1. (70 points) The CBC-MAC construction we learned in class is sequential: one has to finish a block-cipher call before one can proceed to the next one. In modern platforms, this sequential behavior is undesirable. Here’s an attempt to build a fully parallelizable MAC, which we’ll call XMAC. Let’s say we have a blockcipher \( E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n \). Fix a number \( r < n \). Write \([i]\) to denote an \( r \)-bit encoding of integer \( i \leq 2^r \). Suppose that you have a message \( M = P_1 \cdots P_m \), where \( m \leq 2^r \) and each \( |P_i| = n - r \). To generate a tag for it, compute

\[
T = E_K([1] \parallel P_1) \oplus \cdots \oplus E_K([m] \parallel P_m)
\]

In practice \( n = 128 \), and one can choose \( r = 32 \). (Thus your message can be at most 64 GB.) In that case, XMAC is several times faster than CBC-MAC. Unfortunately, XMAC is insecure. Break its MAC security using 3 queries and analyze your advantage. Each message in your queries and forgery output should have just two \((n-r)\)-bit blocks.

2. (60 points) In this problem, our hash function \( H \) is the 32-bit truncated SHA-256. That is, \( H(x) \) is the first 32 bits of SHA-256\((x)\) for every string \( x \). You are supposed to implement the Rho method to find collision on \( H \).

**Output format.** Your program needs to output two distinct strings \( x \) and \( x' \), their common hash output \( H(x) = H(x') \), and their full SHA-256 outputs SHA-256\((x)\) and SHA-256\((x')\) in this order. If you can’t find a collision, output a failure message.

**Requirements.** Since this is a constant-memory attack, you’re not allowed to use any data structure in your code. Moreover, you have to start with a random point \( x_0 \), meaning that if I run your program multiple times, I should get different collisions. To avoid long waiting time, you should terminate your program with a failure message if the Rho attack can’t detect a cycle after \( 2^{20} \) steps. **Make sure that your program can run in 1minprog.**

**Deliverables.** Upload to Canvas a zip file containing your source code, which includes a README.txt that informs me how I should run the program. **Make sure that your zip file is not corrupted.**

3. (120 points) (Variants of padding-oracle attack)

a) (60 points) A null character is simply the byte 00. We say that a byte string is properly null terminated if its last character is null, but no other characters are null.

Suppose that you are given a CTR-mode encryption\(^1\) of an unknown (but properly null terminated) byte string \( M \). In addition, you are given a decryption oracle Dec. If you query \( \text{Dec}(C) \) then the oracle will CTR-decrypt \( C \) to obtain a string \( M' \), and then return true if and only if \( M' \) is properly null terminated, but you don’t get to learn the decrypted message \( M' \).

Recover the entire secret message \( M \); brute-force attacks will be given zero credit. For simplicity, you only need to show how to recover the **second last** byte of \( M \) because the last byte is known to be 00

---

\(^1\)Here CTR is exactly the same as we learned in our class. If the message is fragmentary then we generate a slightly longer pad, truncate it, and then xor the pad to the message.
and recovering other bytes is similar. How many decryption queries will you need in the worst case to recover the entire message?

b) (60 points) Here you have a secret byte string $M$ and an encryption oracle $\text{Enc}$. If you query an IV and a prefix $P$ to the encryption oracle then it returns the CBC-encryption of $P||M$ (padded with $10^*$) under your dictated IV.

Recover the entire secrete message $M$; brute-force attacks will be given zero credit.

Let $M = X_1 \cdots X_m$ where each $X_i$ is a byte. The attack will recover $X_1$, then $X_2$, then $X_3$, and so on. For simplicity, suppose that $m > 16$; you only need to show how to recover $X_{17}$, assuming that $X_1, \ldots, X_{16}$ are already recovered. How many encryption queries will you need in the worst case to recover the entire message?