Cryptography: RSA Algorithm

Daniel Alvarado Bonilla
Technological Institute of Costa Rica
Computer Engineering
2014089192
Email: daniel.alvarado.bonilla@gmail.com

Roberto Rojas Segnini
Technological Institute of Costa Rica
Computer Engineering
2016139072
Email: rojassegniniroberto@gmail.com

***Abstract*—ACA VA EL ABSTRACT**

***Index Terms*—Big O Permutations.**

I. Introduction

The increasing growth of the Internet have brought the issue of privacy in electronic communication. Big volumes of private information are transmitted via electronic devices day. The possibility that another person can intercept, someway, somehow that data that is being sent is high. This is the reason why data should be encrypted, to ensure confidentiality and security of files, in other words knowledge.
Encryption is the average method for making a message that is being communicated private. Anyone wanting to send a confidential message to another user encrypts the message before transmitting it. Only the intended recipient knows how to properly decrypt the message. Any person who was ”spying” on the conversation would only see the encrypted message. Because they would not know how to decrypt it, the message would make no sense.This way privacy is devise.
The RSA, named after the three MIT profesors who invented the algorithm, Ron Rivest, Adi Shamir and Leonard Adleman. This algorithm was floor breaking at that time because of prime factorization. The Fundamental Theorem of Arithmetic states that any number greater then 1 can be written in exactly one way using two prime numbers. The reason this one of the most public encryption algorithm is because n, a large integer used in the algorithm later explained, is a product of two prime numbers, which are truly hard to obtain. In other words RSA derives its security because of the difficulty of factoring large integers that are the product of two large prime numbers.

II. Methodology

The algorithm to create the public and private keys is structure the next way:

1) Choose two very large random prime integers: say p and q
2) Compute n and (n): $n = pq$ and $(n) = (p-1)(q-1)$ (n) is Euler's Totient Function.
3) Choose an integer e, $1 < e < (n)$ such that: $gcd(e,(n)) = 1$ (where gcd means greatest common denominator)
4) Compute d, $1 < d < (n)$ such that: $ed1(mod(n))$ in other words $a * d =$

RSA encrypts ”messages” of limited size, the maximum size of data which can be encrypted with RSA is 245 bytes. No more.
In the following paper, for the experimentation purpose, different implementations of the RSA algorithm will be used. If two really big prime numbers are used to encrypt and decrypt, the d exponent, which is the private key, is going to be a massive calculation to do for the computer. Thus, two different ways of solving $(c^d)modn$ will be used in Python programming language. The first approach is a common $c**d\%n$ in the

III. Pseudocode

Algorithm 1 Encrypt RSA algorithm

```
1: function ENCRYPT($publicKey[e, n]$,plaintext )
2:
3:     m = plaintext
4:     ciphertext = $(m^e)modn$
5:     return ciphertext
6: end function
```

Algorithm 2 Decrypt RSA algorithm

```
1: function ENCRYPT($privateKey[d, n]$,ciphertext )
2:
3:     c = ciphertext
4:     m = $(c^d)modn$
5:     plaintext = m
6:     return plaintext
7: end function
```

IV. Algorithm Complexity

Big O, $O(f(n))$ of the RSA algorithm:
Este algoritmo tiene en sí bastantes funciones pequeñas. Cada una siendo fundamental para la función de poda. Las más relevantes se explicarán a continuación son:
$O(n^k)$

V. Experimentos

A. *Experimento 1*

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

Algorithm 3 Algorithm used for a Faster calculation of

```
function MODEXPO($c$,$d$,$n$ )
                                        ▷ returns c ** d (mod n)

    assert $d >= 0$
    assert $n >= 0$
    base2D = int2baseTwo(d)
    base2DLength = len(base2D)
    modArray = []
    result = 1
    for i in range of (1, base2DLength $+1$) do
        if $i = 1$ then
            modArray.append(a % n)
        else
            modArray.append((modArray$[i - 2] * *2$) % n)
        end if
    end for
    for  i in range of base2DLength do
        if base2D[i] $= 1$: then
            result $* =$ base2D[$i$]$*$ modArray[$i$]
        end if
    end for
    return result % n
end function
```

Algorithm 4 x is a positive integer. Convert it to base two as a list of integers in reverse order as a list.

```
function INT2BASETWO($x$ )

    assert $x >= 0$
    bitInverse = []
    while $x \neq 1$  do
        bitInverse.append($x$ & 1)
        x ¿¿ $= 1$
    end while
    return bitInverse
end function
```