CT437 Assignment 3

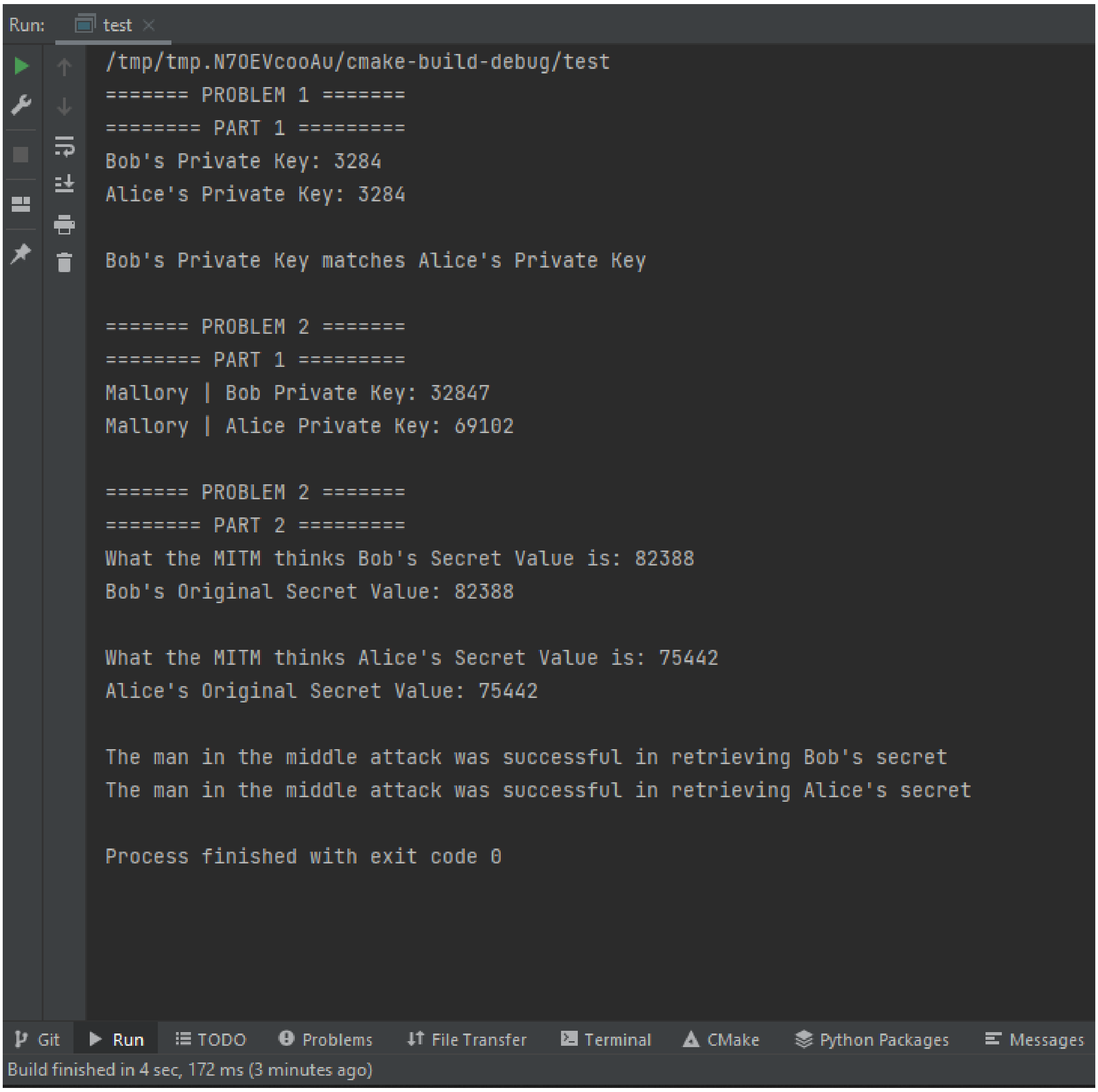
Source Code

```
// Author: Conor Mc Govern
// Module: CT437
// Assignment: 3

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <string.h>

struct dh_params {
        long prime;
        long primitive_root;
};

// Adapted from C++ code on GeeksForGeeks
short int is_prime(long n){
        long i;
        if (n % 2 == 0)
        return 0;
        for (i = 3; i < ceil(sqrt(n)); i+=2){
        if(n % i == 0)
        return 0;
        }
        return 1;
}

// Adapted from C++ code on GeeksForGeeks
long power(long x, long y, long p)
{
        long res = 1;    // Initialize result

        x = x % p; // Update x if it is more than or
        // equal to p

        while (y > 0)
        {
        // If y is odd, multiply x with result
        if (y & 1)
        res = (res*x) % p;
```

```
    // y must be even now
    y = y >> 1; // y = y/2
    x = (x*x) % p;
    }
    return res;
}

// Inserts into the next available position of the array
// Only inserts once
void insert(long s[1000], int n)
{
    for (int i = 0; i < 1000; i++)
    {
    if (s[i] == n) return;
    if (s[i] == 0)
    {
    s[i] = n;
    break;
    }
    }
}

// Adapted from C++ code on GeeksForGeeks
void find_prime_factors(long s[1000], long n)
{
    // Print the number of 2s that divide n
    while (n%2 == 0)
    {
    insert(s, 2);
    n = floor(n/2);
    }

    // n must be odd at this point. So we can skip
    // one element (Note i = i +2)
    for (int i = 3; i <= (int) sqrt(n); i = i+2)
    {
    // While i divides n, print i and divide n
    while (n%i == 0)
    {
    insert(s, i);
    n = floor(n/i);

    }
    }

    // This condition is to handle the case when
    // n is a prime number greater than 2
    if (n > 2)
    insert(s, n);
}

// Function to find the smallest primitive root of n
// Adapted from C++ code on GeeksForGeeks
long find_primitive(long n)
{
    long s[1000];
    memset(s, 0, 1000);

    // Find value of Euler Totient function of n
    // Since n is a prime number, the value of Euler
    // Totient function is n-1 as there are n-1
    // relatively prime numbers.
    long phi = n-1;

    // Find prime factors of phi and store in a set
    find_prime_factors(s, phi);

    // Check for every number from 2 to phi
    for (int r=2; r<=phi; r++)
    {
    // Iterate through all prime factors of phi.
    // and check if we found a power with value 1
    int flag = 0;
    //for (auto it = s.begin(); it != s.end(); it++)
    for (int i = 0; s[i] != 0; i++)
    {
    // Check if r^((phi)/primefactors) mod n
    // is 1 or not
    if (power(r, floor(phi/s[i]), n) == 1)
    {
        flag = 1;
        break;
    }
    }

    // If there was no power with value 1.
    if (flag == 0)
    return r;
    }

    // If no primitive root found
    return -1;
}

// Get a big secret but it has to be less than the prime number
long get_secret(long prime)
```

```
{
	return rand() % (prime - 10000) + 10000;
}

long get_public_key(struct dh_params *params, long secret)
{
	// YA = a^x mod q
	/* As root^secret is a massive number (close to infinity), we split it into
	* bite sized chunked, so we don't overflow the double space. */
	long k = params->primitive_root;
	long i = 0;
	do {
	k *= params->primitive_root;
	k %= params->prime;
	i++;
	} while (i < secret);
	return k;
}

long get_private_key(long prime, long publicKey, long secret)
{
	// K = (y)^x mod q
	/* As publicKey^secret is a massive number (close to infinity), we split it into
	* bite sized chunked, so we don't overflow the double space. */
	long k = publicKey;
	long i = 0;
	do {
	k *= publicKey;
	k %= prime;
	i++;
	} while (i < secret);
	return k;
}

long man_in_the_middle(struct dh_params *params, long publicKey)
{
	for (int i = 0; i < params->prime; i++)
	{
	if (get_public_key(params, i) == publicKey)
	return i;
	}
	return -1;
}

struct dh_params *get_random_dh_params()
{
	struct dh_params *params = malloc(sizeof(struct dh_params));
	// Find a random prime number
	do {
	long random_number = rand() % (100000 + 1 - 10000) + 10000;
	if (is_prime(random_number)) params->prime = random_number;
	} while(params->prime == 0);

	params->primitive_root = find_primitive(params->prime);

	return params;
}

int main() {
	srand(time(NULL));

	struct dh_params *params = get_random_dh_params();

	/*
	* ======= PROBLEM 1 =======
	* ======== PART 1 =========
	*/
	puts("======= PROBLEM 1 =======\n======== PART 1 =========");
	long bob_secret = get_secret(params->prime);
	long bob_public = get_public_key(params, bob_secret);
	long alice_secret = get_secret(params->prime);
	long alice_public = get_public_key(params, alice_secret);

	// Bob exchanges with Alice
	long bob_private = get_private_key(params->prime, alice_public, bob_secret);
	// Alice exchanges with Bob
	long alice_private = get_private_key(params->prime, bob_public, alice_secret);

	printf("Bob's Private Key: %lu\n", bob_private);
	printf("Alice's Private Key: %lu\n\n", alice_private);

	// Show that both Alice and Bob calculate the same key K
	if (alice_private == bob_private)
	puts("Bob's Private Key matches Alice's Private Key\n");
	else
	puts("Bob's Private Key does not match Alice's Private Key\n");

	/*
	* ======= PROBLEM 2 =======
	* ======== PART 1 =========
	*/
	puts("======= PROBLEM 2 =======\n======== PART 1 =========");
	long mallory_secret = get_secret(params->prime);

	// Mallory intercepts Bob's public key
	long mallory_bob_private = get_private_key(params->prime, bob_public,
mallory_secret);
	printf("Mallory | Bob Private Key: %lu\n", mallory_bob_private);

	// Mallory intercepts Alice's public key
	long mallory_alice_private = get_private_key(params->prime, alice_public,
mallory_secret);
	printf("Mallory | Alice Private Key: %lu\n\n", mallory_alice_private);

	/*
	* ======= PROBLEM 2 =======
	* ======== PART 2 =========
	*/
	puts("======= PROBLEM 2 =======\n======== PART 2 =========");
	// MiTM attack for Bob
	long mitm_bob_secret = man_in_the_middle(params, bob_public);
	printf("What the MITM thinks Bob's Secret Value is: %lu\n", mitm_bob_secret);
	printf("Bob's Original Secret Value: %lu\n\n", bob_secret);

	// MiTM attack for Alice
	long mitm_alice_secret = man_in_the_middle(params, alice_public);
	printf("What the MITM thinks Alice's Secret Value is: %lu\n", mitm_alice_secret);
	printf("Alice's Original Secret Value: %lu\n\n", alice_secret);

	if (mitm_bob_secret == bob_secret)
	puts("The man in the middle attack was successful in retrieving Bob's secret");
	else
	puts("The man in the middle attack was not successful in retrieving Bob's secret");

	if (mitm_alice_secret == alice_secret)
	puts("The man in the middle attack was successful in retrieving Alice's secret");
	else
	puts("The man in the middle attack was not successful in retrieving Alice's secret");

	return 0;
}
```

Screenshots



```
/tmp/tmp.N7OEVcooAu/cmake-build-debug/test
======= PROBLEM 1 =======
======== PART 1 =========
Bob's Private Key: 3284
Alice's Private Key: 3284

Bob's Private Key matches Alice's Private Key

======= PROBLEM 2 =======
======== PART 1 =========
Mallory | Bob Private Key: 32847
Mallory | Alice Private Key: 69102

======= PROBLEM 2 =======
======== PART 2 =========
What the MITM thinks Bob's Secret Value is: 82388
Bob's Original Secret Value: 82388

What the MITM thinks Alice's Secret Value is: 75442
Alice's Original Secret Value: 75442

The man in the middle attack was successful in retrieving Bob's secret
The man in the middle attack was successful in retrieving Alice's secret

Process finished with exit code 0
```

Build finished in 4 sec, 172 ms (3 minutes ago)