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.

int printf(const char *format, ...);

      printf() produces output according to the format string format.

      Conversion specifiers:
      %c  Character.
      %d  Signed integer.
      %n  Writes the number of bytes printed so far, as a 4-byte
          integer, to the corresponding memory address.
      %s  String.
      %u  Unsigned integer.
      %x  Unsigned integer, in hexadecimal.

      Each of the above conversion specifiers reads a 4-byte argument on the stack.

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').

void *memset(void s, int c, size_t n);

      The memset() function fills the first n bytes of the memory area
      pointed to by s with the constant byte c.
```

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.
  - Enc refers to an IND-CPA secure encryption scheme unless otherwise specified.

Below is the code in Q4, repeated for your convenience.

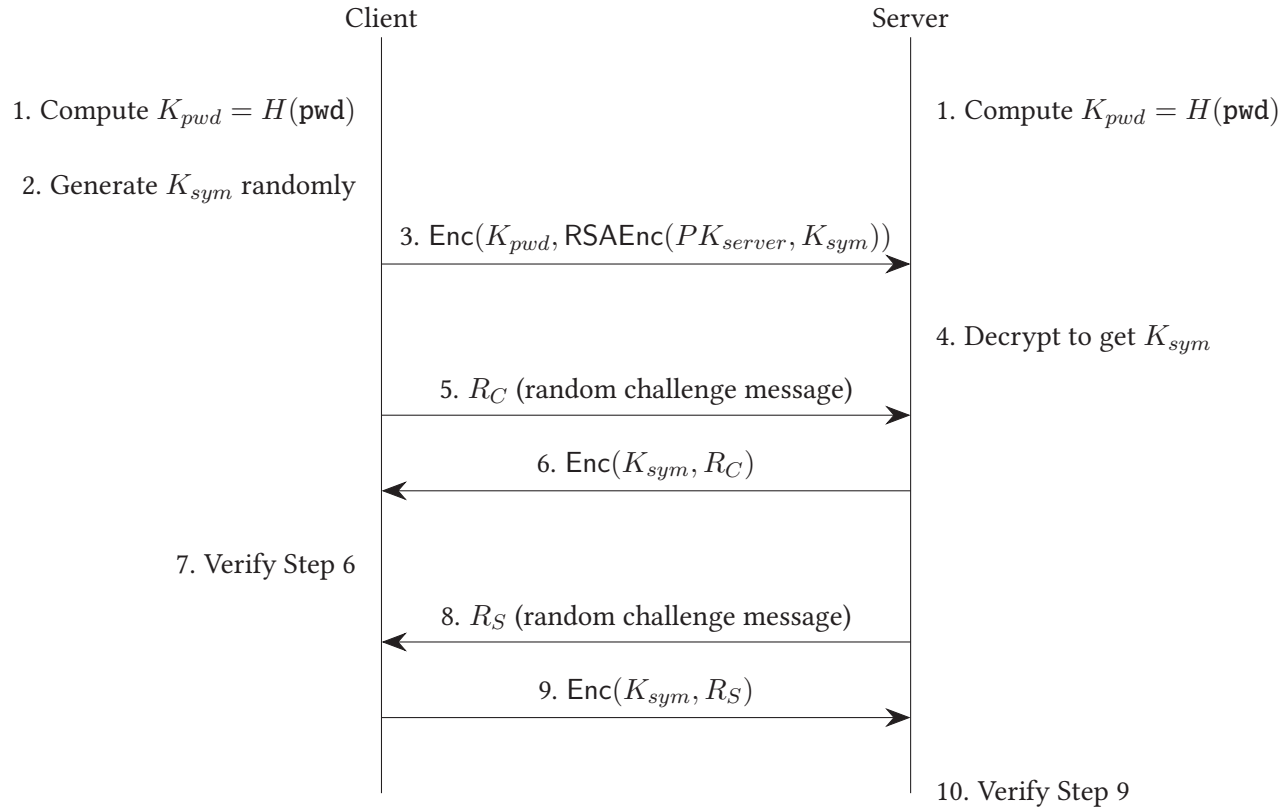
```
1  void boat(void* shellcode_first_half, void* shellcode_second_half) {
2      // fp contains the address of the fgets function
3      uintptr_t fp = (uintptr_t) fgets;
4
5      char[32] buf;
6      char* buf_ptr = &buf;
7
8      fgets(buf, 32);
9      printf(buf);
10
11     fgets(buf, 32);
12     printf(buf);
13 }
```

This is the result of running `disas fgets` in GDB:

```
1 0x08076030: push %ebp
2 0x08076034: mov %esp, %ebp
3 0x08076038: sub %ebp, 20
4 ...
5 0x08076050: mov %ebp, %esp
6 0x08076054: pop %ebp
7 0x08076058: ret
```

Below is the scheme from Q7, repeated for your convenience.

`pwd` is a standard-strength password (i.e. vulnerable to brute-force). PK_{server} is a long-term, trusted public key for the server. Assume there's only a single user/password stored on the server.



Here is an equivalent description of the protocol:

1. Both the client and server derive $K_{pwd} = H(\texttt{pwd})$.
2. The client generates a random symmetric key K_{sym}.
3. The client sends $\mathsf{Enc}(K_{pwd}, \mathsf{RSAEnc}(PK_{server}, K_{sym}))$ to the server.
4. The server decrypts the message from the previous step to get K_{sym}.
5. The client sends a randomly generated number R_C to the server (challenge message).
6. The server replies with $\mathsf{Enc}(K_{sym}, R_C)$.
7. The client verifies that the server's response is valid.
8. The server sends a randomly generated number R_S to the client (challenge message).
9. The client replies with $\mathsf{Enc}(K_{sym}, R_S)$.
10. The server verifies that the client's response is valid.