Student ID: ______________________________

This sheet will not be graded (feel free to write on it), but you must turn it in at the end of the exam.

C Function Definitions

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

      The  function  fread() reads nmemb items of data, each size bytes long,
      from the stream pointed to by stream,  storing  them  at  the  location
      given by ptr.

      Note that fread() does not add  a null byte after input.

char *fgets(char *s, int size, FILE *stream);

      fgets() reads in at most one less than size  characters from stream and
      stores them into the buffer pointed to by s. Reading stops after an EOF
      or a newline.  If a newline is read, it is stored into  the  buffer.  A
      terminating null byte ('\0') is stored after the last character  in the
      buffer.

char *gets(char *s);

      gets() reads a line  from  stdin  into the buffer pointed to by s until
      either a terminating newline or EOF,  which  it  replaces  with  a null
      byte ('\0').
```

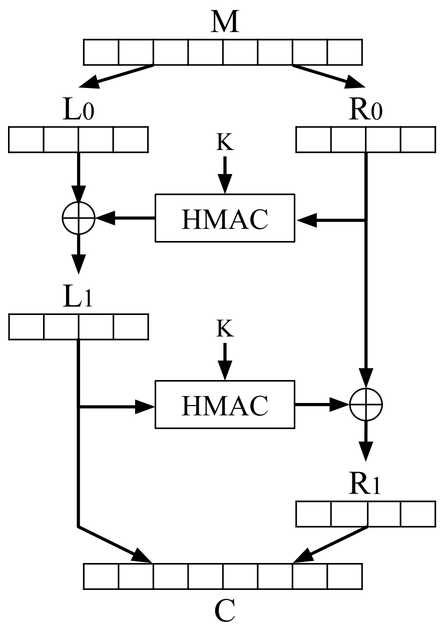
General Exam Assumptions

Unless otherwise specified, you can assume these facts on the entire exam:

- Memory safety:
  - You are on a little-endian 32-bit x86 system.
  - There is no compiler padding or saved additional registers.
  - If stack canaries are enabled, they are four completely random bytes (no null byte).
  - You can write your answers in Python syntax (as seen in Project 1).
  - Unless otherwise specified, all other memory safety defenses are disabled.
  - Each x86 instruction is 4 bytes long in machine code.
- Cryptography:
  - The attacker knows the algorithms being used (Shannon's maxim).
  - $\|$ denotes concatenation.
  - H refers to a secure cryptographic hash function.
  - g and p refer to a public generator element and large prime modulus, respectively.
  - IVs are randomly generated per encryption unless otherwise specified.

Q5 Diagram Copy

Here is a copy of the encryption diagram from Q5, *Not Quite By DESign*, for your reference:



For the rest of the question, consider this scheme for encrypting 128-bit messages:

1. Split the message into two halves.

   $L_0 = M$`[:64]`

   $R_0 = M$`[64:128]`

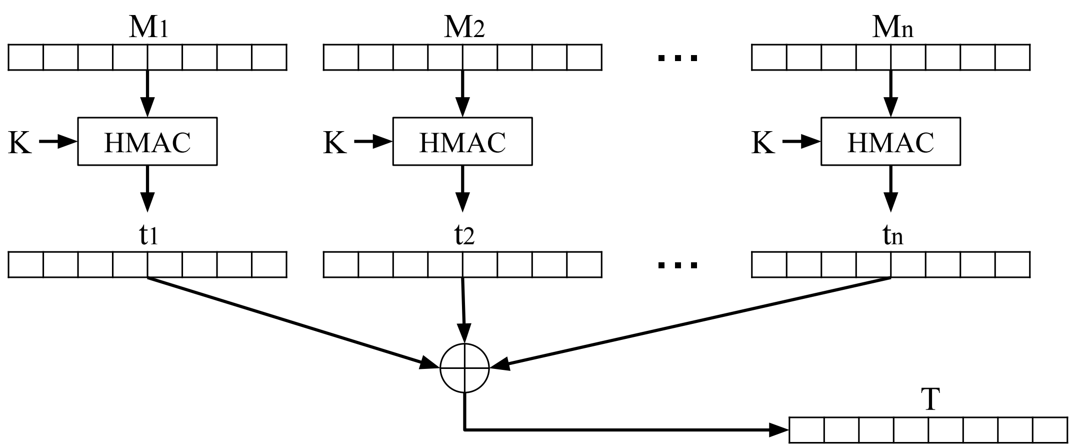
2. Set $L_1 = L_0 \oplus \mathsf{HMAC}(K, R_0)$.
3. Set $R_1 = R_0 \oplus \mathsf{HMAC}(K, L_1)$.
4. Concatenate L_1 and R_1 to get the ciphertext.

   $C = L_1 \| R_1$

For this question, you can assume that HMAC outputs 64 bits.

Q6 Diagram Copy

Here is a copy of the first scheme from Q6, *Mix-and-MAC*, for your reference:



1. Compute HMACs on each individual message. $t_i = \mathsf{HMAC}(K, M_i)$, for $1 \leq i \leq n$.
2. XOR all the HMAC outputs (t_i) together to get the final MAC output. $T = t_1 \oplus t_2 \oplus \ldots \oplus t_n$.