ICS 180: Introduction to Cryptography April 8, 2004

Lecture 2: Computational Notion of Security

Lecturer: Stanislaw Jarecki

1 LECTURE SUMMARY

We introduce the notion of *computational* security, in contrast to the *perfect* security of last lecture, which was an information-theoretic, rather than computational notion. We review the notions of algorithm running time, assymptotic notation, and polynomial time algorithms. We introduce notions of *efficient* algorithms, *negligible* probabilities, and *adversarial advantage* in attacking a cryptosystem. We exemplify these notions with an example of computational security definition for (private-key) encryption, namely the *indistinguishability* of encryption.

2 Computational Notion of Secrecy for Encryption

In the last lecture we saw that the *perfect secrecy* definition for encryption can be satisfied only by ciphers whose keys are as long as the messages they encrypt. With the computational notion of secrecy we try to preserve the spirit of Shannon's perfect secrecy notion, but we relax it in two fundamental ways, which in the end allows us to actually satisfy this notion with some practical algorithms.

Recall that we called a cipher *perfectly secret* if for every pair of messages $m_0, m_1 \in \mathcal{M}$ and ciphertext c we have

$$\underset{k \leftarrow \mathcal{K}}{Prob}[Enc(k, m_0) = c] = \underset{k \leftarrow \mathcal{K}}{Prob}[Enc(k, m_1) = c]$$

In other words, if encryptions of m_0 *look exactly the same* as encryptions of m_1.

The computational version of this definition will say that for every messages m_0, m_1, for every *efficient algorithms* A, the adversarial advantage of A in distinguishing encryptions of m_0 from encryptions of m_1 is *negligibly small*, i.e.

$$| \underset{k \leftarrow \mathcal{K}}{Prob}[A(m_0, m_1, c) = 1 \mid c \leftarrow Enc(k, m_0)] - \underset{k \leftarrow \mathcal{K}}{Prob}[A(m_0, m_1, c) = 1 \mid c \leftarrow Enc(k, m_1)] | < \epsilon$$

for some negligibly small ϵ.

What do we mean by "efficient algorithm" and "negligibly small" factor? By "efficient algorithm" we mean *probabilistic polynomial time*. The next section reviews what probabilistic polynomial time algorithms are, and recalls some associated notation.

3 Review of Algorithm Analysis

First some **notation** (you can skip this now and only refer to it as needed):

- By $\mathbb{Z}$ we denote the set of integers, by $\mathbb{N}$ the naturals (i.e. positive integers plus zero), and by $\mathbb{R}$ the set of real numbers. Note that $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R}$.
- If D, R are sets, we denote functions f from domain D to range R as $f : D \to R$. For example, $f : \mathbb{N} \to [0, 1]$ is a function from naturals to the interval between 0 and 1.
- We'll denote assignment of a random variable according to some distribution with the $\leftarrow$ sign. For example, by $b \leftarrow \{0, 1\}$ we denote the event of picking a random element in set $\{0, 1\}$ and assigning its value to variable b. In other words, in this case b is a random bit.
- By $Prob[A \mid B]$ we denote the conditional probability that event A happens given that event B happens. For example, $Prob[b = 1 \mid b \leftarrow \{0, 1\}] = 1/2$, because that's the probability that a random bit is equal to one.
- By $\{a \mid \mathsf{cond}(a)\}$ we denote a set of all elements a which meet condition $\mathsf{cond}(a)$. For example, $\{a \mid \exists_{i \in \mathbb{Z}}\ a = 2i\}$ is a set of all even numbers.
- By $[a_1, a_2, ..., a_n]$ we'll denote a sequence of elements $a_1, a_2, ..., a_n$. Especially if a_i's are binary strings or bits, sometimes we'll equate such sequence with the string which is a concatenation of these elements, denoted $a_1|a_2|...|a_n$ or $[a_1|a_2|...|a_n]$.
- If a is a binary string, say of length n, then by a_i, for $i \in [1, .., n]$ we'll denote the i-th bit of a. This way we can rewrite a as $[a_1|a_2|...|a_n]$. By $a_{[i..j]}$, for $1 \leq i < j \leq n$, we'll denote the substring $[a_i|a_{i+1}|...|a_j]$ of a.
- If A is a set, A^n stands for set of sequences, each n-element long, of which every element is a member of set A, i.e. $A^n = \{[a_1|a_2|...|a_n] \mid \forall_i\ a_i \in A\}$. For example, $\{0, 1\}^n$ is a set of all binary strings of length n. We'll use similar notaton for sequences formed by repeating some element, for example 1^n denotes a sequence of n ones.
- By A^* we denote the union $A^* = \cup_{i \in \mathbb{N}}\ A^i$. For example, $\{0, 1\}^*$ is a string of all binary strings, including the empty string, sometimes denoted ϵ.
- If s is a binary string then by $|s|$ we mean the *length* of s, e.g. $|010| = 3$. If S is a set then by $|S|$ we mean the *cardinality* of S, i.e. the number of elements in S, e.g. $|\{010, 011\}| = 2$. Using this notation, note that $|\{0, 1\}^n| = n$ and for every $s \in \{0, 1\}^n$ we have $|s| = n$.

Now some **definitions**:

Definition 1 [computing a function] *An algorithm A* computes a function $f : \{0, 1\}^* \to \{0, 1\}^*$ *if on any input $x \in \{0, 1\}$, it outputs $f(x)$.* (We assume that inputs and outputs are binary strings.)

Definition 2 [probabilistic algorithm] *An algorithm is* probabilistic *if it sometimes casts a random coin and decides what to do next on the basis of it.*

Definition 3 [algorithm running time] *Algorithm A* runs in time $T_A : \mathbb{N} \to \mathbb{N}$ *if A runs for at most $T_A(n)$ steps when executed on any input x of length n.*

Definition 4 [big O notation] *If $f, g : \mathbb{N} \to \mathbb{R}$ then we say that g* upper-bounds *f, and we denote it $f = O(g)$, if there exists constant $c > 0$ s.t. $f(n) < cg(n)$ for all $n > 0$.*

Definition 5 [big Θ notation] *If $f, g : \mathbb{N} \to \mathbb{R}$ then we say that g* lower-bounds *f, and we denote it $f = \Theta(g)$, if there exists constant $c > 0$ s.t. $f(n) > cg(n)$ for all $n > 0$.*

Definition 6 [polynomial time algorithm] *We call algorithm A* polynomial time *if there exists a polynomial $p(\cdot)$ s.t. the running time function T_A of A is upper bounded by p, i.e. if $T_A = O(p)$. In other words, if there exists $c > 0$ s.t. $T_A(n) < c * p(n)$ for all n.*

An equivalent definition is that A is poly-time if there exists constant d s.t. $T_A(n) = O(n^d)$.

Definition 7 [efficient algorithm] *We'll call an algorithm* efficient *if it is probabilistic polynomial-time, denoted "PPT".*

4 Negligible Probability

Second, what do we mean by "negligible factor", i.e. how small do we need ϵ to be? We want this probability be a fast decreasing function of the security parameter. The function should be decreasing fast enough that if we wanted ϵ to be smaller than any particular very small amount we want, for example $1/2^{80}$, we would like it to be true for some reasonable security parameters τ that $\epsilon(\tau) < 1/2^{80}$. A standard way of capturing such notion of *reasonable* security parameter is to require that $\epsilon(\tau)$ is smaller than any inverse polynomial. For example, $\epsilon(\tau) = 1/2^{\tau}$ is such a function, in which case $\epsilon(\tau) < 1/2^{80}$ for $\tau \geq 80$. This gives rise to the following definition of a negligible function:

Definition 8 [negligible function] *We call a function $\epsilon\mathbb{N} \to [0, 1]$* negligible *if* for every *constant c there exists some point n_0 s.t. after that point, i.e. for all $n \geq n_0$, we have*

$$\epsilon(n) < \frac{1}{n^c}$$

For example, function $\epsilon(n) = \frac{1}{2^n}$ is *negligible* because after some point n_0, every $n > n_0$ satisfies $\frac{n}{\log n} > c$, which implies that $2^n > n^c$, and hence that $\frac{1}{2^n} < \frac{1}{n^c}$.

Where did $1/2^{-80}$ come from? Because if a single experiment is succesful with probability p then you need to repeat it about p times to have reasonable probability, about $1/2$, of really succeeding in the experiment. Therefore, if it's known that the best reasonably fast algorithm succeeds with probability at most $p = 1/2^{80}$ against some cryptosystem then you'd need to repeat it $1/p = 2^{80}$ times to break the cryptosystem with probablity $1/2$, and if you repeat it just a little more ($1/p * \log(1/p)$, which in this case is $2^{80} * 80 \approx 2^{80+6}$) then only then you will break the cryptosystem with probability almost 1. (Try to work out the math for yourself!) And, finally, algorithms that run for 2^{80} steps indeed seem infeasible, given the current mips/price ratio, for about the next ten years.

5 Relation of Computational Secrecy to Perfect Secrecy

From now on, by a (private-key) encryption scheme Σ we call a triple of *efficient* algorithms $(KGen, Enc, Dec)$ where

- $KGen$ picks a key on input of the *security parameter* τ encoded in unary, i.e. on τ-long string of 1's. (The reason we do that is to formally allow polynomial time in the security parameter to pick the key.)
- The encryption and decryption algorithms are such that for all $k \leftarrow KGen(1^\tau)$, for all $m \in \{0,1\}^*$, if $c \leftarrow Enc(k,m)$ then $m \leftarrow Dec(k,c)$.

Note that unlike in the previous lecture, we don't define the message space (also, the key space is impilicitely defined as the outputs of $KGen$). The reason is that we want the encryption to work on all binary strings, i.e. from now on $\mathcal{M} = \{0,1\}^*$.

Let's look at the definition of *perfectly secret* cipher again. It required that for every pair of messages m_0, m_1 and ciphertext c we have

$$\underset{k\leftarrow\mathcal{K}}{Prob}[Enc(k,m_0) = c] = \underset{k\leftarrow\mathcal{K}}{Prob}[Enc(k,m_1) = c]$$

Another way to state it is that for every m_0, m_1, the following two random distributions are the same:

$$\{c\}_{k\leftarrow\mathcal{K},c\leftarrow Enc(k,m_0)} \equiv \{c\}_{k\leftarrow\mathcal{K},c\leftarrow Enc(k,m_1)}$$

i.e. if the distibution of ciphertexts resulting from picking a random key and encrypting m_0 is the same as the distribution of ciphertexts resulting from picking a random key and encrypting m_1.

Note that adding the messages m_0, m_1 themselves to the view does not change anything. Also, since instead of just picking keys at random from some distribution $\mathcal{K}$ the keys are generated by an efficient algorithm $KGen(1^\tau)$, we rewrite the above requirement as

$$\{(m_0,m_1,c)\}_{k\leftarrow KGen(1^\tau),c\leftarrow Enc(k,m_0)} \equiv \{(m_0,m_1,c)\}_{k\leftarrow KGen(1^\tau),c\leftarrow Enc(k,m_1)}$$

Now we can see that the **first computational relaxation** of the above is that rather than requiring that ciphertexts of m_0 and ciphertexts of m_1 are distributed *identicially*, we can only require that *no efficient algorithm* can tell a difference between these two types of ciphertexts, i.e. that for every PPT A, the distributions of *outputs of* A are the same in the two cases:

$$\{A(m_0,m_1,c)\}_{k\leftarrow KGen(1^\tau),c\leftarrow Enc(k,m_0)} \equiv \{A(m_0,m_1,c)\}_{k\leftarrow KGen(1^\tau),c\leftarrow Enc(k,m_1)}$$

Since we can assume that an attacker algorithm A outputs 1 if it thinks that ciphertext c is more likely to be an encryption of m_1 and 0 if it thinks that c looks more like an encryption of m_0, we can rewrite the above requirement as follows:

$$\underset{\substack{k\leftarrow KGen(1^\tau)\\ c\leftarrow Enc(k,m_0)}}{Prob}[A(m_0,m_1,c) = 1] \quad = \quad \underset{\substack{k\leftarrow KGen(1^\tau)\\ c\leftarrow Enc(k,m_1)}}{Prob}[A(m_0,m_1,c) = 1]$$

Note that the above equation compares the probability that A returns a "false posititive" judgment by outputing 1 on encryption of m_0, with the probability that A returns a "true positive" judgment by outputing 1 on encryption of m_1. If the two probabilities are the same then indeed A is useless as a test that is supposed to determine whether c is an encryption of m_0 rather than m_1!

Stating things in this way leads us to the **second relaxation**: Namely, instead of requiring that the false positives and true positives have the *same* probability, we can require that the difference between these two probabilies is a *negligible function* $\mathsf{negl} : \mathbb{N} \to [0,1]$ of the security paramater.

This leads to the following definition:

Definition 9 (indistinguishable encryption) *We call an encryption scheme* Σ indistinguishable *if for all PPT algorithms* A *and all polynomials* $p(\cdot)$ *there exists a negligible function* $\epsilon(\cdot)$ *s.t. for all security parameters* τ*, for all message pairs* m_0, m_1 *s.t.* $|m_0| = |m_1| < p(\tau)$*, the* adversarial advantage Adv_A *defined as follows is smaller than* $\epsilon(\tau)$*:*

$$Adv_A = \left| \underset{\substack{k \leftarrow KGen(1^\tau) \\ c \leftarrow Enc(k, m_0)}}{Prob} [A(m_0, m_1, c) = 1] - \underset{\substack{k \leftarrow KGen(1^\tau) \\ c \leftarrow Enc(k, m_1)}}{Prob} [A(m_0, m_1, c) = 1] \right| \leq \epsilon(\tau)$$

Some remarks:

(1) Why are we requiring that the two messages m_0, m_q be of equal length? This means that we give up on asking the encryption scheme to hide the message length. In other words, we allow the adversary to tell, given a ciphertext, how long the plaintext message was. Therefore, if $|m_0|$ and $|m_1|$ were allowed to be different in this definition, the adversary would trivially "break" reasonable encryption schemes by easily distinguishing encryptions of some very short m_0 from encryptions of some much longer m_1. And why do we not care to hide the message length? In fact some applications might care, in which case you have to strengthen the security requirement and make sure your encryption is message-length hiding. However, the only way to do that would be to create large overheads for short messages. And since most of the time people don't require this type of security, they also don't want to pay the unnecessary price of the overhead in ciphertext length.

(2) Why is the message length bounded by a polynomial? Because otherwise the definition wouldn't make sense because the definition would then end up making an unreasonable requirement that encryption is secure against even adversaries running in more than polynomial time. To enforce that A runs in time polynomial in the security parameter, we have to limit the size of A's inputs themselves to size polynomial in the security parameter. Otherwise, for example if A's inputs were allowed to be of size 2^τ, then A would be given also at least 2^τ time to run, since A is PPT, i.e. it runs in in time polynomial *in the length of its input*. Since such amount of time would usually allow A to break the cryptosystem by just finding the private key (in many cryptosystem the security parameter τ is simply the length of the private key), we cannot hope to fend off a distinguishing attack A which is allowed to work for this amount of time.

Also, if messages are sent back and forth by realistic communication components (computers, PDAs, etc), then the total length of the messages they send must be, by definition, computable by an efficient machine, and we assume that this is polynomial in the security parameter, although this polynomial might be pretty large.

6 Consequences of Indistinguishability

Notice that there often will exist some efficient algorithm A which will have a negligible but non-zero advantage Adv_A in distinguishing encryptions of m_0, m_1. For example, given (m_0, m_1, c), A could just guess the key k output by $KGen(1^\tau)$ by picking a random key k', running the decryption algorithm $m' = Dec(k', c)$, and outputing "one" if $m = m_1$, and "zero" otherwise. Note that since the key generation algorithm must be itself efficient, the keys it outputs can be no longer than $p(\tau)$ for some polynomial $p(\cdot)$. Therefore, A which just guesses k at random will be succesful with probability at least $1/2^{p(\tau)}$, in which case Adv_A of definition 9 will be at least $1/2^{p(\tau)}$ too. However, since $p(\cdot)$ is a polynomial, this advantage is a negligible function of τ.

However, what happens if one repeats some efficient attack algorithm A for polynomialy-many times (say $p(\tau)$ times) and tries to accumulate the votes given by each run of A (note that each time A is run, we can assume that it votes either "zero" or "one", its decision as to whether c is more likely an encryption of m_0 or m_1)? Let's call this new attack A'. If A takes polynomial time then so does A', because $Time_{A'} = p(\tau) * Time_A$. However, assuming that Adv_A is non-zero but negligible, can the advantage of A' be more than negligible? In other words, can we boost the *negligibly* successful attack into a *non-negligibly* successful one by mere repetition? The answer is "no!", but we'll skip the argument because it's too technical.

However, we can gain some insight into it by asking a related question: If the advantage of A is *not* negligible but pretty small, we can show that this type of repeated run of A can boost this small advantage into an overwhelming one. We'll see that if on other hand Adv_A is negligible then this boosting method fails.

Fact 10 *If an attack A has a small but non negligible advantage in breaking indistinguishability of encryption, there is an attack A' which runs A polynomialy many times (and takes the majority vote) which has an* overwhelming *advantage in breaking the indistinguishability of the encryption scheme.*

By "overwhelming", we mean so great than $1 - 2^{-\tau}$, i.e. "negligibly close" to 1 (because function $2^{-\tau}$ is negligible).

Proof: We'll use Chernoff bound to show this. The Chernoff bound tells you that if you take n independent samples of a random variable V_i whose expected value is μ then the probability that the average taken of these n runs, i.e. V/n where $V = \sum_{i=1}^{n} V_i$, is farther than ϵ from μ, is bounded by inverse exponential in ϵ^2 and n. In particular,

$$Prob[V/n > \mu + \epsilon] \leq 2^{-2\epsilon^2 n}$$

Assume, for simplicity, that the probability that A outputs "one" on encryption of m_1 is $1/2 + \delta$ while the probability that it outputs "one" on encryption of m_0 is $1/2 - \delta$. Since $Adv_A = (1/2 + \delta) - (1/2 - \delta) = 2\delta$ is negligible, it must be that δ is a negligible function of τ. Define V_i as the output of i'th run of A on encryption of m_0. The value of each V_i is either 1 or 0, and its expected value is $\mu = 1/2 - \delta$.

The probability that A' outputs 1 on encryption of m_0 is the probability that at least $n/2$ of the runs of A output 1 on this encryption, i.e. the probability that $\sum V_i > n/2$, i.e. the probability that $V/n > 1/2 = \mu + \delta$. By the Chernoff bound, this probability is at most $2^{-2\delta^2 n}$.

In a similar way we can argue that the probability that A' outputs 0 on encryption of m_1 is less than $2^{-2\delta^2 n}$, and hence the probability that A' outputs 1 on encryption of m_1 is at least $1 - 2^{-2\delta^2 n}$.

Therefore the advantage of A' is at least

$$Adv_{A'} \geq (1 - 2^{-2\delta^2 n}) - 2^{-2\delta^2 n} = 1 - 2^{1-2\delta^2 n}$$

And therefore $Adv_{A'} \geq 1 - 2^{-\tau}$ if $2\delta^2 n > \tau + 1$, which holds if $n > \frac{\tau+1}{2\delta^2}$. If δ is not negligible then $\delta(\tau) > 1/p(\tau)$ for some polynomial p and all large enough τ's, in which case the equation holds for all large enough τ's as long as $n > \frac{1}{2}(\tau + 1)p^2(\tau)$, which is polynomial in τ. □

Note that the same argument fails when δ is negligible in τ, for example $\delta = 2^{-\tau}$. In this case we'd have to make $n > \frac{1}{2}(\tau + 1)2^{2\tau}$ which would be exponential in τ and thus A' would be inefficient.