```cpp
#include <iostream>
#include <vector>

//Function for calculating the module of a base with a huge power with a
 recursive approach
long long mod_big(long long base, long long power, int mod)
{
    // If we reach power of 0 we know that we have base^0 which is 1 and 1 mod
     anything is 1
    if (power == 0)
    {
        return 1;
    }

    // If power % 2 is == 1 we have an odd power so we multiply by base and do
     power - 1 so that we have an even power
    if(power % 2 == 1)
    {
        return (base * mod_big(base, power - 1, mod)) % mod;
    }

    // Keep on dividing power by 2 so that we can just square the result and
     take the mod so that we don't get overflow
    long long half = mod_big(base, power / 2, mod);

    return (half * half) % mod;
}

int main()
{
    std::srand(static_cast<unsigned int>(std::time(nullptr)));
    int min = 10000;
    int max = 100000;

    long long prime = 0;

    bool check = true;

    //Creates a random prime number between min and max
    do {
        check = true;
        prime = std::rand() % (max - min + 1) + min;
        for (int i = 2; i <= prime/2; i++)
        {
            if (prime % i == 0)
            {
                check = false;
            }
        }
    } while (check == false);
```

```
//Create vector for seen mods so that we can find a primitive root
std::vector<int> seen;
long long current = 0;
check = true;
long long primitive_root = 0;

long long test = 0;
std::vector<int> primitive_tests;
// Push back 0 and 1 since they can't be primitive roots and checking them
 would potentially waste time
primitive_tests.push_back(0);
primitive_tests.push_back(1);
//Gets a primitive root for the prime number
while(primitive_root == 0)
{
    //Picks a random number from 0 to prime-1 then checks if it is a
     primitive root
    test = rand() % prime;
    // While loop to pick a random number if the one picked has already
     been tested
    while(std::find(primitive_tests.begin(), primitive_tests.end(), test)
     != primitive_tests.end())
    {
        test = rand() % prime;
    }

    //Puts the number we just tested for primitive root into vector of
     seen primitive tested
    primitive_tests.push_back(test);

    // Go through from 1 to prime-1 to see if test^j has been seen yet; if
     all numbers j = 1 to prime-1 for test^j are different then test is a
     primitive root
    for (long long j = 1; j < prime; j++)
    {
        current = mod_big(test, j, prime);
        if (std::find(seen.begin(), seen.end(), current) != seen.end())
        {
            check = false;
            seen.clear();
            break;
        }
        else
        {
            seen.push_back(current);
        }
    }

    if (check == true)
    {
```

```
            primitive_root = test;
            break;
        }

        check = true;
    }

    //Prints out the prime number and the primitive root we are dealing with
    std::cout << "The prime number is: " << prime << std::endl;
    std::cout << "The primitive root for prime is: " << primitive_root <<
     std::endl;

    long long XA = 0;
    long long XB = 0;
    long long YA = 0;
    long long YB = 0;

    //Gets random number from 0 to prime-1 for XA and XB
    XA = rand() % prime;
    XB = rand() % prime;


    /* Uncomment if we want to input powers for XA and XB
    std::cout << "What would user A's power be which is less than " << prime
     << ": ";
    std::cin >> XA;
    std::cout << "What would user B's power be which is less than " << prime
     << ": ";
    std::cin >> XB; */


    // Calculate YA and YB so that we can calculate the keys KA and KB
    YA = mod_big(primitive_root, XA, prime);
    YB = mod_big(primitive_root, XB, prime);

    //Calculate KA and KB
    long long KA = mod_big(YB, XA, prime);
    long long KB = mod_big(YA, XB, prime);

    std::cout << "Without MitM Attack - Key for A: " << KA << std::endl <<
     "Key for B: " << KB << std::endl;


    //This section is for Problem 2 and simulates 2 attacks on Diffie Hellman

    // This simulates a MitM attack between A and B by M
    long long XM = 0;
    long long YM = 0;

    //Gets random number from 0 to prime-1 for XM
    XM = rand() % prime;
```

```
/* Uncomment if we want to input power for XM
std::cout << "What would user M's power be which is less than " << prime
 << ": ";
std::cin >> XM; */

// Calculates YM
YM = mod_big(primitive_root, XM, prime);

// Calculates the keys M, A, B with a MitM attack
long long KM_A = mod_big(YA, XM, prime);
long long KM_B = mod_big(YB, XM, prime);
KA = mod_big(YM, XA, prime);
KB = mod_big(YM, XB, prime);


std::cout << "With MitM Attack - Key for A: " << KA << std::endl << "M's
 Key with A: " << KM_A << std::endl << "Key for B: " << KB << std::endl <<
 "M's Key with B: " << KM_B << std::endl;


// This is if we know what YA, YB, primitive_root, and the prime number
 are we can find what XA and XB are
bool XA_check = false;
bool XB_check = false;

// Goes through all possible powers until we find one that equals YA or YB
 and then we know that those are XA and XB respectively since a is a
 primitive root of prime
for (int i = 0; i < prime; i++)
{
    if (mod_big(primitive_root, i, prime) == YA)
    {
        std::cout << "XA is: " << i << std::endl;
        XA_check = true;
    }

    if (mod_big(primitive_root, i, prime) == YB)
    {
        std::cout << "XB is: " << i << std::endl;
        XB_check = true;
    }

    if (XA_check == true && XB_check == true)
    {
        break;
    }
}


return 0;
```

```
}
```