CMPUT 675: Computational Complexity Theory **Winter 2019**

Lecture 16 (Mar 7): Cryptography I

Lecturer: Zachary Friggstad *Scribe: Jason Cannon*

16.1 Introduction

Although a number of cryptography systems are used in practice today, no system that uses "short keys" has been proven to be secure against adversaries with limited computing power (eg. polynomial-time adversaries). Let us consider RSA, a public key crypto-system that is only secure under the assumption that factoring is hard, *i.e.*, that given two large primes p and q, one can quickly compute the product pq but cannot easily uncover the factors p and q. Even if factoring is hard all the time, we might not know whether it is hard under the generation of these large random prime keys. While this does not seem to be the case, it still has not been proven. In fact, in the quantum setting, this problem is polynomial-time solvable!

Other common crypto-system examples suffer from similar problems. For example, Diffe-Hellman relies on the discrete logarithmic problem, which can be shown to be equivalent to factoring. We also have private key systems like the Advanced Encryption and Data Encryption Standards (AES and DES, respectively). None of the crypto-systems we use are provably secure, we just hope that they are! In fact, to highlight this point, DES has been cracked and required a "patch" to re-establish security (at least for now).

In this lecture, we will discuss the theory of cryptography and what we hope to achieve from secure systems.

16.1.1 One-Time Pad

In fact, we do know of one perfectly secure cryptography-system: the *one-time pad.* In this scheme, if we want to encrypt a message $x \in \{0,1\}^n$ we do so by choosing a random key $k \sim \{0,1\}^n$ and sending $x \oplus k$ (vector addition modulo 2). We can then decrypt the message to obtain x from $y = x \oplus k$ by computing $y \oplus k$. Consider that for any two languages of length n any ciphertext is equally likely. From an attacker's perspective, any ciphertext looks like a uniformly random string so they have no hope of discerning the original message x. The problem is that the receiver must have the same random sequence, so they must be able to securely share a key whose length is equal to the original message length. Perhaps the sender and receiver could secretly meet up beforehand, but perfect security is reliant on that initial exchange of the random bits and a single use of the given key.

16.2 Encryption Scheme and Perfect Secrecy

Our goal is to talk about encryption with short keys. We will place fairly strict requirements on what it means for our encryption scheme not to be broken.

Definition 1 (Encryption scheme) *An encryption scheme with length $K(n)$ keys is a pair of algorithms (E, D) such that $\forall x \in \{0,1\}^*, \forall k \in \{0,1\}^{K(|x|)}$*

$$D_k(E_k(x)) = x.$$

So the length of the key is a function of the length of the message. Notice that we should interpret this scheme as having two inputs (the key and the ciphertext itself) even if the parameters are not displayed in the usual comma-separated fashion. Intuitively, we are given a pair: an encryption algorithm E and a decryption algorithm D, such that for every key k and plaintext x when we encrypt x and then decrypt $E_k(x)$ with the key k we obtain the original plaintext x. We will now discuss our ultimate goal for encryption schemes: *perfect secrecy.*

Definition 2 (Perfect secrecy) *An encryption scheme has perfect secrecy if* $\forall m, \forall x, x' \in \{0,1\}^m, \forall y \in \{0,1\}^*$

$$\Pr_{k\sim\{0,1\}^{K(|x|)}}[E_k(x) = y] = \Pr_{k\sim\{0,1\}^{K(|x'|)}}[E_k(x') = y].$$

Intuitively, this is similar to what we observed with the one-time pad scheme. If we pick two keys perfectly at random, then the distributions over ciphertexts should be the same, and the output should look exactly like a random string. So, for every possible message length l, pair of messages m_1 and m_2, and ciphertext y, then the probability of encrypting the first string m_1 and getting y should be the same as the probability of encrypting the second string m_2 and getting the same y. For any two messages of the same length, the distributions over outputs using random keys is the same. No information can be gleaned from the original message, and the encryption is perfectly secure.

16.3 Computational Security

Let us now consider the next result, which shows us that if $\mathbf{P} = \mathbf{NP}$ then we simply cannot achieve perfect secrecy with efficient encryption and decryption algorithms that use keys whose length is shorter than the message length *even by one bit.* In fact, if we want decent secrecy we *must* assume that $\mathbf{P} \neq \mathbf{NP}$. Intuitively, if $\mathbf{P} = \mathbf{NP}$, many assumptions that currently allow for perfect security would crumble. For example, we could factor large integers and break RSA, or solve the discrete logarithmic problem to crack Diffie-Helman. Furthermore, given some ciphertext generated by some polynomial-time encryption algorithm, a key could be non-deterministically guessed and used to decrypt the ciphertext.

However, it follows that one-time pad is still perfectly secure even if $\mathbf{P} = \mathbf{NP}$, but only given that the key is at least as long as the message. In this section, our goal is to show that if you use a key whose length is even single bit less than the message length and $\mathbf{P} = \mathbf{NP}$ then perfect secrecy is completely destroyed.

Theorem 1 *If* $\mathbf{P} = \mathbf{NP}$*, for all polynomial-time encryption schemes (both E and D run in polynomial-time) there is a polynomial time Turing Machine A such that* $\forall m$ *such that* $K(m) < m, \forall x_0 \in \{0,1\}^m$ *there is some* $x_1 \in \{0,1\}^n$ *such that*

$$\Pr_k[A(x_0, E_k(x_0)) = 1] = 1 \text{ and } \Pr_k[A(x_0, E_k(x_1)) = 0] \geq \frac{1}{2}$$

Intuitively, under $\mathbf{P} = \mathbf{NP}$ if we encrypt with shorter keys then for any plaintext message x_0 we are interested in, there is another plaintext message x_1 such that an *efficient* algorithm is able to determine with decent probability between the two by just looking at their ciphertexts if they are encrypted with random keys. Furthermore, as the key length decreases, this probability of discerning between x_0 and x_1 will driven very close to 1 (as the proof reveals).

Proof. Let $S = \left\{E_k(x_0) : k \in \{0,1\}^{K(m)}\right\}$ be all possible ciphertexts of x_0. For any key $k \in \{0,1\}^{K(m)}$, let $T_k = \{E_k(x) : x \in \{0,1\}^m\}$, that is, the possible ciphertexts you can see by encrypting messages of length m

using this given key k. Observe that $|T_k| = 2^m$ because E_k is injective implying that for a fixed key, two plain-text messages cannot encrypt to the same ciphertext (or else we could not uniquely decrypt our ciphertext). Therefore, $|T_k - S| \geq 2^m - 2^{K(m)} \geq 2^{m-1}$ (as $K(m) < m$). Thus, consider the probability of picking a random x and a random key k of appropriate size, we have that $\Pr\limits_{x,k}[E_k(x) \notin S] \geq \frac{1}{2} \implies \exists x_1$ s.t. $\Pr\limits_{k}[E_k(x_1) \notin S)] \geq \frac{1}{2}$. Hence, the following algorithm A has the desired behaviour:

Algorithm A

Input: A pair x_0, y where x_0 is a message of interest and y is the ciphertext of some message of length $|x_0|$.
Output: Output 1 if $\exists k$ such that $E_k(x_0) = y$ (i.e. $y \in S$), 0 otherwise.

This is an efficient algorithm because determining the existence of a key is in **NP** and we assumed **P** = **NP**. ■

16.3.1 One way functions

Let us now introduce a simple definition that will be useful for simplifying notation in future sections.

Definition 3 (Negligible functions) *A function $\varepsilon : \mathbb{N} \to [0,1]$ is called negligible if $\varepsilon(n) \in O\left(\frac{1}{n^c}\right)$ for all $c \geq 0$.*

That is, $\varepsilon()$ vanishes faster than any inverse polynomial. Recall in a previous lecture when we stated a seemingly weaker definition of **BPP**, denoted $\mathbf{BPP}^1$, where we were content with succeeding with probability $\frac{1}{2} + \frac{1}{\text{poly}(n)}$ of the time. We then showed that $\mathbf{BPP} = \mathbf{BPP}^1$ by increasing the probability to be exponentially close to 1. However, with $\varepsilon()$, if we begin with $\frac{1}{2} + \varepsilon(n)$ then it follows that the *Chernoff Bounds* will not be able to boost the probability beyond a negligible quantity within a polynomial number of iterations.

We will now introduce the notion of a *one-way function*—a function that is easy to compute, but hard to invert for a polynomial-time algorithm on average.

Definition 4 (One-way functions) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function if it can be computed in polynomial-time and $\forall$ polynomial-time PTM A, $\forall m$*

$$\Pr_{x \sim \{0,1\}^m}[f(A(f(x)) = f(x)] \leq \varepsilon(m)$$

for some negligible $\varepsilon()$. Or written differently

$$\Pr_x[A(f(x)) = y \text{ s.t. } f(y) = f(x)] \leq \varepsilon(m).$$

That is, for any input size, if you provide random input into the function and ask A to construct an inverse, then there no efficient algorithm can do this with non-negligible probability.

Conjecture 1 *There exists a one-way function.*

This conjecture seems stronger than $\mathbf{P} \neq \mathbf{NP}$. If $\mathbf{P} = \mathbf{NP}$ then we know that one-way functions do not exist. Intuitively, given some output we can non-deterministically guess its input, because we assume our functions are polynomial-time computable: more precisely we can use a previous result in the course to show that if **P** = **NP** then we can, in fact, efficiently construct the preimage of some string that is the output of a one-way function. But it is not clear that $\mathbf{P} \neq \mathbf{NP}$ implies one-way functions exist.

Let us introduce one more concept before we can explore the main theorem of the lecture.

16.3.2 Computationally Secure

The proof of Theorem 1 can be easily adapted to show that any encryption scheme using keys that are shorter than the message length is not perfectly secret even if $\mathbf{P} \neq \mathbf{NP}$. But the algorithm in the proof would have to solve problems in $\mathbf{NP}$. Can we get security against adversaries with bounds on their computational power?

Definition 5 (Computationally secure) *An encryption scheme (E, D) is computationally secure if E, D runs in polynomial-time and $\forall m, \forall$ polynomial-time PTM*

$$\Pr_{\substack{x \sim \{0,1\}^m \\ k \sim \{0,1\}^{K(m)}}} [A(m, E_k(x)) = (i, x_i)] \leq \frac{1}{2} + \varepsilon(m)$$

Intuitively, recall that *perfect secrecy* means that the distribution over encrypted outputs is the same. An encryption scheme is *computationally secure* if no efficient algorithm can learn a single bit of the message with high probability. That is, the probability over a random key and random message that A can output a single correct bit of the original plaintext is basically $1/2$, differing only by a negligible quantity. The algorithm might as well be tossing coins.

We are now ready to outline the main theorem. Ultimately, our goal is to come up with a computationally secure crypto-system by assuming the existence of one-way functions. Although we will not be able to completely prove this, we will prove a result that is similar enough so that the necessary concepts can be understood.

Theorem 2 *If one-way functions exist, then $\forall c \geq 1$ there is a computationally secure encryption scheme using keys of length $O(n^{1/c})$.*

In fact, only the running time of the encryption and decryption will increase for larger c. In the scheme we discuss, ciphertext will have the same size as the original message. For example, if we wanted to send a terabyte of information we could just pick $c = 3$ and use keys of length $1\,000$.

16.3.3 Pseudorandom generators

Briefly, let us discuss a topic that seems quite different but which is actually closely related: *pseudorandom number generators*. The idea is that we want to take a few random bits and inflate them to many random bits. Our goal is to show how we can do this sucsthe result is still "random enough" to any polynomial-time algorithm in that the behaviour of the algorithm will not noticeably change if it reads random bits from a pseudorandom generator instead of truly independent coin flips.

We will efficiently show that if one-way functions exist, then in fact pseudorandom number generators of any polynomial size inflation in the length exist as well. However, we do need to make the following assumption: there exists a one-way injections which preserves the size (*i.e.* such that length m strings map to length m strings in a one-to-one way).

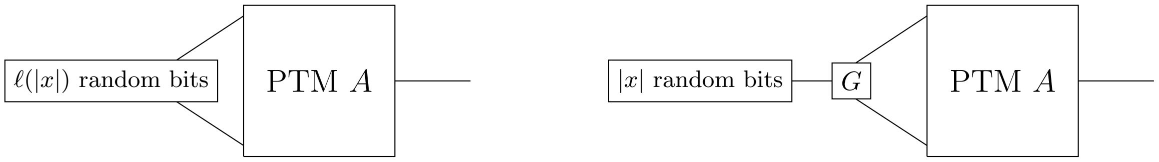
Definition 6 *A function $G : \{0,1\}^* \to \{0,1\}^*$ is a pseudorandom generator with stretch $\ell(n) > n$ if*

- *G can be computed in polynomial-time*
- $\forall x \in \{0,1\}^*, |G(x)| = \ell(|x|)$

- $\forall n, \forall$ *polynomial-time PTM A*

$$\left| \Pr_{r \sim \{0,1\}^n}[A(G(r)) = 1] - \Pr_{r' \sim \{0,1\}^{\ell(n)}}[A(r') = 1] \right| \leq \varepsilon(n)$$

Thus, we have stated that no efficient algorithm can distinguish between using $\ell(|x|)$ truly random bits or sampling $|x|$ random bits and applying the generator G.



The output distributions are effectively identical.

Theorem 3 *If one-way functions exist, then for every $c \in \mathbb{N}$, there exists a pseudorandom generator with stretch n^c, that is, $\ell(n) = n^c$.*

As a corollary, it follows that if pseudorandom generators exist, then computationally secure encryption exists. The proof is left as an exercise for the next assignment: it asks you to show using a one-time pad where the key is the output of a pseudorandom generator is a computationally secure encryption scheme.

We will prove only a special case of the above theorem, when the one-way function is a *permutation.*

Conjecture 2 *There exists one-way permutations: i.e. $|f(x)| = |x|, \forall x$ and f is injective.*

To prove the previous theorem, it will be useful to have the following alternative characterization of pseudorandom generators.

Definition 7 *Say G is unpredictable with stretch $\ell(n)$ if $\forall n, \forall$ polynomial-time PTM B*

$$\Pr_{\substack{x \sim \{0,1\}^n \\ i \sim [\ell(n)]}}[B(1^n, G(x)_1, G(x)_2, \ldots, G(x)_{i-1}) = G(x)_i] \leq \frac{1}{2} + \varepsilon(n)$$

for some negligible $\varepsilon()$. Here, we let $[\ell(n)]$ denote the set $\{1, 2, 3, \ldots, \ell(n)\}$.

For generators, *unpredictable* seems to be a weaker notion than *pseudorandom.* In other words, generator G is unpredictable if predicting a random bit, when given previous bits from the output of G, is difficult for every polynomial-time algorithm. Clearly, if G is a pseudorandom generator then it is also unpredictable. Indeed, by definition of pseudorandom, no efficient algorithm can distinguish between *truly random* and *pseudorandom* bits. Hence, it would be impossible to predict the ith bit in the same way as with truly random coin tosses because the next bit is independent of the previous ones. Intriguingly, the two definitions are equivalent. The harder part is understanding why this inability to predict a single bit means that no efficient algorithm can distinguish between the two at all.

Theorem 4 (Yao 1982) *G is a pseudorandom generator if and only if G is unpredictable.*

Proof Idea. We will prove that if G is not a pseudorandom generator then it is predictable. Suppose that G is not pseudorandom. Then there exists a probabilistic PTM A such that for infinitely many n,

$$\left| \Pr_{r \sim \{0,1\}^n}[A(G(r)) = 1] - \Pr_{r' \sim \{0,1\}^{\ell(n)}}[A(r') = 1] \right| \geq \frac{1}{n^c}$$

Without loss of generality (perhaps by negating the output of A) we can ensure the above inequality holds for infinitely many n's without the absolute value.

We will define our predictor B which will use A to predict the required bit. Given as input $B(1^n, y_1, y_2, \ldots, y_{i-1})$ our algorithm will: (1) sample $z_1, \ldots, z_{\ell(n)} \sim \{0,1\}$ independently *(fill remaining bits of the sequence with truly random bits)* and (2) compute and output $A(y_1, \ldots, y_{i-1}, z_i, \ldots, z_{\ell(n)})$. Considering that A has the property above (that it can somewhat noticeably distinguish between the distributions) we will show that B can predict the correct output with non-negligible probability. We will complete the proof at the start of next lecture.

References

AB09 S. Arora and B. Barak, Computational Complexity: A Modern Approach, *Cambridge University Press, New York, NY, USA*, 2009.