

Practice 6

Hard problems in cryptography II

Breaking RSA

May 12, 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. Design an algorithm to find the private key in RSA, d , if you have the public key (e, n) . Write the pseudocode of your algorithm. Give an example to explain how it works.
2. Implement your algorithm. Take five public keys from those listed in google classroom and prove your algorithm.

Products

- Your personal information, date of the lab session and the topic that we are studying in this lab session.
- Pseudocode and your example for exercise 1.
- Screenshots, showing your program running.

Designing the algorithm

Explanation

RSA has the following algorithm to calculate the keys:

1. Take two enough big primes p and q .
2. Calculate $n = pq$
3. Calculate $O(n) = (p-1)(q-1)$
4. Find e such that $\gcd(e, O(n)) = 1$
5. Find $e^{-1} \bmod O(n)$ denoted by d

Then, as e and n are known, the steps required to know d are as follows:

1. Calculate p and q factoring n
2. Calculate $O(n) = (p-1)(q-1)$
3. Resolve $e * e^{-1} \equiv 1 \pmod{O(n)}$, i.e., find e^{-1} .

Pseudocode

```
algorithm breaking_rsa (e, n)
  (p, q) := find_prime_factors(n)
  o := (p - 1) * (q - 1)
  d := modular_inverse(e, o)

  output "The secret key is: ", d
```

Example

My exercise:

$(e, n) = (233471, 4510872627967657319)$, $d = 3333861988376362559$

Performing the algorithm:

$e = 233471$

$n = 4510872627967657319$

$(p, q) = \text{factors}(4510872627967657319) \Rightarrow (p, q) = (594004403, 7594005373)$

$o = (p-1) * (q-1) = (7594005373-1) * (594004403-1) = 4510872619779647544$

Using Euclid's algorithm:

$d = \text{modularInverse}(233471, 4510872619779647544) = 3333861988376362559$

End.

Implementation

```
// Calculate d given (e, n)
auto breaking_rsa(ull e, ull n) {

    std::cout << "Using e: " << e << "\nUsing n: " << n << "\n\n";

    auto [p, q] = find_prime_factors(n);
    std::cout << "p & q: " << p << " " << q << "\n";

    auto o = (p - 1) * (q - 1);
    std::cout << "o(n): " << o << "\n";

    return std::tuple(modular_inverse(o, e), o);
}
```

Test Cases (in.txt)

5

Brenda
1987 5963


LuisPavel
34567 23264323

HectorJair
1283 4258141

AngyFlores
188443 944159

Joel
233471 4510872627967657319

Execution

```
Cryptography/P06/build on  master [!?]
→ ./breaking_rsa < in.txt
=====
Classmate's name: Brenda
Using e: 1987
Using n: 5963

p & q: 67 89
o(n): 5808
d: 2923
Validating:  $e * d \bmod o(n) = 1$ .
=====
Classmate's name: LuisPavel
Using e: 34567
Using n: 23264323

p & q: 3323 7001
o(n): 23254000
d: 8616903
Validating:  $e * d \bmod o(n) = 1$ .
=====
Classmate's name: HectorJair
Using e: 1283
Using n: 4258141

p & q: 1987 2143
o(n): 4254012
d: 3159839
Validating:  $e * d \bmod o(n) = 1$ .
=====
Classmate's name: AngyFlores
Using e: 188443
Using n: 944159

p & q: 947 997
o(n): 942216
d: 942211
Validating:  $e * d \bmod o(n) = 1$ .
=====
Classmate's name: Joel
Using e: 233471
Using n: 4510872627967657319

p & q: 594004403 7594005373
o(n): 4510872619779647544
d: 3333861988376362559
Validating:  $e * d \bmod o(n) = 1$ .
```

The complete source code can be found at my personal Github 😊

<https://github.com/JoelHernandez343/Cryptography/tree/master/P06>