1 Zero Knowledge

1.1 Warm-Up: Zero Information

When does a random variable Y reveal *no information* about another random variable X? In probability theory, the answer to this question is: when X and Y are *independent*. One equivalent form of independence is that for *any* fixed value of X, the conditional distribution of Y (given X) is always just some certain distribution D_Y.

Observe that perfect secrecy can be seen in this light: a shared-key encryption scheme is perfectly secure if (and only if) for any message $m \in \mathscr{M}$, the distribution of the ciphertext $c \in \mathscr{C}$ (over the choice of $k \leftarrow \mathsf{Gen}$) is the same: some fixed distribution D' over $\mathscr{C}$.

1.2 Zero *Computational* Knowledge

What is the *computational* analogue of revealing no information? When working with bounded computation, we speak of *knowledge* instead of information. This is because even though a value Y might reveal *all* the information in X, that information might not be accessible to a bounded algorithm. For example, for a one-way permutation f with hard-core predicate h, the value $f(x)$ reveals all the information in x, but it gives no *knowledge* about $h(x)$, because $h(x)$ "looks like" a uniformly random bit. Seen another way: up to computational indistinguishability, $h(x)$ "might as well have been" produced by an *efficient algorithm that is not given x at all!* (That is, the algorithm that just outputs a uniformly random bit.)

Let us see a concrete example of this kind of definition for shared-key encryption. Informally, it says that a ciphertext of a message m might as well be produced by an efficient algorithm (a simulator) that is not given the message. (Note the syntactic similarity to the information-theoretic statements above.)

Definition 1.1. A shared-key cryptosystem SKC for message space $\mathscr{M}$ is *zero-knowledge* if there exists an nuppt *simulator* $\mathscr{S}$ such that for all $m \in \mathscr{M}$,

$$\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m)\} \stackrel{c}{\approx} \{\mathscr{S}(1^n)\}.$$

Recall our simplest definition of security for shared-key encryption, one-time indistinguishability, which says that for all $m_0, m_1 \in \mathscr{M}$,

$$\{k \leftarrow \mathsf{Gen} : \mathsf{Enc}_k(m_0)\} \stackrel{c}{\approx} \{k \leftarrow \mathsf{Gen} : \mathsf{Enc}_k(m_1)\}.$$

Theorem 1.2. *A shared-key cryptosystem SKC is one-time indistinguishable if and only if it is zero-knowledge.*

Proof. If SKC is zero-knowledge, let $\mathscr{S}$ be the simulator guaranteed by Definition 1.1. Then by hypothesis, for all $m_0, m_1 \in \mathscr{M}$,

$$\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_0)\} \stackrel{c}{\approx} \{\mathscr{S}(1^n)\} \stackrel{c}{\approx} \{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m_1)\}.$$

One-time indistinguishability follows by the hybrid lemma.

If SKC is one-time indistinguishable, let $m' \in \mathscr{M}$ denote some arbitrary message, and define $\mathscr{S}(1^n)$ to choose $k \leftarrow \mathsf{Gen}(1^n)$ and output $c \leftarrow \mathsf{Enc}_k(m')$. Then by one-time indistinguishability, for any $m \in \mathscr{M}$ we have

$$\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m)\} \stackrel{c}{\approx} \{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m')\} \equiv \{\mathscr{S}(1^n)\}. \qquad \square$$

1.3 Why Zero Knowledge?

What do we gain from zero-knowledge/simulation-style security definitions? So far, not much other than a new perspective. However, this new perspective is very powerful: it can capture, in a unified way, many of the disparate security definitions we have seen so far (e.g., for pseudorandom generators/functions, signatures, etc.). Even more importantly, this perspective will allow us to give meaningful security definitions for protocols in which the adversary may be an *active participant*. By "active" we mean not just that the adversary can make queries to the scheme (e.g., chosen-message or chosen-ciphertext attacks), but that it instead *plays the role* of one of the scheme's algorithms (possibly in a malicious way).

2 Interactive Proofs

2.1 Motivation

What is a (mathematical) proof? Two of the most common answers include:

1. A sequence of sound logical deductions, starting from a set of axioms and/or hypotheses, that concludes in a theorem?

2. An argument that convinces the peer community?

A "classical" proof (in the mold of the first answer above) is a *static* object, which can be easily checked by applying mechanical rules. In contrast, many proofs in mathematics have required a lot of effort to find — much more effort than it takes to verify them. Intuitively, then, a classical proof may reveal more than zero knowledge to the party verifying it, in that the verifier may not have been able to generate the proof itself.

In practice, many proofs are partially "interactive:" a prover asserts a theorem, and a verifier (e.g., the peer community) asks questions of the prover until it is satisfied that the theorem is correct. It turns out that this style of proof in some cases appears to be qualitatively *more powerful* than a static, classical proof.

Consider the following example "theorem:" Coke and Pepsi are different soft drinks. To prove this claim, it might not suffice to give different recipes for the two drinks. First, how does one check that a given recipe corresponds to the claimed drink? And even if one could, how can one be sure that two different recipes don't end up producing the same end product (e.g., by a different sequence of chemical reactions)?

On the other hand, consider the following interactive "proof" for establishing the theorem: the verifier secretly buys either a bottle of Coke or Pepsi at random, pours it into a glass, and asks the prover to identify the drink (by looking at, tasting it, or whatever the prover requires to decide). The prover and verifier may repeat the exercise multiple times (with a fresh random choice of drink each time). If the theorem is indeed true (Coke and Pepsi are different), then a sufficiently talented prover can always identify correctly. However, if the theorem is false (Coke and Pepsi are exactly the same), then the prover would have only a $1/2$ chance of guessing successfully in each iteration. Therefore, if the prover succeeds in every iteration, the verifier can conclude — with very high but not complete confidence — that the theorem is true.

2.2 Formalization

Let P and V denote "interactive" algorithms, which take inputs and random coins as usual, but also (informally speaking) can pass messages to each other. Only one machine at a time performs "work;" when it passes a message to the other, the other machine starts working. We denote their joint execution on their

respective inputs by $P(\cdot) \leftrightarrow V(\cdot)$. The notation $\text{out}_V[\cdot]$ denotes the output of V in the specified interaction. A *language* L is just a subset of $\{0,1\}^*$, whose elements we often call "theorems."

Definition 2.1. An *interactive proof system* (with *soundness error* $s \in [0,1]$) for a language $L \subseteq \{0,1\}^*$ is a pair of algorithms: a (possibly computationally unbounded) prover P, and a ppt verifier V, with the following properties:

1. *Completeness* ("the specified prover convinces the specified verifier of a true statement"): For all $x \in L$, $\text{out}_V[P(x) \leftrightarrow V(x)] = 1$, with probability 1.

2. *Soundness* ("*no* possibly cheating prover can convince the specified verifier of a false statement"): For every computationally unbounded P^* and all $x \notin L$,

$$\Pr[\text{out}_V[P^*(x) \leftrightarrow V(x)] = 1] \leq s.$$

Remark 2.2. Some notes on the above definition:

- Completeness is a property of the specified ("honest") prover and verifier algorithms, and says nothing about security. Soundness protects the verifier: as long as it runs its specified V algorithm, a cheating prover — *no matter how it might arbitrarily deviate from the protocol!* — will not be able to convince it of a false theorem (with probability greater than s).

- We could relax the completeness condition to require $\text{out}_V[P(x) \leftrightarrow V(x)] = 1$ with probability at least $1 - c > s$ for some c called the *completeness* error. Every proof system we'll see in this course will have *perfect* completeness, i.e., $c = 0$.

- If we allow the probabilistic verifier machine and the all-powerful prover to interact for a polynomial number of rounds, we get the class of problems called IP. In 1992, Adi Shamir found one of the central results of complexity theory: IP equals PSPACE, the class of problems solvable by an ordinary deterministic Turing machine using polynomial space (and unbounded time).

2.3 Interactive Proof for Graph Nonisomorphism

A formal analogue of the Coke-versus-Pepsi question is the *graph nonisomorphism* problem, defined below.

A graph $G = (V,E)$ consists of a finite set V of vertices, and an edge set $E \subseteq V \times V$ (where $(v_0, v_1) \in E$ means that there is an edge between vertices v_0 and v_1). Without loss of generality, we often say that the vertex set is $[n] = \{1, \ldots, n\}$ for some nonnegative integer n.

Definition 2.3. Two graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ are *isomorphic*, written $G_0 \equiv G_1$, if there exists a bijection $\rho : V_0 \to V_1$ such that $(v_0, v_1) \in E_0$ if and only if $(\rho(v_0), \rho(v_1)) \in E_1$.

Definition 2.4. The *graph nonisomorphism language* GNI is defined as

$$\mathsf{GNI} = \{(G_0, G_1) : G_0, G_1 \text{ are graphs such that } G_0 \not\equiv G_1\}.$$

To date, it is unknown whether there exists an efficient algorithm that decides the graph (non)isomorphism language (though there *are* efficient algorithms that decide it for a broad class of special cases). It is not even known if GNI is in NP, i.e., if membership in GNI can be efficiently verified given a (polynomial-length) static proof. However, we are able to give a very simple *interactive* proof for GNI.

Protocol 2.5. We define the following verifier and prover algorithms for GNI, which interact as follows:

1. $V(G_0, G_1)$: Choose $b \leftarrow \{0,1\}$ and a uniformly random permutation π of V_b (the vertex set of G_b). Send the permuted graph $H = \rho(G_b)$ to the prover. (Intuitively, the verifier is challenging the prover to identify which of the original two graphs H is isomorphic to.)

2. $P(G_0, G_1)$: Upon receiving a graph H from the verifier, find (perhaps using a large amount of computation) a $b' \in \{0,1\}$ such that G' is isomorphic to $G_{b'}$. Send b' to the verifier. (Note that if G_0 and G_1 are not isomorphic, then there is at most one valid b'.)

3. $V(G_0, G_1)$: If $b' = b$, accept; otherwise, reject.

Theorem 2.6. *The algorithm pair (P,V) described above forms an interactive proof system for GNI, with soundness error $1/2$.*

Proof. First we show completeness. Let $(G_0, G_1) \in \mathsf{GNI}$, i.e., G_0 and G_1 are not isomorphic. Then no graph can be isomorphic to both G_0 and G_1. It follows that for any H generated by (honest) V with random choice b, the bit b' computed by the prover always equals the b chosen by the verifier, and V always accepts.

Now we show soundness. If $(G_0, G_1) \notin \mathsf{GNI}$, i.e., G_0 and G_1 are isomorphic, then a uniformly random permutation of G_0 and G_1 are identically distributed. More precisely, suppose that $G_1 = \pi(G_0)$ for some permutation π. Then for any permutation ρ, we have $\rho(G_1) = (\rho \circ \pi)(G_0)$. In particular, if ρ is a uniformly random permutation, then $\rho \circ \pi$ is also uniformly random, hence $\rho(G_0)$ and $(\rho \circ \pi)(G_0)$ are identically distributed. Therefore, the graph H sent by the verifier is statistically independent of its internal bit b, and for any prover reply b', the probability that $b' = b$ is exactly $1/2$, as desired. □