CSE 5852: Lecture 7

Benjamin Fuller

October 4, 2016

1 Last Class

Last class we talked about some concepts from modular arithmetic. We defined two groups that will be important for the rest of the class $\mathbb{Z}_p$ (with the addition operation) and $\mathbb{Z}_p^*$ (with the multiplication operation). We then showed that $a \cdot c + b \mod p$ is a good strongly universal hash function.

Finally, we concluded by talking about what changes need to be made to our definition in order to overcome the barriers of the one-time pad. We came up with two changes:

1. We need to allow the adversary to have non zero probability of success.
2. We need to restrict to adversaries that run in a reasonable amount of time. For example, they shouldn't be able to exhaust the key space.

2 Defining computational security

In this class we will begin to define what it means for computational security. At the beginning of the class we talked about who we want our encryption schemes to be "private" against, this list included major governments. Lets list some things that we may want to provide security against:

- The set of all C programs that run in less than a year
- The set of all programs runnable in memory held by my computer
- The set of all programs computable in EC2.
- The set of all programs computable by an alien race that is really good at running programs.
- The set of all programs runnable before the heat death of the universe.

How do we have a good sense of these things? Even the top most thing on this list is difficult to judge. How can we tell a priori how long a program is going to run? We can't this is an undecidable problem. (Who remembers what an undecidable problem is?)

2.1 What is our goal?

With the set of programs together we are trying to simultaneously achieve too goals, we want out scheme to be secure against all current attackers and don't want future technologies or algorithms to be able to obviate our system. Ideally, we could provide the second thing unconditionally that the problem is just hard for computers, but that is unlikely given our current understanding of computer science.

Two approaches to this problem: 1) concrete and 2) asymptotic. These two approaches attempt to balance the goals described above.

Concrete Security In the concrete approach we attempt to bound the maximal success probability of an adversary running in a particular time t. That is, we show that the scheme is (t, ϵ)-secure. Not that this automatically provides security for all adversaries that run in time $t' < t$. However, it says absolutely nothing about what happens for a $t' > t$. (We can show some weak bounds that slightly increasing t' does not help too much but morally this statement is correct.)

Suppose we have a private key encryption scheme with $t = 2^{60}$ and $\epsilon = 2^{-60}$. Standard desktop machines can execute approximately 10^{10} operations per second or 2^{33}. This would require 2^{30} seconds to gain an advantage of 2^{-60}. This value is 34 years which seems safe. However, super computers are able to compute approximately 10^{16} operations a second or 2^{53}. This computer would require only 128 seconds to break this scheme or approximately 2 minutes.

If instead we take $t = 2^{80}$, then the super computer would take 2^27 seconds or 4 years. However, this assumes that computing techniques stay constant throughout this period. Moore's law says that semiconductor density doubles every 18 months. This has traditionally lead to a comparable increase in computing power (which started as an increase in processor speed and has now moved to an increase in parallelism).

For truly sensitive data having it remain private for 4 years may not be sufficient. This may lead to us being conservative and building a scheme secure against $t = 2^{128}$. There's been 2^{58} seconds since the big bang. Notice however that 2^{-60} did not appear in any of these calculations. This is our probability of failure. If failure occurs once ever 2^{-30} seconds that is once every 100 years. Once every 2^{-60} seconds is once every 100 billion years.

Concrete security is very useful in practice. This is often what you want to know when deploying a scheme. However it is difficult to get a hold on. How do we measure the best attacks against the algorithm? How do we get an accurate understanding of the best computing technology? What kind of attacker are assuming? What about advances in computing technology? (Consider quantum computing.)

Asymptotic Security Any real deployment of a crypto protocol must use concrete security. It is important to tell adopters the best estimate for how long it will take to break the protocol based on the designers best understanding of the state of the art in computing.

A more relaxed approach is to say that as computing advances we have an algorithm that will work. So when the best computing can do 2^{30} operations a second we use one algorithm, when the best computing can do 2^{40} operations

a second we use a second algorithm and so on. The idea is that we should be able to select a new algorithm based on our understanding of current computing power. When it becomes time to actually deploy this algorithm we will select the right algorithm based on our current understanding of computing technology.

The approach proceeds as follows:

1. Rather than constructing a single algorithm, all of our algorithms take in a parameter n called the security parameter. This parameter specifies which algorithm should be used. Notionally as n increases the algorithm should be secure against adversaries running in longer time.

2. Rather than measuring adversary success as a single number we consider it as a function of n. Does the adversary become more successful as n increases? Less? This makes both ϵ and t become functions of n.

3. We say the system is secure if when A's running time is less than $t(n)$ their success probability is at most $\epsilon(n)$.

The natural question becomes what is the right bound to seek for $t(n)$ and $\epsilon(n)$. In the majority of cryptography we do the following (though I am not arguing this is the right approach):

1. Assume the adversary is a probabilistic Turing machine. That is, it is able to flip random coins but is not obliged to do so.

2. Consider security when the adversary runs in time polynomial in the security parameter. Note that this definition does not make sense in the concrete security setting, there's no such thing as a polynomial unless our running time is a function of something.

3. Argue that the adversary's success probability is smaller than any polynomial in the security parameter. Specifically we say that the adversary's advantage must be negligible (as defined below).

Definition 1. *A function $p : \mathbb{Z}^+ \to \mathbb{R}^+$ is a polynomial (bounded) function if there exists $k, N \in \mathbb{Z}^+$ such that for all $n > N$ if holds that $p(n) \leq n^k$.*[1][2]

Definition 2. *A function $f : \mathbb{Z}^+ \to \mathbb{R}^+$ is negligible function if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $f(n) < 1/p(n)$.*

What can a Turing Machine do? In this class we won't require the formal description of how a Turing machine or what computing power it has. Just as a quick reminder a Turing machine is a set of states along with a single tape that grows in a single direction. You can alternatively think of an computation that can be effectively performed on in your favorite programming language. The formalism of a Turing machine will not be important for most of the class.

[1] Here we are talking about a function that is bounded by a polynomial not an actual polynomial. For example $p(n) = \sin(n)$ would satisfy our definition but this function is not a polynomial. For the purposes of this problem set we are concerned with polynomial time. In this setting, we care that the function is bounded above by a polynomial. That is what this definition guarantees.

[2] This definition is equivalent to saying that $p(n) \leq cn^k$ for some constant $c > 0$. The constant c can be avoided by increasing k, so we remove it to simplify notation.

2.2 Modifying Shannon secrecy

Definition 3 (Shannon Secrecy)**.** *An encryption scheme* $(\mathcal{M}, K, \mathsf{Enc}, \mathsf{Dec})$ *satisfies* Shannon secrecy *if for every two messages* $m_1, m_2 \in \mathcal{M}$ *and for every ciphertext* c*:*

$$\Pr_{k \in K}[\mathsf{Enc}_k(m_1) = c] = \Pr_{k \in K}[\mathsf{Enc}_k(m_2) = c].$$

This definition is significantly easier to modify.

Definition 4 (Indistinguishable)**.** *An encryption scheme* $(\mathcal{M}, K, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable encryptions if for all* $\mathcal{A}$ *for every two messages* $m_1, m_2 \in \mathcal{M}$ *and for every ciphertext* c*:*

$$|\Pr_{k \in K}[\mathcal{A}(\mathsf{Enc}_k(m_1)) = 1] - \Pr_{k \in K}[\mathcal{A}(\mathsf{Enc}_k(m_2)) = 1]| < \epsilon.$$

Why is it important that $\mathcal{A}$ outputs 1 and not m_1 or m_2? This is because this guarantee should hold for all pairs of messages which are defined after the adversary. By limiting ourselves to an adversary that outputs two values we can effectively argue that they perform the same on the two messages.

References

[SWBH49] Claude E. Shannon, Warren Weaver, Richard E. Blahut, and Bruce Hajek. *The mathematical theory of communication*, volume 117. University of Illinois press Urbana, 1949.