Massachusetts Institute of Technology
6.5610: Applied Cryptography and Security — February 3, 2026
Professor Srini Devadas — **Due:** February 13, 2026 at 4:59pm

Problem Set 1

Please submit your problem set, in PDF format, on Gradescope. *Each problem should be in a separate page.*

You are to work on this problem set in groups. For problem sets 1 and 2 we will randomly assign the groups for the problem set. After problem set 2, you are to work on the following problem sets with groups of your choosing of size three or four. If you need help finding a group, try posting on Piazza. See the course website for our policy on collaboration. Each group member must independently write up and submit their own solutions.

Homework must be typeset in LaTeXand submitted electronically! Each problem answer must be provided as a separate page. Mark the top of each page with your group member names, the course number (6.5610), the problem set number and question, and the date. We have provided a template for LaTeX on the course website (see the *Psets* tab at the top of the page).

Problem 1-1. One-way functions and collision resistance

A family of functions $\{f_\lambda\}_{\lambda\in\mathbb{N}}$, where each $f_\lambda : \{0,1\}^\lambda \to \{0,1\}^*$, is *one-way function* (OWF) if each f_λ is computable in polynomial-time and, for every polynomial-time adversary A, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[f_\lambda(x) = f_\lambda(x') \colon \begin{array}{c} x \xleftarrow{R} \{0,1\}^\lambda \\ x' \leftarrow A(f_\lambda(x)) \end{array}\right] \le \mu(\lambda).$$

In other words, given $f_\lambda(x)$ and λ it is difficult to find x' such that $f_\lambda(x') = f_\lambda(x)$.

A family of functions $\{f_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be *collision resistant* if it is polynomial-time computable, for every $\lambda \in \mathbb{N}$, $f_\lambda : \{0,1\}^* \to \{0,1\}^\lambda$, and for all polynomial-time adversaries A, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[f_\lambda(x) = f_\lambda(x') \wedge x \neq x' \ :\ (x, x') \leftarrow A(1^\lambda)\right] \le \mu(\lambda).$$

In other words, it is difficult to find distinct x, x' such that $f_\lambda(x) = f_\lambda(x')$.

For each of the following functions $g = \{g_\lambda\}_{\lambda\in\mathbb{N}}$ determine if g is necessarily a one-way function (OWF). If so, explain in a few sentences why it is a OWF, and if not, provide an attack.

For simplicity, in what follows we define f and g for a given input length, and omit the subscript λ from the collision resistant hash functions.

(a) Let $f : \{0,1\}^\lambda \to \{0,1\}^*$ be a OWF, and let $g : \{0,1\}^\lambda \to \{0,1\}^*$ where $g(x) = f(x) \,\|\, x[0]$.

(b) Let $f : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a OWF, and let $g : \{0,1\}^\lambda \to \{0,1\}^\lambda$ where $g(x) = f(f(x))$.

(c) Let $g : \{0,1\}^* \to \{0,1\}^\lambda$ be collision resistant. Is $g|_{\{0,1\}^{2\lambda}}$ necessarily a OWF (where $g|_{\{0,1\}^{2\lambda}}$ denotes the function g restricted to the domain $\{0,1\}^{2\lambda}$)?

For each of the following functions g determine if g is necessarily a collision resistant function. If so, explain in a few sentences why it is collision resistant, and if not, provide an attack.

(d) Let $f : \{0,1\}^* \to \{0,1\}^\lambda$ be collision resistant, and let $g : \{0,1\}^* \to \{0,1\}^\lambda$ where $g(x) = f(x[: -1])$, where $x[: -1]$ is x with the last bit removed.

(e) Let $f : \{0,1\}^* \to \{0,1\}^\lambda$ be collision resistant, and let $g : \{0,1\}^* \to \{0,1\}^\lambda$ where $g(x) = f(f(x))$.

Problem 1-2. Hellman Tables

Hellman's algorithm is cryptanalytic tool for function inversion with preprocessing. These preprocessing attacks are useful when an attacker wants to invert the same cryptographic function (e.g., a hash function) many times at different points.

A preprocessing algorithm for function inversion works in two phases:

- **Offline (preprocessing) phase:** The attacker evaluates f many many times and outputs a data structure σ_f of size S that encodes its knowledge about f.
- **Online phase:** The attacker has a challenge instance $y = f(x)$, for an unknown x, and the attacker wants to find x. The attacker uses it's precomputed data structure σ_f, makes T queries to the function f and outputs a value x' such that $f(x') = y$.

The goal is to jointly minimize the size S (in machine words) of the precomputed data structure σ_f, along with the running time T of the online algorithm.

(a) Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a PRF. Say that an attacker wants to perform precomputation relative to F such that, later on, given oracle access to $F(k, \cdot)$ for some unknown key, the attacker can recover k. Explain how the attacker can use a precomputation attack to solve this problem.

(b) In 1–2 sentences, give a preprocessing algorithm that inverts an arbitrary function $f\colon [N] \to [N]$ using space $S = 0$ and online time $T = N$, where we define $[N] = \{1, 2, 3, \ldots, N\}$.

(c) In 1–2 sentences, give a preprocessing algorithm that inverts an arbitrary function $f\colon [N] \to [N]$ using space $S = N$ and online time $T = 0$.

Hellman's method achieves a much more interesting time-space trade-off than the algorithms you constructed in (a) and (b). To see how Hellman's algorithm works, let's examine the case where f is a random permutation defined on some space $\mathcal{X}$ (such as $\{0,1\}^\lambda$), that is $f : \mathcal{X} \to \mathcal{X}$ and is one-to-one. We can view the function f as a directed graph $G = (V, E)$ with vertex set $V = \mathcal{X}$ and edge set $E = \{(x, y) : f(x) = y\}$.

During the preprocessing phase, Hellman's algorithm traverses the graph and stores back pointers every t steps. Then, when inverting a point in the online phase, the algorithm walks the graph forward, checking for back pointers in the table at each step until it hits the starting value again. The online algorithm then return the parent node of the starting value. Since f is a permutation, each node in G will have one parent and thus G will be a union of cycles.

(d) What is the correct value of t to minimize S and T as stated above? Given this value, what are the values of S and T?

When performing this algorithm, the amount of storage required for Hellman's data structure is often too large to fit in a computer's memory, so it must be stored on disk. Thus, the online algorithm needs to make roughly t disk accesses when exploring the graph. However, we can decrease the amount of disk accesses by performing some tricks, which can improve the runtime of the algorithm.

(e) Imagine that the online phase of Hellman's algorithm runs on a computer with an arbitrarily large hard disk and $\mathsf{polylog}(N)$ RAM memory. Explain how to modify Hellman's algorithm in such a way that the online algorithm requires only $O(1)$ disk accesses per inversion.

You may assume that you have access to a truly random function on whatever domain and range you like. *Hint: Think about how one would determine if a back pointer exists at value x before checking the table.*

Problem 1-3. Finite rings and fields

For any positive integer n consider the ring $\mathbb{Z}_n$ that consists of the elements $\{0, 1, \ldots, n-1\}$ where addition and multiplication are modulo n.

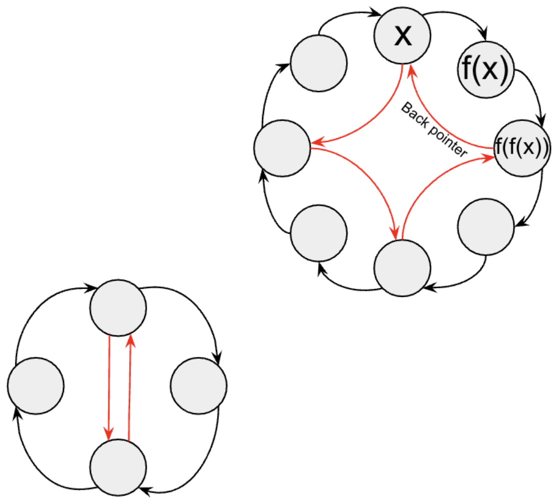

Figure 1: Example of what G might look like. The red arrows represent the back pointers from Hellman's algorithm.

(a) Show that for every prime p, every non-zero element in $\mathbb{Z}_p$ has a multiplicative inverse (and hence it is a field, in which case it is often denoted by $\mathbb{F}_p$).
(Hint: You may use the extended GCD algorithm that given two integers x, y outputs two integers a, b such that $a \cdot x + b \cdot y = \mathsf{GCD}(x, y)$ where here arithmetic is over the integers; assume standard integer properties).

(b) For any non-prime n, construct an element $a \in \mathbb{Z}_n$ that does not have an inverse (and hence $\mathbb{F}_n$ is not a field)

(c) Consider the polynomial $f(x) = x^2 + 2x + 2$ over $\mathbb{F}_3$.

1. Show that $f(x)$ is irreducible over $\mathbb{F}_3$.
   (Hint: f is irreducible over $\mathbb{F}_3$ if and only if it has no roots in over $\mathbb{F}_3$).
2. Consider the finite field $\mathbb{F}_{3^2} = \mathbb{F}_3/f(x)$, which consists of all linear functions over $\mathbb{F}_3$ where addition is coordinate-wise modulo 3 and multiplication is done modulo the polynomial $f(x)$. List all the elements in this field.
3. Find the order of the element x in $\mathbb{F}_{3^2}$, namely, find the smallest integer $a > 0$ such that $x^a = 1$?

Problem 1-4. Birthday attack on Even-Mansour encryption scheme

In the class, we see how AES is constructed by alternately XOR-ing with the key and applying a random permutation π. Here, we consider the single-key Even-Mansour scheme, similar to a one-round AES:

$$\mathsf{Enc}(m) = \pi(m \oplus k) \oplus k,$$

where the secret key k and the message m are both n bit long. Your job is to break the scheme with a birthday attack. The first step of the attack would be to collect many pairs of $(m_i, \mathsf{Enc}(m_i))$, then construct a computable function f such that if two messages collide (i.e. $f(m_1) = f(m_2)$), it reveals the secret key by somehow mixing those two messages.

(a) Find such a function f, and how a collision pair reveals the secret key.
Hint: Notice that for any two messages m_1, m_2, $\mathsf{Enc}(m_1) \oplus \pi(m_1 \oplus k) = \mathsf{Enc}(m_2) \oplus \pi(m_2 \oplus k)$. Under which condition (on m_1 and m_2) can we eliminate the only unknown k?

Now we have transformed the problem of recovering the secret key into a collision-finding problem. Assuming that π is a random-like function, we can also model your f as a random function. By the birthday paradox, we can find collisions with $\Theta(2^{n/2})$ plaintexts. However, since f is random-like, we can hit a collision "by accident", in which case we cannot recover the secret key.

(b) Prove that if two uniformly random messages m_1, m_2 satisfy $f(m_1) = f(m_2)$, there exists a constant c such that the probability of m_1, m_2 revealing the secret key is at least c. You may model every $f(x)$ as a random n-bit string.

This means that if we try enough collision pairs, we should be able to find the secret key. With all the theory in our heads, we now implement the attack in Python.

(c) On Piazza, you can find a zip file `pset1.zip` that contains the file needed for this problem.

- `pset1_enc.py` implement the Even-Mansour encryption with a hidden key as a function `enc(m:int) -> int`, but the code is obfuscated. Your goal is to find the key (hopefully not by de-obfuscate the code).
- `pset1_main.py` provides the skeleton code, including an example encryption scheme. **The random permutation π is the same for the example scheme and the one in** `pset1_enc.py`. You can modify the key and see if your attack can recover the key correctly.

Please write down the secret key (in hex format) as the answer and submit your code (`pset1_main.py`) to "Problem Set 1 Code" on Gradescope. It will be autograded on a set of private keys. We read your code for sanity checks, so make sure that the code is fairly understandable.

Note: On our Macbook Pro 2023, the attack takes less than 10 seconds for the test implementation, and 3-15 minutes for the obfuscated version with the secret key. If you want to speed it up, try implementing Pollard's rho algorithm, or even the parallelized version.