adversary $V^*$ there exists a p.p.t simulator $S$ such that for every n.u. p.p.t. distinguisher $D$, there exists a negligible function $\epsilon(\cdot)$ such that for every $x \in L$, $y \in R_L(x)$, $z \in \{0,1\}^*$, $D$ distinguishes the following distributions with probability at most $\epsilon(\lambda)$.

- $\{\text{view}_{V^*}[P(x, y) \leftrightarrow V^*(x, z)]\}$
- $\{S(x, z)\}$

TODO: *Intuition on zero-knowledge definitions.*

## 5 Sigma Protocols

TODO: *Under construction. In the meantime, see Susan Hohenberger's notes (linked on website).*

We describe zero-knowledge proofs in Camenisch-Stadler notation. The following is a proof for the statement $g^x = X$ with witness $x$.

$$\text{PK}\{(x) : g^x = X\}$$

**Definition 9 (Discrete Log Assumption).** Let $\{G_\lambda\}_\lambda$ be a family of groups where $|G_\lambda| = 2^{\text{poly}(\lambda)}$. Let the advantage of an adversary $A$ in the discrete log problem to be

$$\text{Adv}_A^{DLog}(\lambda) = \Pr[x \leftarrow \mathbb{Z}_{|G_\lambda|}, x' \leftarrow A(1^\lambda, g^x) : x = x'].$$

The discrete log problem is hard for $\{G_\lambda\}_\lambda$ if there exists a negligible function $\epsilon(\cdot)$ such that for all adversaries $A$ we have

$$\text{Adv}_A^{DLog}(\lambda) = \epsilon(\lambda).$$

**Prover** P($x$)
- $k \leftarrow \mathbb{Z}_{|G|}$
- $K \coloneqq g^k$
- SEND($K, V$) ▷ Send commitment $K$.
- ONRECEIVE($c, V$) ▷ Receive challenge $c$.
- $s \coloneqq xc + k$ SEND($s, V$) ▷ Send response $s$.
- **return**

**Verifier** V($X$)
- ONRECEIVE($K, P$) ▷ Receive commitment $K$.
- $c \leftarrow \mathbb{Z}_{|G|} \setminus \{0\}$
- SEND($c, P$) ▷ Send challenge $c$.
- ONRECEIVE($s, P$) ▷ Receive commitment $s$.
- **if** $X^c K = g^s$ **then**
  - **return** ACCEPT
- **else**
  - **return** REJECT

To demonstrate that the above protocol is indeed an interactive proof, we show that it satisfies the soundness and completeness properties. Showing completeness is straightforward:

$$X^c K = (g^x)^c g^k = g^{xc+k} = g^s$$

Showing soundness requires building an extractor: given the transcripts of two runs of the protocol with identical commit phases ($(K, c_1, s_1)$ and $(K, c_2, s_2)$), one can extract the witness $x$. Since $s_1 = xc_1 + k$ and $s_2 = xc_2 + k$, this implies that $x = \frac{s_1 - s_2}{c_1 - c_2}$.

To show that the protocol is honest-verifier zero-knowledge, we must construct a simulator $S$ such that

$$\{\text{view}_V[P(x) \leftrightarrow V(g^x)] \approx \{S(g^x)\}.$$

The view of $V$ in the interactive proof is $(K, c, s)$ (the commitment, the challenge, and the response).

**Simulator** S($X$)
- $s \leftarrow \mathbb{Z}_{|G|}$
- $c \leftarrow \mathbb{Z}_{|G|} \setminus \{0\}$
- $K \coloneqq g^s / X^c$
- **return** $(K, c, s)$

## 6 More Resources

For further details on complexity theory and interactive proofs, see Sipser [3]. Matthew Green has two fantastic primers on zero-knowledge proofs. [1,2]. Susan Hohenburger's lecture notes on proofs of knowledge explains the Schnorr protocol in more detail (`https://www.cs.jhu.edu/~susan/600.641/scribes/lecture10.pdf`).

## References

1. Matthew Green. Zero knowledge proofs: An illustrated primer. `https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/`, 2013.
2. Matthew Green. Zero knowledge proofs: An illustrated primer, part 2. `https://blog.cryptographyengineering.com/2017/01/21/zero-knowledge-proofs-an-illustrated-primer-part-2/`, 2017.
3. Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.