6.893: Cryptographic Techniques for Protecting Privacy **Fall 2020**

Problem Set 1

Due: September 18, 2020 at 5pm, Boston time via Gradescope.

Instructions: You **must** typeset your solution in LaTeX using the provided template:

`https://6893.csail.mit.edu/homework.tex`

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **MGWNYV** to sign up. The solution to each problem must begin on a new page.

Bugs: I make mistakes! If it looks like there might be a mistake in the statement of a problem, please ask a clarifying question on Piazza.

Useful background information for this problem set include: the Union Bound, Markov's inequality, the Chernoff Bound, the Birthday Bound, and averaging arguments. Appendix A of Arora and Barak is one useful reference for some of these topics.

Problem 1: True/False [5 points].

(a) Which of the following are true in a world where $\mathsf{P} = \mathsf{NP}$?

  i Secure PRFs exist in the standard model.

  ii Secure PRFs exist in the random-oracle model.

  iii The one-time-pad cipher is secure.

(b) If there exists a PRG with 1-bit stretch, there exists a PRG with n^{800}-bit stretch (where n is the length of the PRG seed).

(c) Let $P\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a pseudorandom permutation. Then:

  i $f_{k=0}(x) := P(0, x)$ is (always) a one-way function.

  ii $f_{k=0}(x) := P(0, x)$ is (always) a one-way permutation.

  iii $f_{x=0}(k) := P(k, 0)$ is (always) a one-way function.

  iv $f_{x=0}(k) := P(k, 0)$ is (always) a one-way permutation.

Problem 2: Key Leakage in PRFs [5 points]. Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{K} = \mathcal{X} = \mathcal{Y} = \{0,1\}^n$. Let $\mathcal{K}_1 = \{0,1\}^{n+1}$. Construct a new PRF F_1, defined over $(\mathcal{K}_1, \mathcal{X}, \mathcal{Y})$, with the following property: the PRF F_1 is secure; however, if the adversary learns the last bit of the key then the PRF is no longer secure. You must show

- that your PRF is secure and
- an efficient attack on your PRF given the last bit of the PRF key.

This shows that leaking even a *single* bit of the secret key can destroy the PRF security property.
[**Hint:** Try changing the value of F at a single point.]

Problem 3: From a OWP to a PRG [10 points]. Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation. Then consider the function $G\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n+1}$, defined as

$$G(x,r) := \big(f(x) \parallel r \parallel \langle x,r\rangle\big).$$

Here, $\langle x,r\rangle$ is the inner-product of x and r modulo 2.

Notice that G is length-increasing, since it maps $2n$ bits to $2n+1$ bits. We claim that if f is a one-way permutation then G is a pseudo-random generator. In this problem, you will prove that there is no efficient algorithm that takes as input the first $2n$ bits of G's output and predicts the last bit of G's output with high probability.

(a) As a warm-up, say that there exists an efficient algorithm $\mathcal{A}$ such that $\Pr_{x,r}[\mathcal{A}(f(x),r) = \langle x,r\rangle] = 1$. Construct an efficient algorithm $\mathcal{B}$, which calls $\mathcal{A}$ as a subroutine, that perfectly inverts f. That is, $\Pr_x[\mathcal{B}(f(x)) = x] = 1$.

(b) Next, we say that $x \in \{0,1\}^n$ is *good* for $\mathcal{A}$ if $\Pr_r[\mathcal{A}(f(x),r) = \langle x,r\rangle] \geq 3/4+\epsilon$ for some positive constant ϵ, where the probability is taken *only* over $r \xleftarrow{\text{R}} \{0,1\}^n$. Construct an efficient algorithm $\mathcal{B}$ that takes as input $f(x)$ for a good $x \in \{0,1\}^n$ and outputs x with probability at least $1/2$, by calling $\mathcal{A}$ at most $O(n\cdot\log n)$ times.

(c) Assume now that $\mathcal{A}$ satisfies $\Pr_{x,r}[\mathcal{A}(f(x),r) = \langle x,r\rangle] \geq 3/4+\epsilon$, for some constant $\epsilon > 0$, where the probability is taken over the independent and uniform random choice of x and r from $\{0,1\}^n$. Show that x chosen uniformly from $\{0,1\}^n$ is *good* (in the sense of Part (b)) with some constant probability.

What you have shown is that if there is an algorithm $\mathcal{A}$ that predicts (with probability at least $3/4+\epsilon$, for $\epsilon > 0$) the last bit of G's output given the first $2n$ bits of G's output, we can construct an algorithm $\mathcal{A}$ that breaks the one-wayness of f.

Problem 4: Random functions [10 points]. Let $H\colon \{0,1\}^n \to \{0,1\}^n$ be a hash function that we model as a random oracle.

It is common to store user passwords in "hashed" form. That is, rather than storing a password $p \in \{0,1\}^n$, a server stores the hash $h = H(p)$. When a client wants to authenticate to the server, the client sends its password p' to the server. The server computes $h' = H(p')$ and allows the client to log in if $h = h'$.

(a) Fix C distinct passwords $p_1,\ldots,p_C \in \{0,1\}^n$. What is the probability—over the random choice of H—that some two passwords hash to the same value? This probability will be a function of C and n. You may give an upper bound on the probability of a collision, as long as your bound is non-trivial.

(b) One standard technique for increasing the cost of offline password-guessing attacks is to hash the password many times in sequence. (NIST's PBKDF2 standard does this.) Say now that we have T random functions $H_1,\ldots,H_T\colon \{0,1\}^n \to \{0,1\}^n$ and define:

$$H_{\text{big}}(x) := H_T(\cdots H_2(H_1(x))\cdots).$$

Again, fix C distinct passwords and give an upper bound on the probability—now over the random choice of $H_1,\ldots,H_T$—that some two passwords hash to the same value. This probability will be a function of C, T, and n.

(c) To simplify your implementation, you decide to iterate the *same* hash function T times. So now, you hash the passwords using $H^{(T)}$ where

$$H^{(T)}(x) := \underbrace{H(\cdots H(H(x))\cdots)}_{T \text{ times}}.$$

Again, fix C distinct passwords and bound the probability—now just over the random choice of H—that some two passwords hash to the same value.

(d) Modern processors have dedicated hardware instructions for computing the AES block cipher quickly. To improve the number of hash-iterations per second, your friend decides to implement H in the following way:

- For each user i, choose a random AES key k_i.
- Hash the password using $H_{\mathrm{AES}_i}(x)$, where

$$H_{\mathrm{AES}_i}(x) := \underbrace{E(k_i, \cdots E(k_i, (E(k_i, x))\cdots)}_{T \text{ times}}$$

  and $E(\cdot,\cdot)$ is the AES block cipher.
- Store the pair $(k_i, H_{\mathrm{AES}_i}(x))$ in the server's password database.

Say that $T = \mathrm{poly}(n)$. If an attacker steals the password database, how many invocations of AES, as a function of T and n, are required to recover a single user's password? (Here, make the unrealistic assumption that the user's password is a random n-bit string.)

(e) **Extra credit [3pts] – Optional, but recommended!** You use the hash function from Part (c) with $T = 2^{2n/3}$. Let $h \in \{0,1\}^n$ be the hash of a random n-bit password under $H^{(T)}$. Show that an attacker can find a password $p^* \in \{0,1\}^n$ such that $h = H^{(T)}(p^*)$ by invoking H at most $2^{n/2} \cdot \mathrm{poly}(n)$ times. That is, even though we iterated H for $2^{2n/3}$ iterations, there is a password-cracking attack that runs in time $\approx 2^{n/2} \ll 2^{2n/3}$.

Does the same attack work if you use H_{big} from Part (b) with $T = 2^{2n/3}$?

Problem 5: Feedback [1 points].

(a) Roughly how many hours did you spend on this problem set?

(b) What was your favorite problem on this problem set? In one sentence: why?

(c) What was your least favorite problem on this problem set? In one sentence: why?

(d) **[Optional]** If you have any other feedback on this problem set or on the course, please write it here or submit it using the anonymous feedback form.