# Lab Assignment #1

Q1

cracking_password.c > ...

```c

#include <stdio.h>

int main() {
    // Declare four character variables to represent each letter in the combination
    char i, j, k, l;

    // First loop: iterate 'i' through all lowercase letters from 'a' to 'z'
    for(i = 'a'; i <= 'z'; i++) {
        // Second loop: iterate 'j' through all lowercase letters
        for(j = 'a'; j <= 'z'; j++) {
            // If 'j' is the same as 'i', skip to the next iteration
            // to avoid repeating the same letter
            if(j == i) continue;

            // Third loop: iterate 'k' through all lowercase letters
            for(k = 'a'; k <= 'z'; k++) {
                // If 'k' is the same as 'i' or 'j', skip to the next iteration
                if(k == i || k == j) continue;

                // Fourth loop: iterate 'l' through all lowercase letters
                for(l = 'a'; l <= 'z'; l++) {
                    // If 'l' is the same as 'i', 'j', or 'k', skip to the next iteration
                    if(l == i || l == j || l == k) continue;

                    // Print the combination of letters
                    printf("%c%c%c%c\n", i, j, k, l);
                }
            }
        }
    }
    return 0;
}

```

This code systematically iterates through the alphabet for each position in the 4-letter combination, skipping any letters that have already been used in that combination to ensure uniqueness. Each combination is printed on a new line.

Output


The **| wc -l** part of the Bash command is used to count the number of lines outputted by the preceding command. Here's a breakdown:

- **|** (pipe): This is a pipe in Unix and Linux. It takes the output of the command on its left and uses it as the input for the command on its right.
- **wc** (word count): This is a command in Unix/Linux that displays the number of lines, words, and bytes contained in a file or provided as input.
- **-l** (lines): When used with **wc**, this option tells **wc** to count only the number of lines.

So, when you run **./a.out | wc -l**, it means:

1. Execute the **./a.out** command (which runs your compiled C program).
2. Take the output of **./a.out** (which is the list of all possible 4-letter combinations) and pass it to **wc**.
3. Use **wc -l** to count the number of lines in the provided input, which corresponds to the number of combinations.

Since each combination is printed on a new line by the C program, counting the lines gives you the total number of combinations.

Q2

**Task 1: Completing the C Program to Find the Private Key**

In the given **decryptKey.c** program, you need to:

1. Initialize BIGNUM variables for **e**, **n**, **phi(n)**, **d**, the encrypted message **C**, and the decrypted message **D**.
2. Calculate **d** using **BN_mod_inverse()**.
3. Read the encrypted message from the user and decrypt it using **BN_mod_exp()**.

The completed sections are highlighted:

C cracking_password.c U  C decryptKey.c 3, U

C decryptKey.c > main(int, char * [])

```c
#include <stdio.h>
#include <string.h>
#include <openssl/bn.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

// Function to print a BIGNUM value in hexadecimal format
void printBN(char *msg, BIGNUM *tmp){
    char *number_str = BN_bn2hex(tmp);
    printf("%s%s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main(int argc, char *argv[]){
    // Initialize context for OpenSSL functions
    BN_CTX *ctx = BN_CTX_new();

    // Initialize BIGNUM variables for RSA key components and messages
    BIGNUM *e = BN_new(); // Public key exponent
    BIGNUM *d = BN_new(); // Private key
    BIGNUM *n = BN_new(); // Modulus
    BIGNUM *phi_n = BN_new(); // Totient of n
    BIGNUM *C = BN_new(); // Encrypted message (ciphertext)
    BIGNUM *D = BN_new(); // Decrypted message

    // Assign values to public key exponent (e) and modulus (n) from hexadecimal strings
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&n, "E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1");
    BN_hex2bn(&phi_n, "E103ABD94892E3E74AFD724BF28E78348D52298BD687C44DEB3A81065A7981A4");

    // Calculate the private key (d) using modular inverse of e and phi_n
    BN_mod_inverse(d, e, phi_n, ctx);

    char *CC = malloc(100 * sizeof(char));
    printf("\nEnter your Encrypted Message:\n");
    scanf("%s", CC); // Read the encrypted message from user input
```

```c
C decryptKey.c > main(int, char * [])
37      scanf("%s", CC); // Read the encrypted message from user input
38      BN_hex2bn(&C, CC); // Convert the input to BIGNUM format
39
40      // Decrypt the ciphertext using RSA decryption formula: D = C^d mod n
41      BN_mod_exp(D, C, d, n, ctx);
42
43      // ... [earlier parts of the program]
44
45  // Convert the decrypted message from BIGNUM to hexadecimal string
46  printf("\nOriginal Message:\n");
47  char str1[500] = "print(bytes.fromhex(\"";
48  char *str2 = BN_bn2hex(D);
49  char str3[] = "\").decode('utf-8'))";
50  strcat(str1, str2);
51  strcat(str1, str3);
52  char* args[] = {"python3", "-c", str1, NULL};
53  execvp("python3", args); // Execute Python3 command to convert hex to ASCII
54
55  // ... [rest of the program]
56
57
58      // Free allocated memory for BIGNUM variables and context
59      BN_free(e);
60      BN_free(d);
61      BN_free(n);
62      BN_free(phi_n);
63      BN_free(C);
64      BN_free(D);
65      BN_CTX_free(ctx);
66      free(CC);
67
68      return EXIT_SUCCESS;
69
70
71  }
```

```c
// Convert the decrypted message from BIGNUM to hexadecimal string
printf("\nOriginal Message:\n");
char str1[500] = "print(bytes.fromhex(\"";
char *str2 = BN_bn2hex(D);
char str3[] = "\").decode('utf-8'))";
strcat(str1, str2);
strcat(str1, str3);
char* args[] = {"python3", "-c", str1, NULL};
execvp("python3", args); // Execute Python3 command to convert hex to ASCII
```

This modification uses Python 3's **bytes.fromhex()** method to convert the hexadecimal string to bytes and then decodes it as 'utf-8' to get the ASCII representation.

After making these changes, recompile your program and run it again. The output should now properly display the original message in ASCII format, decoded by Python 3.

Output 1

```
os@MSI:/mnt/c/users/1999o/Downloads/cfiles$ gcc decryptKey.c -lcrypto
os@MSI:/mnt/c/users/1999o/Downloads/cfiles$ ./a.out

Enter your Encrypted Message:
7CED643C0FD1559F41E734321E19B66ED86A8E866C5C329DC8CC5DE980CC7A7A

Original Message:
EE463: Operating Systems
os@MSI:/mnt/c/users/1999o/Downloads/cfiles$ gcc encryptRSA.o -lcrypto -o encryptRSA
```

Output 2

**Task 2: Decrypting the Given Message**

Run the compiled program, and when prompted, enter the encrypted message you want to decrypt. The program will output the original plaintext message.

Regarding the attached **encryptRSA.o** file, you can compile it as per the instructions given and use it to encrypt messages. The C program will then be able to decrypt these messages if you have the correct private key (**d**).

```
os@MSI:/mnt/c/users/1999o/Downloads$ gcc encryptRSA.o -lcrypto -o encryptRSA
os@MSI:/mnt/c/users/1999o/Downloads$ ./encryptRSA

Enter Original Message:
King Abdulaziz University

Encoded Message:

Re-enter Encoded Message:
4b696e6720416264756c617a697a20556e6976657273697479

Encrypted Message:
0D0E0218FA3056DF66689798745DA5F05A11EDD8BA532622DB530787BAF72E2D

Re-enter Encoded Message:
0D0E0218FA3056DF66689798745DA5F05A11EDD8BA532622DB530787BAF72E2D

Encrypted Message:
9A23A7A6B1D078178B78280D2964425EE9DAD9C817FE51DA81A46A30D79A0F48
```