Practice 5

Hard problems in cryptography

Discrete Logarithm problem & prime factors

Session lab 5 | Wednesday, May 6, 2020

By Joel Harim Hernández Javier

Description

Use a cryptographic library in your favorite programming language (C,C++, Java or Python) to solve the following exercises.

Programming Exercises

1. Develop a program to solve the discrete logarithm problem, for the following instances.
    a. $11^x \ mod \ 1009 \ = \ 400$
    b. $5^x \ mod \ 10007 \ = \ 5235$
    c. $2^x \ mod \ 100000000003 \ = \ 1922556950$
    d. $3^x \ mod \ 500000009 \ = \ 406870124$
    e. $3^x \ mod \ 500000009 \ = \ 187776257$
2. Develop a program to find the prime factors of the following composite numbers.
    a. 100160063
    b. 10006200817
    c. 250035001189
    d. 250000009000000081

Products

- ➔ Your personal information, date of the lab session and the topic that we are studying in this lab session.
- ➔ Briefly describe what you did to solve the exercises.
- ➔ The answer for each exercise.
- ➔ Screenshots, showing your program running.

Procedure

Discrete logarithm problem

$$base^x \bmod module = result$$

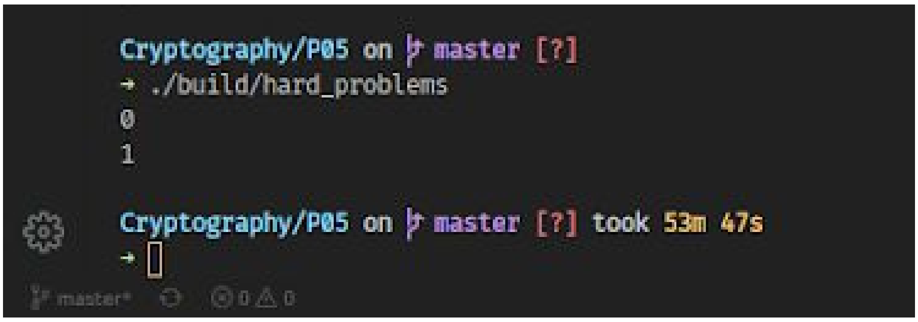
First of all I calculated the limits and the restrictions of the problem. I found the following ones:

$$2^0 \leq base \leq 2^4$$
$$2^0 \leq module \leq 2^{37}$$

Therefore:

$$2^0 \leq result \leq 2^{37}$$
$$2^0 \leq x \leq 2^{37}$$

My first approach with this problem was the brute force, trying all values from 1 to $module - 1$, using the `CryptoPP::Integer` class and `CryptoPP::a_exp_b_mod_c()` function, and the results weren't good (the first 10^9 tests for problem 2.c took almost an hour):



After many attempts and optimizations, I realized that this wasn't the way to solve the problem.

After investigating, my second attempt was with the `baby-step giant-step` algorithm. I adapted this C++17 code, using the `CryptoPP::Integer` class and `CryptoPP::a_exp_b_mod_c()`, since the original one overflowed the `uint64_t` variables.
https://en.wikipedia.org/wiki/Baby-step_giant-step#C++_algorithm_(C++17)

Final code:

```
// Computes x such that b^x % mod == result
// Based on
https://en.wikipedia.org/wiki/Baby-step_giant-step#C++_algorithm_(C++17)
ull babystep_giantstep(ull b, ull result, ull mod){

    const auto m = (ull)(std::ceil(std::sqrt(mod)));
    auto map = std::unordered_map<ull, ull>{};
```

```
    CryptoPP::Integer base = b, exp = (mod - m - 1), module = mod, e = 1;

    for (ull i = 0; i < m; ++i){

        map[integerToULL(e)] = i;
        e = a_times_b_mod_c(e, base, module);

    }

    const auto factor = a_exp_b_mod_c(base, exp, module);
    e = result;

    for (ull i = 0; i < m; ++i){

        auto it = map.find(integerToULL(e));
        if (it != map.end())
            return i * m + it->second;
        e = a_times_b_mod_c(e, factor, module);

    }

    return -1;

}
```

Execution

```
Cryptography/P05 on master [!]
→ ./build/hard_problems
Develop a program to solve the discrete logarithm problem, for the following instances:

11^x mod 1009 = 400 => x = 900
5^x mod 10007 = 5235 => x = 8900
2^x mod 100000000003 = 1922556950 => x = 50000000
3^x mod 500000009 = 406870124 => x = 400000000
3^x mod 500000009 = 187776257 => x = 500000000

Cryptography/P05 on master [!]
→
```

Answers

a. $11^x \; mod \; 1009 \; = \; 400$ $\quad x = 900$

b. $5^x \; mod \; 10007 \; = \; 5235$ $\quad x = 8900$

c. $2^x \; mod \; 100000000003 \; = \; 1922556950$ $\quad x = 50000000$

d. $3^x \; mod \; 500000009 \; = \; 406870124$ $\quad x = 400000000$

e. $3^x \; mod \; 500000009 \; = \; 187776257$ $\quad x = 500000000$

Prime factorization

First of all I calculated the limits and the restrictions of the problem:

$$2^0 \leq 250000009000000081 \leq 2^{58}$$

So, the ideal variable size is `unsigned long long` $(0, 2^{64}-1)$.

I realized that dividing the input variable while checking from 2 to $\sqrt{x}$, calculation steps would be skipped. Finally, last factor would remain in the input variable and it would be 1 or greater than 2:

```
// Get prime factors of n
std::vector<ull> getPrimeFactors(ull n){

    auto r = std::vector<ull>();

    // Separated for optimization
    if (!(n & 1))
        r.push_back(2);
    while (!(n & 1))
        n >>= 1;

    for (ull i = 3; i * i <= n; ++i){

        if (n % i == 0)
            r.push_back(i);
        while (n % i == 0)
            n /= i;

    }

    if (n > 2)
        r.push_back(n);

    return r;

}
```

Execution

```
Cryptography/P05 on ᚴ master [!]
→ ./build/hard_problems


Develop a program to find the prime factors of the following composite numbers:

Factors of 100160063:
{ 10007, 10009 }
Factors of 10006200817:
{ 100019, 100043 }
Factors of 250035001189:
{ 500029, 500041 }
Factors of 250000009000000081:
{ 500000009 }

Cryptography/P05 on ᚴ master [!] took 7s
→
```

Answers

a. 100160063 :
   {10007, 10009}
b. 10006200817 :
   { 100019, 100043 }
c. 250035001189 :
   { 500029, 500041 }
d. 250000009000000081 :
   { 500000009 }

The complete source code can be found at my personal Github 🙂
https://github.com/JoelHernandez343/Cryptography/tree/master/P05