Estudio del tiempo de encriptación RSA según la complejidad de las llaves generadas

Ilustración 1, Breaking RSA encryption with math, Generada por Copilot Designer



Asignatura: Matemáticas: Análisis y Enfoques NM

Número de páginas:

Índice

[1. Introducción 3](#_Toc161771231)

[1.1 Elección del tema 3](#_Toc161771232)

[1.2 Contexto de la importancia de RSA 3](#_Toc161771233)

[2. Marco teórico 3](#_Toc161771234)

[2.1. Algoritmo de cifrado RSA 3](#_Toc161771235)

[2.2. Teoría de Números 3](#_Toc161771236)

[2.3. Funcionamiento de RSA: 3](#_Toc161771237)

[2.4. Prueba de Miller–Rabin de Primitividad 4](#_Toc161771238)

[2.4.1 Probables primos fuertes 4](#_Toc161771239)

[2.4.5 Elección de bases 4](#_Toc161771240)

[3. Diseño de programas de encriptación 5](#_Toc161771241)

[4. Resultados 5](#_Toc161771242)

[5. Modelización 6](#_Toc161771243)

[Anexos 6](#_Toc161771244)

[Bibliografía 7](#_Toc161771245)

# 1. Introducción

## 1.1 Elección del tema

## 1.2 Contexto de la importancia de RSA

# 2. Marco teórico

## 2.1. Algoritmo de cifrado RSA

El algoritmo de cifrado RSA, nombrado por sus creadores Ron Rivest, Adi Shamir y Leonard Adleman, representa un hito en la seguridad criptográfica. Basado en la complejidad de la factorización de números primos, RSA se ha convertido en un pilar fundamental para garantizar la confidencialidad en las comunicaciones electrónicas.

## 2.2. Teoría de Números

**[edit. notes] Fuente: chap2 page 63, Handbook of applied criptography**

El corazón de RSA está en la teoría de números, específicamente en la dificultad de factorizar el producto de dos números primos grandes. La factorización es un problema computacionalmente costoso y, hasta la fecha, no existe un algoritmo eficiente conocido que pueda factorizar grandes números en tiempo polinómico.

## 2.3. Funcionamiento de RSA:

1. Generación de Claves:
   1. Eliges dos números primos distintos grandes, y .
   2. Calculas el módulo de ;
   3. Calculas la función totiente de Euler: .
   4. Seleccionas un exponente público , que suele ser un número primo pequeño, como o . debe ser mayor que 1 y menor que .
   5. Calculas el exponente privado , que es el inverso multiplicativo modular del módulo . En otras palabras, .
2. Cifrado:
   1. Conviertes el mensaje en texto plano en una representación numérica , mediante la codificación ASCII, donde es un entero tal que .
   2. Utilizas la clave pública del destinatario para cifrar el mensaje.
   3. Calculas el texto cifrado (mod n).
3. Descifrado:
   1. El destinatario utiliza su clave privada para descifrar el texto cifrado .
   2. El destinatario calcula el mensaje original , y convierte el código ASCII en texto.

## 2.4. Prueba de Miller–Rabin de Primitividad

**[edit. notes] Pequeño Teorema de Fermat, Fuente:** [**Pequeño teorema de Fermat - Wikipedia, la enciclopedia libre**](https://es.wikipedia.org/wiki/Peque%C3%B1o_teorema_de_Fermat)

**[edit. notes] Primos fuertes, Fuente:** [**Número primo fuerte - Wikipedia, la enciclopedia libre**](https://es.wikipedia.org/wiki/N%C3%BAmero_primo_fuerte)

La prueba de Miller-Rabin de Primitividad es un algoritmo utilizado para determinar si un número dado es probablemente primo. Este algoritmo se basa en los siguientes conceptos matemáticos.

### 2.4.1 Probables primos fuertes

Dado un entero impar , escribimos como , donde y son enteros positivos y **d** es impar, mientras que es par.

Sea a un número entero tal que ;Diremos que es probablemente primo fuerte para la base si se cumple alguna de las siguientes relaciones:

* 1. para algún 0 ≤ r < s.

La idea detrás de la prueba de Miller-Rabin es que cuando es un primo impar, pasa la prueba debido a dos resultados:

1. Por el Pequeño Teorema de Fermat, .
2. Las únicas raíces cuadradas de módulo son y .

Si no es un probable primo fuerte para alguna base a, entonces es compuesto, y a se llama un testigo de la propiedad de que es compuesto. Sin embargo, esta propiedad no es una caracterización exacta de los números primos. Si es compuesto, puede haber una base a para la cual sea probable primo fuerte, en cuyo caso se llama un *pseudoprimo* fuerte y se lo denomina un *mentiroso* fuerte. La prueba probabilística de Miller-Rabin se basa en la comprobación para diferentes bases de que un número dado es probable primo fuerte.

### 2.4.5 Elección de bases

Afortunadamente, ningún número compuesto es un *pseudoprimo* fuerte para todas las bases al mismo tiempo. Sin embargo, no se conoce una forma sencilla de encontrar un testigo, una solución es probar todas las bases posibles, pero esto produce un algoritmo determinista ineficiente.

Otra solución es elegir una base al azar. Esto produce una rápida prueba probabilística. Que se basa en que cuando es compuesto, la mayoría de las bases son testigos, por lo que la prueba detectará como compuesto con una probabilidad razonablemente alta, mayor que 0.75.

## 3. Diseño de programa de encriptación

Para poder realizar el estudio del tiempo que se requiere para encriptar, diseñe un programa en Python, que encripta la frase “*Hello World!”*, que es la primera frase que todo programador usa para hacer su primer programa, esto permite realizar las mismas operaciones matemáticas solo variando en números de dígitos de los números primos usados para la encriptación.

Debido al enfoque matemático de este trabajo opte por no hacer el uso de bibliotecas de Python[[1]](#footnote-1) que realizan toda la encriptación de forma automática como *cryptography[[2]](#footnote-2)*, escribiendo un programa simple y altamente documentado que hace uso de “matemática pura” para realizar la encriptación y recoger las medidas de tiempo de computación; El código se puede en centrar en [GitHub](https://github.com/raf181/Trabajo-de-Campo-de-Matematicas/blob/main/Code/rsa_pure_math-(prime%20grab).py).

Debido a la complejidad e ineficiencia de generar los números primos en Python sin hacer uso de bibliotecas de encriptación, opte por desarrollar un programa en el lenguaje de programación Rust[[3]](#footnote-3) y haciendo uso de la prueba de primitividad de Miller–Rabin, esto permite generar los números primos con una longitud deseada. El código se puede en centrar en [GitHub](https://github.com/raf181/Trabajo-de-Campo-de-Matematicas/blob/main/prime-gen/src/main.rs), con las [listas](https://github.com/raf181/Trabajo-de-Campo-de-Matematicas/blob/main/prime-gen/src/main.rs) de primos ya generados.

## 4. Resultados

**[edit. notes] Cambiar colores y traducir los títulos**

Para simplificar la recopilación de datos decidí implementar una función en el programa de Python que guarda los tiempos directamente a un archivo csv, que permite graficar y realizar los cálculos necesarios. Por cuestiones de espacio solo incluiré algunos de los datos y cálculos, el resto de los datos se pueden encontrar en [excel](https://github.com/raf181/Trabajo-de-Campo-de-Matematicas/blob/main/key_length_results.xlsx) y los cálculos en anexos.

## 5. Modelización

La expresión de regresión polinómica general es la siguiente:

El sumatorio de los cuadrados que nos lleva al sistema de ecuaciones es el siguiente:

A través del método de mínimos cuadrados también se pueden determinar polinomios del grado que se desee. En este caso, la recta de regresión polinómica es de grado 3. Por lo tanto, se deberá resolver ecuaciones, ya que también se debe tener en cuenta el término independiente.

En nuestro caso , por lo que deberemos resolver un sistema con un total de 4 ecuaciones.

El sistema de ecuaciones quedará planteado de la siguiente manera:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | y | x |  |  |  |  |  |  |  |  |
|  | 8.06E-06 | 8 | 64 | 512 | 4096 | 32768 | 262144 | 7.08E-05 | 0.000566 | 0.004531 |
|  | 7.25E-06 | 9 | 81 | 729 | 6561 | 59049 | 531441 | 7.13E-05 | 0.000642 | 0.005774 |
|  | 8.18E-06 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 8.93E-05 | 0.000893 | 0.00893 |
|  | 1.11E-05 | 11 | 121 | 1331 | 14641 | 161051 | 1771561 | 0.000135 | 0.001481 | 0.016291 |
|  | 1.22E-05 | 12 | 144 | 1728 | 20736 | 248832 | 2985984 | 0.000158 | 0.001894 | 0.022723 |
|  | 1.22E-05 | 13 | 169 | 2197 | 28561 | 371293 | 4826809 | 0.000171 | 0.002227 | 0.028956 |
|  | 1.31E-05 | 14 | 196 | 2744 | 38416 | 537824 | 7529536 | 0.000201 | 0.002813 | 0.039376 |
|  | 1.76E-05 | 15 | 225 | 3375 | 50625 | 759375 | 11390625 | 0.000281 | 0.004214 | 0.063214 |
|  | 1.73E-05 | 16 | 256 | 4096 | 65536 | 1048576 | 16777216 | 0.000292 | 0.004669 | 0.074711 |
|  | 1.89E-05 | 17 | 289 | 4913 | 83521 | 1419857 | 24137569 | 0.000323 | 0.005485 | 0.093249 |
|  | 2.14E-05 | 18 | 324 | 5832 | 104976 | 1889568 | 34012224 | 0.000403 | 0.007251 | 0.13052 |
|  | 2.57E-05 | 19 | 361 | 6859 | 130321 | 2476099 | 47045881 | 0.000502 | 0.009545 | 0.181352 |
|  | 2.65E-05 | 20 | 400 | 8000 | 160000 | 3200000 | 64000000 | 0.000536 | 0.01072 | 0.2144 |
|  | 3.72E-05 | 21 | 441 | 9261 | 194481 | 4084101 | 85766121 | 0.00079 | 0.016599 | 0.348584 |
|  | 4.43E-05 | 22 | 484 | 10648 | 234256 | 5153632 | 1.13E+08 | 0.001014 | 0.022312 | 0.490873 |
|  | 5.34E-05 | 23 | 529 | 12167 | 279841 | 6436343 | 1.48E+08 | 0.001098 | 0.025254 | 0.580853 |
|  | 5.34E-05 | 24 | 576 | 13824 | 331776 | 7962624 | 1.91E+08 | 0.001146 | 0.027498 | 0.659958 |
|  | 0.000389 | 272 | 4760 | 89216 | 1758344 | 35940992 | 7.55E+08 | 0.007281 | 0.144065 | 2.964295 |

Para realizar, este sistema también es necesario una tabla de frecuencias con los valores necesarios para resolver este sistema. En este caso la tabla de frecuencias se presenta de esta manera:

Por lo tanto, quedará el siguiente sistema:

# Anexos

* Generador de números primos (Rust): [Archivo](https://github.com/raf181/Trabajo-de-Campo-de-Matematicas/blob/main/prime-gen/src/main.rs)
* Encriptado con matemáticas puras (Python): [Archivo](https://github.com/raf181/Trabajo-de-Campo-de-Matematicas/blob/main/Code/rsa_pure_math-(prime%20grab).py)

# Bibliografía

1. Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996. <https://cacr.uwaterloo.ca/hac/>
   1. [chap2.pdf (uwaterloo.ca)](https://cacr.uwaterloo.ca/hac/about/chap2.pdf)
   2. [chap4.pdf (uwaterloo.ca)](https://cacr.uwaterloo.ca/hac/about/chap4.pdf)
   3. [chap8.pdf (uwaterloo.ca)](https://cacr.uwaterloo.ca/hac/about/chap8.pdf)
2. colaboradores de Wikipedia. (2023, 29 septiembre). Test de primalidad de Miller-Rabin. Wikipedia, la Enciclopedia Libre. <https://es.wikipedia.org/wiki/Test_de_primalidad_de_Miller-Rabin>
3. Estadística, P. Y. (2023, March 1). Mínimos cuadrados. Probabilidad Y Estadística. <https://www.probabilidadyestadistica.net/minimos-cuadrados/>
4. ▷El Método de mínimos cuadrados: definición y ejemplos☑ - Mi Profe. (2022, February 22). Mi Profe. <https://miprofe.com/minimos-cuadrados/>

1. Welcome to Python.org. (2024, March 13). Python.org. [Link](https://www.python.org/) [↑](#footnote-ref-1)
2. cryptography. (2024, febrero 24). PyPI. [Link](https://pypi.org/project/cryptography/) [↑](#footnote-ref-2)
3. Rust, el lenguaje de programación. (n.d.). [Link](https://www.rust-lang.org/es) [↑](#footnote-ref-3)