Lecture 3 - Statistical Security, Computational Models

Boaz Barak

September 22, 2005

Admin Project, holidays

Objections to impossibility result Last time we saw that a perfectly secure encryption scheme requires key size $\geq$ message size.

Whenever faced with an impossibility result that says we can not do something we want, it is a good idea to examine the underlying assumptions behind this result, and see if we can relax these assumption to still get what we want (or at least something close to that).

There are two natural relaxations for the *perfect secrecy*. We'll start with the first one, which is that we can allow the posteriori probability of guessing to be slightly larger than the a-priori.

Statistical security Suppose that we allowed the adversary to have a tiny advantage in its posteriori guessing probability compared to the a-priori probability.

For example, we say that a scheme enjoys ϵ *statistical semantic security* if in the semantic security game the adversary guesses $f(x)$ with probability at most max-pr$f(X) + \epsilon$. Note that we replaced the word *perfect* with ϵ *statistical*.

Similarly, we can say that a scheme is ϵ *statistically indistinguishable* if the probability adversary guesses the messages is at most $\frac{1}{2} + \epsilon$.

If ϵ is very small (say 10^{-6} or maybe even 10^{-100}) then ϵ-secure schemes will be just as good as perfectly secure schemes for all practical purposes.

Thus, if we could bypass the impossibility result for perfect secrecy using this notion this would be great.

Equivalence It turns out both of these definitions are again essentially equivalent to a relaxation of perfect secrecy which we call ϵ*-statistical secrecy*. This means that for every x, x' the distributions Y_x and $Y_{x'}$, even if not identical, are still within at most ϵ *statistical distance*.

Definition 1. Let X and Y be two distributions over $\{0,1\}^n$. The *statistical distance* of X and Y, denoted by $\Delta(X,Y)$ is defined to be

$$\max_{T \subseteq \{0,1\}^n} \Big| \Pr[X \in T] - \Pr[Y \in T] \Big|$$

if $\Delta(X,Y) \leq \epsilon$ we say that $X \equiv_\epsilon Y$

Lemma 1.

$$\Delta(X,Y) = \frac{1}{2} \sum_{w \in \mathsf{Supp}(X) \cup \mathsf{Supp}(Y)} |\Pr[X = w] - \Pr[Y = w]|$$

Proof. left as exercise. □

Minimal key size for statistically secure schemes. Unfortunately, statistical security doesn't enable us to get much shorter keys. In fact, if we use just one bit less the adversary can get an advantage of 1/4 (which we consider huge — remember that we were hoping for 10^{-100}!). We'll prove the following theorem:

Theorem 1. *Let (E, D) be a valid encryption with $E : \{0,1\}^n \times \{0,1\}^{n+1} \to \{0,1\}^*$. Then there exist plaintexts x_1, x_2 with $\Delta(x_1, x_2) > 0.1$.*

Proof. In the proof we'll use the following seemingly trivial observation: for a random variable Y, if $\mathbb{E}[Y] \leq \mu$ then $\Pr[Y \leq \mu] > 0$.

Let $x_1 = 0^{n+1}$ and let $S = \mathsf{Supp}(E_{U_n}(x_1))$. Note that $|S| \leq 2^n$.

Consider the following experiment: we choose a random message $x \leftarrow_{\mathrm{R}} \{0,1\}^{n+1}$ and define the following 2^n random variables: for every k, $T_k(x) = 1$ if $E_k(x) \in S$ and 0 otherwise.

For every k, $E_k(\cdot)$ is one to one and hence $\Pr[T_k = 1] \leq 1/2$. This means that $\mathbb{E}[T_k] \leq 1/2$.

Define $T = \sum_{k \in \{0,1\}^n} T_k$. Then

$$\mathbb{E}[T] = \mathbb{E}[\sum_k T_k] = \sum_k \mathbb{E}[T_k] \leq 2^n/2$$

This means that the probability that $\Pr[T \leq 2^n/2] > 0$ or in other words, there exists x such that $\sum_k T_k(x) \leq 2^n/2$. This means that for such x, at most half of the keys k satisfy $E_k(x) \in S$, or equivalently $\Pr[E_{U_n}(x) \in S] \leq 1/2$. Since $\Pr[E_{U_n}(0^{n+1}) \in S] = 1$ we get that

$$\Delta\left(E_{U_n}(0^{n+1}), E_{U_n}(x)\right) \geq 1/2$$

This technique — proving the existence of an object with a particular property by proving that the probability the property is satisfied is positive — is called the *probabilistic method*. There's a beautiful book about it with this name by Alon and Spencer. □

Computational Security Unfortunately, we saw that statistical security does not allow us to really break the impossibility result.

We now turn to a closer examination of that impossibility result. In particular, in real life people *are* using encryption schemes with keys shorter than the message size to encrypt all sort of sensitive information including credit card numbers. Could we use the proof of the impossibility result to *break* these schemes and gain notoriety and fortune?

Indeed, the proof of the impossibility result does in fact give a *algorithm* to break any encryption scheme. It's even quite simple (10 lines of C code).

The only problem is that if the key is of size n, then this 10 line C program will run in time roughly 2^n. This is going to take quite a long time even for n that is not too large.

Consider a key that is $1KB$ long (note that memory cards for digital camera typically have at least $128,000KB$). Even if we take Moore's law to its limit, and assume that we have placed on any atom in the observable universe a super-computer operating at the speed of light, we would still not be able to run 2^{1000} operations before the sun collapses. It's a safe bet that any credit cards we manage to steal will be expired by then...

This raises the idea of designing encryption that are unbreakable within any reasonable time.

Computationally Secure Encryption The main problem we face is that, while the particular C program arising from that proof runs in exponential time, we don't have any guarantee that there is not another program that is actually efficient. In fact, we already saw this is the case for the substitution cipher, where the number of possibilities for the key is huge but still we can break the scheme efficiently.

Another problem is that we want a precise mathematical definition. That is, the previous perfect secrecy definition was a precise statement about the functions (E, D) that can be formulated and proven to hold using the tools of mathematics. We don't want a vague definition such as "breaking E is very hard" since we can't work with such a definition.

This means that we need to give a precise mathematical formulation to statements such as "the problem P can not be solved in reasonable time". However, this arises the question of *how* do we model the adversary's resources. The adversary may use an IBM, Apple or Unix system, she may use a network of connected computers, she may use a super computer, or a special purpose computer she constructed just for this task, perhaps not made out of silicon but maybe out of analog or biological components, she can also use a mixture of computer and human intelligence, using say particularly gifted mathematicians to help break our encryption.

Can we give a mathematically precise definition that implies that a computational problem cannot be solved in say T years no matter what mixture of these and other resources are used?

It turns out the answer is most likely "Yes". To do so, we need to give a mathematical model that captures the notion of computation in all its forms. The main model we'll use will be the model of *Boolean circuits*.[1]

Boolean circuits A Boolean circuit is best explained with a picture (see handout for detailed description). The important parameter with a Boolean circuit is its *size* which is the number of vertices (e.g., gates) it contains.

Two basic facts about Boolean circuits:

1. Every function can be computed by a sufficiently large circuit:

   Theorem 2. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be some function. Then, there exists a circuit C of size at most $100mn \cdot 2^n$ that computes $f(\cdot)$.*

   Proof. We'll construct m different circuits each of size at most $100n \cdot 2^n$ for each of the outputs of $f(\cdot)$ and just concatenate them together (thus increasing the size by a factor of m). This means that without loss of generality we can think of $f(\cdot)$ as having a single output. Let S be the set of $s \in \{0,1\}^n$ such that this output is 1. Then,

   $$f(x) = \bigvee_{s \in S} (x \stackrel{?}{=} s)$$

[1] We note that the scientific community is still studying whether or not this model of Boolean circuits does bound all that can be done efficiently in the physical universe. Although it seems that it captures all mechanical and biological devices that *currently* exist, a fascinating challenge is posed to this model by *quantum computers*. These are hypothetical computing devices that may be built in the future and whose computing power relative to Boolean circuits is still very much an open question (see Scott Aaronson's thesis for more on this). However, for almost all the material of this course the choice of model (e.g., Boolean circuits or Quantum circuits) does not matter much, and so this debate does not effect us greatly.

Consider the function $g_s : \{0,1\}^n \to \{0,1\}$ where $g_s(x) = 1$ iff $x = s$. This function can be implemented by a $4n$ sized circuit (it is equal to the AND of x_i for the i's where $s_i = 1$ and $\neg x_i$ for the i's where $s_i = 0$). Therefore, since $|S| \leq 2^n$, we get that a single bit output $f(\cdot)$ can be implemented by a circuit of size at most $4n \cdot 2^n$. □

2. Some functions actually require a circuit of exponential size.

Theorem 3. *For every (sufficiently large) n there exists a function $f : \{0,1\}^n \to \{0,1\}$ that can not be computed by a circuit of size $2^{0.9n}$.*

Proof. Since a circuit of size T is a graph of in-degree $\leq T$ and labels coming out from a set of size $\leq 3T$, it can be described (using an adjacency list representation) using less than $100T \log T$ bits.
It follows that the number of such circuits is at most $2^{100T \log T}$. For $T = 2^{0.9n}$ this is equal to $M = 2^{100n \cdot 2^{0.9n}}$ and so circuits of that size can compute at most M functions. However, the number of functions $f : \{0,1\}^n \to \{0,1\}$ is 2^{2^n} which for sufficiently large n is much bigger than M. It follows that most of the functions from $\{0,1\}^n$ to $\{0,1\}$ can not be computed by circuits of size $2^{0.9n}$. □

Infinite functions We'll sometimes want to talk about functions defined on infinite domains (e.g., $\{0,1\}^*$). There are two ways to model that such a function is efficiently computable:

Notation: f_n = restriction of $f(\cdot)$ to $\{0,1\}^n$.

The class $\mathbf{P_{/poly}}$ We say that $f : \{0,1\}^* \to \{0,1\}^*$ is in $\mathbf{P_{/poly}}$ if there's a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every n, f_n is computable by a $p(n)$-sized circuit.

The class P If $f(\cdot)$ is in $\mathbf{P_{/poly}}$ then this does not mean that there is a *single* efficient algorithm to compute $f(\cdot)$ but rather that there is a different algorithm for each input length. If there is a actually a single algorithm (for example, there is a polynomial-time C program that can generate each of these circuits) then we say that $f \in \mathbf{P}$.

There is a way to formalize this notion using Turing machines (see handout).

Note that $\mathbf{P} \subsetneq \mathbf{P_{/poly}}$, because $\mathbf{P_{/poly}}$ contains *all* the unary functions — functions that depend only on the input *length* and not its contents, and it is not hard to find such functions that don't have a single algorithm.[2]

The Class NP: NP captures the problems whose solution can be efficiently *verified*. A Boolean function $f : \{0,1\}^* \to \{0,1\}$ is in **NP** if there's an efficient algorithm V such that $f(x) = 1$ iff $\exists_y$ such that $V(x,y) = 1$. (See handout for a more formal definition.)

Examples of problems in NP: Planarity, Primality, Traveling Sales Person, Circuit Satisfiability, 3SAT, Maximum Independent Set.

NP-completeness Circuit satisfiability has an interesting property - if there's a polynomial algorithm for it, then there is a polynomial algorithm for *every* problem in **NP**.

Proof by reduction Such results are proven by reduction: assume that there is an algorithm A for CSAT, and let V be the verification algorithm for the problem. Construct a circuit C where $C(y) = 1$ iff $V(x,y) = 1$. Then, there's a satisfying assignment for C iff $\exists y$ such that $V(x,y) = 1$.

[2]For people familiar with cardinalities, note that the number of such functions is $2^{|\mathbb{N}|} = \aleph$ where the number of possible algorithms is only $\aleph_0$.

Other NP complete problems. By a chain of reduction we can establish that other problems are **NP**-complete. $\mathsf{CSAT} \rightarrow \mathsf{3SAT} \rightarrow \mathsf{INDSET}$.