King Abdulaziz University

EE-305

Uses of prime numbers in cryptography

Instructor: Dr. Emad Khalaf

Name: Hesham Banafa

ID: 1742275

Date: 20/4/2020

Table of contents

Abstract

In this paper, the importance of prime numbers in public key cryptography will be discussed. The focus will be on how large prime numbers are found and used in computing other values that are used in the RSA such as a key. standard and why the numbers should be large for security reasons. Furthermore, the relation between the values offer asymmetric qualities that are not used only for encryption, but verification of authors, senders or any entities holding a key such as a company or website.

Introduction

In today's world, with millions of users of users using the internet, the security of user data and messages should be secure form outsiders. Cyryptography is an essential tool for any sort of secure communication and is needed for keeping government secrests and documents safe from other entities trying to harm a nation. There are many kinds of encryption that help with this task. However, most methods require both sender and receiver to know a common password, phrase or key. This can become difficult to achieve because sending the "Secret" over a connection will give it away to anyone trying to listen in. To solve this problem, public key encryption became wildly used in almost all connections.

In essence, public key encryption, gives the freedom to send a "key" to anyone who wants to establish a secure connection without fear of insecurity. For example, The RSA encryption algorithm uses key pairs that are labeled "public" and "private". The public key is then used to encrypt a message or any sort of data, to then be only decrypted with the corresponding private key, making an asymmetric relation. The key pair can also be used for making cryptographic signatures by doing to opposite operation. The signature is encrypted with the message with the senders private key. Making it only readable by the senders public key. Having unlocked the signature with a given public key makes sure to verify the origin of the message.

Importance of prime numbers

Prime numbers are essential in the RSA algorithm due to their nature. Since prime numbers have only two divisors, they can be used to a key pair that can only be broken using prime factorization, which can become vary difficult with large numbers. Making prime numbers perfect for this application.

In the RSA system, two large prime numbers are randomly generated and used to make other calculations to produce the key pair. For example,

Let $p = 11$ and $q = 13$

Let $n = p*q = 143$

Then the set factors of n is $\{1, 11, 13, 143\}$

This property is used with numbers larger than 2^{4096}, which makes calculating p or q near impossible with today's computers.

Algorithms for finding primes

This section will go over common algorithms used to find prime numbers which vary in efficiency.

Book algorithm

The algorithm found in the book check for primility using a simple process.

The algorithim goes as follows for check x:

1. If 2 divides x, then the x is not prime. Otherwise continue.
2. Let K be a largest integer less than or equal the square root of x
3. Let $1 < D <= K$. Check if for all D if divides x. If so, x is not prime. Otherwise x is prime.

Python code:

```python
def isPrime(number):
    if number == 2:
        return True
    #if 2 devides number then num is not prime. pg.21
    if number % 2 == 0 or number == 1:
        return False
    #largest integer less than or equal square root of number (K)
    rootOfNum = math.sqrt(number)
    K = math.floor(rootOfNum)
    #Take odd D such that 1 < D <= K
    #If D devides number then number is not prime. otherwise prime.
    for D in range(1, K, 2):
        if D % 2 == 0 or D == 1:
            pass
        else:
            if number % D == 0 or number % 5 == 0:
                return False
    return True
```

This algorithm is simple and effective for values less then 2^{64} due to the nature of it counting up to the square root of the number in question. For larger values, it is recommended to use a different way.

Miller-Rabin algorithm

The Miller-Rabin method try's to find if, a given n, is not prime rather than 'is prime'. In other words, it find if the number is composite and is probabilistic. Under our tests, it holds vary well. The algorithm is as follows (Miller–Rabin primality test, 1967):

1. Given n is odd integer and n > 3
2. Let K = 10 (number of rounds)
3. Choose a; 1 < a < n -1
4. Write n - 1 = 2^s *d where d is odd (factor powers of 2 of n-1)
5. Evaluate the sequence a^d, $(a^d)^2$, $(a^d)^4$, ... , $(a^d)^{2^\wedge s} = a^{n-1}$ in mod n if a term in the sequence evaluates to 1, then n is a composite and we say 'a' is a 'witness for the compositeness'. Otherwise, continue with another 'a'.

6. If a term evaluates to -1, then n is probably prime.

The Miller-Rabin algorithm was tested and resulted in a much more efficient generation of prime numbers. It allows generation of primes with a size of 2^{4096} or more in seconds on a modern computer. It was chosen to be used in our program as a tool to generate prime numbers.

RSA standard

Key generation

The program generates a random number of a given bit length from user input. The random number is then tested with the Miller-Rabin test is used to verify it is a prime number. This step is done twice to find two large primes p, q. Then n is the product of those primes. Making the only factors {1, p, q, n}. From (RFC 8017 - PKCS #1, 2016), the required calculation to generate a key pair are as follows:

n is the public key.

Let e=65537 as it is the recommended value. Part of the public key.

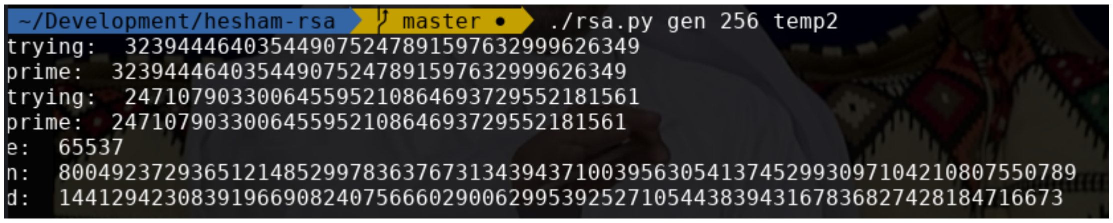
Then calculate phi = (p-1)*(q-1). Should be kept secret.

The private key is $d = e^{-1}$ (mod phi).

Now (n, e) are published to whomever is to send you encrypted messages.

And (d) is kept private to decrypt incoming messages.

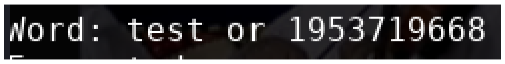
Example output of the program:



```
~/Development/hesham-rsa  master •  ./rsa.py gen 256 temp2
trying:  323944464035449075247891597632999626349
prime:  323944464035449075247891597632999626349
trying:  247107903300645595210864693729552181561
prime:  247107903300645595210864693729552181561
e:  65537
n:  80049237293651214852997836376731343943710039563054137452993097104210807550789
d:  14412942308391966908240756660290062995392527105443839431678368274281847l6673
```

Encryption

Due to the nature of RSA, the text needs to be in a numeric form. The program, thus, encodes the text into utf-8 bytes that are turned into an integer representing the word. Example output of the program:



```
Word: test or 1953719668
```

The number is then encrypted using the formula: $c = m^e$ (mod n) where m is the message and c is the cipher.

Decryption

The decryption process is the reverse of the encryption. However, the private exponent is needed for the calculation. Thus, we need (n, d) for the formula $m = c^d \pmod n$ where m is the decrypted message and c is the cipher.

Signing

Bibliography

En.wikipedia.org. 1967. *Miller–Rabin Primality Test*. [online] Available at: <https://en.wikipedia.org/wiki/Miller-Rabin_primality_test> [Accessed 20 April 2020].

Tools.ietf.org. 2016. *RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2*. [online] Available at: <https://tools.ietf.org/html/rfc8017> [Accessed 20 April 2020].