Universidad Nacional Autónoma de México

Facultad de Ciencias

Proyecto 3

Modelado y Programación

Johann Ramón Gordillo Guzmán - 418046090

José Jhovan Gallardo Valdéz - 310192815

Gerardo Daniel Martínez Trujillo - 311314348

Proyecto presentado como parte del curso de **Modelado y Programación** impartido por el profesor **José de Jesús Galaviz Casas**.

23 de Octubre del 2019

Link al código fuente: https://github.com/JohannGordillo

1. Introducción

En el presente documento se presenta detalladamente el proceso de definición y solución de un problema de programación propuesto por el cliente, la **Secretaria de la Defensa Nacional (SEDENA)**: la implementación de un criptosistema RSA.

El criptosistema ya ha sido implementado por el cliente utilizando el lenguaje de programación **Python**. Sin embargo, se nos ha solicitado implementarlo en un lenguaje distinto, agregar pruebas unitarias para verificar su funcionamiento ante posibles modificaciones futuras, y agregarle tres mejoras útiles a criterio del programador.

Un **criptosistema** es el conjunto de procedimientos que garantizan la seguridad de la información, usando para ello técnicas y algoritmos criptográficos.

El criptosistema RSA (Rivest-Shamir-Adleman) es de **clave pública (asimetrico)**. Este tipo de criptosistemas se caracterizan por utilizar claves distintas para el cifrado y descifrado de la información, a diferencia de los criptosistemas de **clave privada (simétricos)**. En particular, en el algoritmo RSA, sus dos claves sirven indistintamente tanto para cifrar como para autenticar.

La seguridad de este algoritmo radica en el **problema de factorización de números enteros**, que se basa en el hecho de que en un número n de 1024 bits el método usualmente usado de descomponer este número en sus factores dividiéndolo por todos los enteros positivos entre dos es impráctico. Más aún, el problema de factorización de números enteros pertenece a **clase NP**.

El algoritmo consiste en tres pasos:

- **Generación de Claves.**
  Para este paso se eligen dos primos p, q de manera aleatoria, que sean distintos, lo suficientemente grandes y que tengan una longitud en bits similar.
  Con estos primos se forma el número $n = pq$. Posteriormente, escogemos un número e que sea primo relativo con $(p-1)$ y $(q-1)$. El par ordenado (e, n) conformará la clave pública del algoritmo, y pueden ser conocidos por cualquiera. Es importante que e sea menor a la función indicatriz de euler de n, es decir, que $\phi(n) = (p-1)(q-1)$, y que sea primo relativo con este mismo número.
  Al tener e un inverso mód $\phi(n)$, podemos denotarlo como d, y entonces formaremos un par ordenado (d, n) que será la clave privada del algoritmo. El número d se debe mantener en secreto.

- **Cifrado.**
  Supongamos que se tiene un mensaje m al que veremos como un número tal que $m < n$. Para obtener el mensaje cifrado c realizamos la operación $c = m^e (mod n)$.

- **Descifrado.**
  Para recuperar el mensaje m cifrado mediante el paso anterior, únicamente realizamos la operación $m = c^d (mod n)$.

El algoritmo presentado es bastante seguro, más no imposible de romper, por lo que se procurará ponerlo al límite y verificar que cumpla su función mediante pruebas unitarias.
Todos los detalles de implementación en un lenguaje de programación particular y la metodología usada para la eficiente solución del problema se especifican en este documento.

2. Definición del problema

Nuestro cliente, la **Secretaria de la Defensa Nacional (SEDENA)** nos ha solicitado elaborar un programa con tres métodos principales utilizando el criptosistema Rivest-Shamir-Adleman (RSA):

- Un método que genere las llaves: pública y privada.
  Este método deberá buscar números primos p y q distintos y aleatorios de al menos 100 dígitos.

- Otro método que cifre utilizando las llaves generadas en el punto anterior.
  Este método puede recibir como parámetros n y e generados con el método anterior y el texto que se va a cifrar m.

- Otro método que descifre utilizando las llaves generadas en el primer punto.
  Este método puede recibir como parámetros n, d y el texto cifrado usando el método anterior.

- Se pueden agregar los métodos auxiliares necesitados por el programador para implementar el criptosistema.

Además, se implementará una interfaz gráfica para seleccionar archivos, así como pruebas unitarias. Fuera de ello, al definición del problema no varia respecto a la definición planteada en la primera versión del proyecto.

3. Análisis del problema

3.1. Primera versión del proyecto

1. **Requisitos Funcionales**

   Dado un mensaje (una cadena) de a los más 257 carácteres, debemos implementar el algoritmo de encriptación RSA con cada uno de sus pasos. Debemos generar los primos p, q tales que nos permitan generar las claves públicas y privadas. Una vez obtenidas estas claves, se debe encriptar el mensaje de entrada. El mensaje previamente encriptado debe poderse desencriptar mediante un método de desencriptación, usando las claves generadas.

2. **Requisitos No Funcionales**

   El programa deberá poseer una **tolerancia a fallas**, ya que es de suma importancia que la implementación del algoritmo sea eficiente para garantizar que la información en forma de mensaje sea protegida.
   También deberá ser **escalable**, pues en un futuro se podrían implementar mejoras como una interfaz gráfica u otras maneras de obtener los mensajes a encriptar y desencriptar.
   Se cumplirá con la **eficiencia** requerida por el cliente, asegurándose el programa de que los mensajes sean encriptados de una manera rápida y con ahorro de recursos de la computadora. Análogamente con la operación de desencriptamiento del mensaje.
   De una manera más detallada, podemos definir el problema de la siguiente manera:

   a) **Entrada**
   Una cadena de texto, la cual será ingresada por el usuario a través de la consola.

      1) **Formato**
      Cadena de texto.

      2) **Tamaño**
      A lo más 257 carácteres.

      3) **Cantidad**
      Una cadena.

      4) **Fuente**
      Entrada estándar.

   b) **Salida**
   Dada una cadena cualquiera, se devuelve una cadena encriptada, misma que se puede desencriptar devolviendo la cadena original.

      1) **Formato**
      Cadena de texto.

      2) **Cantidad**
      Una cadena.

      3) **Destino**
      Salida estándar.

3.2. Segunda versión del proyecto

1. **Requisitos Funcionales**

   Dado un archivo de texto plano, debemos implementar el algoritmo de encriptación RSA con cada uno de sus pasos. Debemos generar los primos p, q tales que nos permitan generar las claves públicas y privadas. Una vez obtenidas estas claves, se debe encriptar el mensaje contenido en el archivo de texto ingresado. El mensaje previamente encriptado debe poderse desencriptar mediante un método de desencriptación, usando las claves generadas.

2. **Requisitos No Funcionales**

   El programa contará con una interfaz gráfica, con la cual se le permitirá al usuario final la selección de el archivo con el texto que se encriptará usando el algoritmo RSA. La interfaz deberá ser **amigable**, intuitiva y sencilla de usar.

   El programa deberá poseer una **tolerancia a fallas**, ya que es de suma importancia que la implementación del algoritmo sea eficiente para garantizar que la información en forma de mensaje sea protegida.

   También deberá ser **escalable**, pues a pesar de que ya se han implementado la iterfaz gráfica y el manejo de archivos, siempre es posible agregar mejoras al proyecto.

   Se cumplirá con la **eficiencia** requerida por el cliente, asegurándose el programa de que los mensajes sean encriptados de una manera rápida y con ahorro de recursos de la computadora. Análogamente con la operación de desencriptamiento del mensaje. Así como la obtención de datos por medio de la interfaz y la lectura y escritura de archivos.

   De una manera más detallada, podemos definir el problema de la siguiente manera:

   a) **Entrada**
   La entrada será texto plano, obtenida de un archivo de texto en el directorio de archivos del usuario. El archivo se seleccionará por medio de una interfaz gráfica (GUI) amigable e intuitiva.

      1) **Formato**
      Archivo de texto.

      2) **Tamaño**
      Texto de a lo más 257 carácteres.

      3) **Cantidad**
      Un archivo.

      4) **Fuente**
      Directorio de archivos del usuario final.

b) **Salida**
Dada una cadena contenida en un archivo de texto, se devuelve la misma cadena encriptada en un archivo de texto, misma que se puede desencriptar devolviendo la cadena original.

1) **Formato**
Archivo de texto.

2) **Cantidad**
Un archivo.

3) **Destino**
Directorio de archivos del usuario.

Como cambios, ahora en lugar de usar la entrada estándar para recibir una cadena del usuario, usaremos una interfaz gráfica que le permita seleccionar un archivo de texto, mismo que encriptaremos y colocaremos en otro archivo de texto.

El archivo de texto con la cadena encriptada podrá ser desencriptado usando el algoritmo y será devuelto a su estado original.

Sin embargo, ya no tendremos que preocuparnos solo por la eficiencia del algoritmo, si no también por la eficiencia del programa al obtener y escribir los datos en los archivos de texto dentro del directorio de archivos del usuario.

El criptosistema RSA siempre constará de los tres pasos definidos en la introducción del presente documento, por lo que únicamente cambiarán las mejoras hechas al proyecto y la cantidad y calidad de las pruebas unitarias, así como el lenguaje y la estructura organizacional del proyecto completo. También se procurará una documentación más detallada y comentarios más precisos dentro del código.

4. Selección de la mejor alternativa

Para la realización de la primera versión de este proyecto se usó el lenguaje de programación **Python**. Sin embargo, para la realización de la segunda y presente versión se ha decidido utilizar el lenguaje de programación **Rust**.

4.1. ¿Qué es Rust?

Rust es un lenguaje de programación compilado diseñado por **Mozilla**. Es un lenguaje multiparadigma, y soporta los siguientes paradigmas:

- Programación orientada a objetos.
- Programación funcional.
- Programación por procedimientos.

El compilador de Rust se llama **rustc** y usa LLVM como su back-end.

El principal objetivo de Rust es la creación de programas del lado del cliente y del servidor que se ejecuten en internet. Además, Rust está diseñado para tener un acceso seguro a la memoria de la computadora y no permite apuntadores nulos.

4.2. ¿Por qué Rust?

Una de las razones por las que se decidió usar Rust sobre otros lenguajes de programación, es su alto desempeño comparado con lenguajes interpretados, como el mismo Python en el que fue implementado el presente proyecto en el pasado.

Además, el multiparadigma de Rust nos permitirá programar tanto con orientación a objetos como con programación funcional, dándole versatilidad a nuestro código y la posibilidad de poder abordar el problema desde ángulos distintos.

Otra razón por la que se eligió este lenguaje es su constante crecimiento y popularidad, así como su comunidad de usuarios.

Rust es tan rápido como C y C++, pero es más seguro que ambos, además de que permite un acceso seguro a la memoria.

Como útlima razón, pero no menos importante, es la calidad de su documentación y la cantidad de tutoriales disponibles en internet para su aprendizaje.

4.3. ¿Por qué mejoraría ahora su proyecto?

En la primera versión del proyecto, el usuario tenía que escribir toda la cadena en la entrada estándar, lo que era un problema si la cadena tenía la longitud máxima permitida de 257 carácteres. Para probar el algoritmo con varias cadenas de dicha longitud, la tarea se realizaba de manera ineficiente.

En la versión que estamos desarrollando, la cadena se encuentra en un archivo de texto, mismo que será seleccionado por el mismo usuario dentro de su directorio de archivos con ayuda de una interfaz gráfica (GUI) amigable y estética. Así, el usuario ahorrará tiempo y esfuerzo de una manera significativa, al evitar tener que escribir las cadenas una por una en la entrada estándar.

Además, Rust provee de un mejor diseño que Python, todo en el lenguaje puede verse como una expresión, y el código es más ordenado ya que se evita el uso de identaciones para definir bloques de código y en su defecto se usan llaves como en C, C++ Y Java. Por si eso fuera poco, la eficiencia del programa aumentará considerablemente debido a que Rust es mucho más rápido que Python al ser un lenguaje de programación compilado.

5. Pseudocódigo

El primer algoritmo usado en la elaboración del proyecto es el Algoritmo de Euclides Extendido, que usamos para calcular el máximo común divisor de dos enteros dados, y además nos permite expresar al mismo como una combinación lineal.

5.1. Primera versión del proyecto

Algorithm 1 Algoritmo para calcular el MCD de dos números y expresarlo como combinación lineal.

```
1: procedure EuclidesExtendido(a,b)                          ▷ a, b enteros
       if b == 0 then
           return a, 1, 0
       end if
       d1, x1, y1 ← EuclidesExtendido(b, a %b)
       d ← d1
       x ← y1
       y ← x1 − (a//b) ∗ y1
       return d, x, y                                        ▷ d el mcd de a y b
```

Algorithm 2 Algoritmo para generar un número aleatorio impar de 1024 bits.

```
1: procedure GenerarPosiblePrimo(a,b)
       n ← ObtenNumeroAleatorio(1024)                        ▷ Calcula el número aleatorio
       n ← n | (1 << 1024 − 1) | 1
       return n
```

Algorithm 3 Algoritmo de Miller-Rabin para saber si un número n es primo o no, ejecutando la prueba k veces.

```
1: procedure EsPrimo(n,k)                                   ▷ n, k enteros
    if n == 2 or n == 3 then
        return True
    end if
    if n  1 == 0 then
        return False
    end if
    r ← 0
    d ← n − 1
    while d  1 == 0
        r ← r + 1
        d ← d/2
    end while
    for i ← 0 < k do
        a ← ObtenNumeroAleatorio(2, n-1)        ▷ Obtiene un número al azar entre 2 y n − 1
        x ← a^b (mod n)
        if x == 1 or x == n − 1 then
            continue
        end if
        for j ← 0 < r − 1 do
            x ← x^2 (mod n)
            if x == 1 then
                    return False
            end if
            if x == n − 1 then
                    break
            end if
        end for
        return False
    end for
    return True
```

Algorithm 4 Algoritmo para generar un número primo.

```
1: procedure GenerarPrimo
       p ← GenerarPosiblePrimo()                    ▷ Calcula un posible primo
       while not EsPrimo(p) do
           p ← GenerarPosiblePrimo()                ▷ Calcula un posible primo
       end while
       return p
```

Algorithm 5 Algoritmo para generar e y d del algoritmo RSA.

```
1: procedure GenerarED(phi)
       p ← GenerarPosiblePrimo()                    ▷ Calcula un posible primo
       g, d ← EuclidesExtendido(p, phi)
       d ← d % phi
       while g ≠ 1 or p > phi do
           p ← GenerarPosiblePrimo()
           g, d ← EuclidesExtendido(p, phi)
       end while
       return p, d
```

Algorithm 6 Algoritmo para encriptar un mensaje.

```
1: procedure Encriptar(m)
       p ← GenerarPrimo()
       q ← GenerarPrimo()
       phi ← (p − 1) ∗ (q − 1)
       d, e ← GenerarED(phi)                        ▷ Calcula las claves
       c ← m^e (mod n)
       return c
```

Con estos algoritmos es suficiente para encirptar un mensaje de una manera eficiente, obteniendo cada uno de los componentes necesarios por el algoritmo RSA, como son los números primos aleatorios de 1024 bits y la obtención de las claves pública y privada.

Sin embargo, para que el criptosistema esté completo también es necesario implementar un procedimiento que nos permite obtener el mensaje original que fue encriptado usando los algoritmos anteriores, es por ello que se implementa el método **desencriptar()** cuyo pseudocódigo se presenta en la siguiente página de este documento.

Algorithm 7 Algoritmo para desencriptar un mensaje.

1: **procedure** DESENCRIPTAR(c)
    $p \leftarrow$ GENERARPRIMO()
    $q \leftarrow$ GENERARPRIMO()
    $phi \leftarrow (p-1) * (q-1)$
    $d, e \leftarrow$ GENERARED(phi) ▷ Calcula las claves
    $m \leftarrow c^d \ (mod\ n)$
    return m

5.2. Segunda versión del proyecto

En la segunda versión del código, se implementan los mismos algoritmos definidos con anterioridad en el presente documento. Sin embargo, fue necesario implementar otros algoritmos para la implementación de la interfaz gráfica y para el manejo de archivos.

Algorithm 8 Algoritmo para encriptar un archivo de texto.

1: **procedure** ENCRYPT_FILE(filename, destination)
    max_lenght = 257
    my_rsa ← new RSA()
    content ← READFILE(filename)
    if LENGHT(content) > max_lenght **then**
        SPLIT_OFF(content)
    end if
    encrypted_content ← my_rsa.ENCRIPTAR(content)
    key ← my_rsa.GET_E()
    key ← key + "\n"
    key ← my_rsa.GET_N()
    path ← TO_STRING(destination)
    path ← path + "/encrypted_"
    path ← path + filename
    keypath ← TO_STRING(destination)
    keypath ← keypath + "/key.txt"
    WRITE_TO_FILE(path, encrypted_content)

Algorithm 9 Algoritmo para desencriptar un archivo de texto.

```
1: procedure DECRYPT_FILE(filename, keyfile destination)
       max_lenght = 257
       my_rsa ← new RSA()
       content ← READFILE(filename)
       keyf ← READFILE(keyfile)
       e ← 0
       n ← 0
       While Not EOF(Filename) do
           e ← READLINE()
           n ← READLINE()
       End While
       message ← my_rsa.DESENCRIPTAR_CON_CLAVE(content, e, n)
       path ← TO_STRING(destination)
       path ← path + "/plain_"
       path ← path + filename
       WRITE_TO_FILE(path, message)
       Return
```

6. Pruebas y mejoras

6.1. Pruebas

Cada uno de los componentes del algoritmo es de suma importancia, por lo que el equipo de desarrolladores decidió implementar las siguientes pruebas:

- **Prueba del Algoritmo de Euclides Extendido**
  Este método calcula el maximo común divisor de una pareja de enteros y lo expresa como una combinación lineal.
  El método es de suma importancia para el proyecto, pues se usa en la obtención de las claves pública y privada del algoritmo, por lo que se realizan pruebas sobre él.

- **Prueba de generación de números Primos**
  En el programa se incluye un método para la generación de primos, sin embargo hay que asegurarse de que el número entero devuelto por el método sea efectivamente un primo, ya que si no se están devolviendo primos, todo nuestro algoritmo puede correr peligro y no será seguro el resguardo de la información. Para probar que el número dado sea un primo, se usará el algoritmo para prueba de primalidad de Miller-Rabin definido en el siguiente punto.

- **Prueba de primalidad: Algoritmo de Miller-Rabin**
  Para asegurarnos que un número sea primo hay muchos algoritmos, pero uno de los más eficientes es el algoritmo de Miller-Rabin implementado en el presente proyecto. Sin embargo, se tiene que probar su funcionalidad pasándole números primos y esperando una respuesta positiva, así como pasándole números compuestos y esperando una respuesta negativa.

- **Prueba de encriptamiento y desencriptamiento de mensajes**
  Es la prueba más importante del proyecto, pues se encargará de que dado un mensaje, éste se encripte de manera correcta, y posteriormente de desencripte de la misma manera, dando lugar al mensaje original.

6.2. Mejoras

La versión anterior del presente proyecto ha sido muy básica y se nos ha solicitado implementar tres mejoras, las cuales han sido seleccionadas minuciosamente por los desarrolladores y son las siguientes:

- **Manejo de archivos**
  En la implementación anterior, el mensaje a ser encriptado por nuestro algoritmo se ingresaba por medio de la entrada estándar. Sin embargo, esto es bastante tedioso para mensajes largos, por lo que se ha decidido implementar manejo de archivos y que la cadena del archivo de entrada sea la que se encripte, dando lugar a otra cadena en un archivo de texto de salida que posteriormente podrá ser desencriptado usando uno de los métodos implementados en el presente proyecto.

- **Interfaz gráfica (GUI)**
  Para facilitar aún más el trabajo del usuario final, se ha decidido implementar una interfaz gráfica que permita al usuario seleccionar su archivo de texto a encriptar desde su directorio de archivos. La interfaz será amigable y visualmente estética.

- **Manejo de keys y de mensajes mayores a 257 carácteres**
  Se ha implementado la generación de un archivo con las key (o bien, claves) para el uso de la interfaz gráfica que nos permita usar el algoritmo. Además, la primera implementación no permitía el manejo de mensajes con una longitud mayor a 257 carácteres, por lo que se decidió escalar el proyecto para permitir el manejo de estas longitudes dentro de los archivos de texto.

7. Análisis estadístico

Después de correr el programa 1,000 veces mediante un script de Bash, el tiempo promedio que se tomó fue de 0.1784 *segundos*.

8. Situación extraordinaria

Al ser un algoritmo altamente eficiente y confiable, tiene una gran tolerancia a errores, por lo que no fue posible hacerlo fallar. Su implementación está basada en teoría matemática y computacional altamente estudiada, así como su funcionamiento.

Sin embargo, el programa anterior fallaba al ingresar mensajes con más de 257 carácteres, pero en nuestra implementación, cuando en uno de los archivos a encriptar se supera el número de carácteres permitidos, lo que se hace es eliminar los carácteres sobrantes y tomar unicamente los primeros 257.

De esta manera, el algoritmo funciona a la perfección, aunque en un inicio se supere el máximo número de carácteres permitidos.

Otra manera en la que fallaba la primera versión del proyecto, era cuando el mensaje era una secuencia de carácteres no válidos, pero en esta última implementación, se usa el manejo de errores para mostrarle al usuario que fue lo que ocurrió.

9. Plan a futuro

9.1. Nuevas Funcionalidades

El presente proyecto fue diseñado con la finalidad de cumplir su función solicitada, pero además cuenta con un diseño que lo provee de escalabilidad, por lo que podremos agregar más mejoras en un futuro en caso de ser solicitadas por el cliente. Algunas de las posibles funcionalidades que pueden ser implementadas al proyecto son la posibilidad de enviar el mensaje encriptado a través de una red, implementar nuevas pruebas de primalidad de enteros (como la prueba de Fermat, por ejemplo) y no limitar el número de carácteres en la cadena dentro de los archivos de texto.

9.2. Bugs

Ningún programa es libre de bugs, y el equipo de desarrolladores está consciente de ello, por lo que ante descubrimientos de bugs y errores en el código, los programadores se encargarán de corregirlos y agregarlos a las pruebas unitarias. Y de ser necesario, se crearán pruebas de integración.

9.3. Eficiencia

Si los desarrolladores desubren una solución más eficiente para el problema, se le notificará al cliente para estimar el costo de nuevas implementaciones. Rust es un lenguaje de programación muy rápido, sin embargo el programa puede ser implementado sin problemas en C y C++ también conocidos por su excelente eficiencia.

9.4. Escalabilidad

Si el cliente necesita que se procesen más archivos, incluso se puede implementar cómputo paralelo o cómputo concurrente para el procesamiento de dichos archivos, y para su selección basta con haces modificaciones a la interfaz gráfica y a la manera en que esta funciona.

10. Distribución del trabajo

- **Johann Gordillo**

  - Elaboración del reporte del proyecto.

  - Elaboración del README.

  - Métodos para encriptar y desencriptar.

- **Jhovan Gallardo**

  - Elaboración de pruebas unitarias.

  - Implementación del algoritmo RSA.

- **Gerardo Martínez**

  - Elaboración de la interfaz gráfica.

  - Implementación de procesamiento de archivos.

11. Costo final del proyecto

El costo final del proyecto fue calculado con base en las horas que el equipo de desarrollo responsable de la tarea tuvo que invertir para verificar que la solución al problema planteado por el cliente, **Secretaria de la Defensa Nacional**, fuera eficiente; y el código de la implementación, robusto.

Se implementaron pruebas para que, en un futuro, desarrolladores que trabajen en el proyecto puedan implementar nuevas funcionalidades y aún así verificar si el código sigue funcionando correctamente bajo ciertos casos planetados estrategicamente por los desarrolladores.

El costo final es de **$2,500.00** pesos mexicanos.

Para aclaraciones del proyecto y del costo del mismo, contactar con **Mat. Johann Gordillo** al correo: **jgordillo@ciencias.unam.mx**

12. Bibliografía

- Universidad Politécnica de Madrid. (s.f). *Criptosistemas de clave pública. El cifrado RSA*. Recuperado el 22 de Octubre de 2019 de: http://www.dma.fi.upm.es/recursos/aplicaciones/matematica_discreta/web/aritmetica_modular/rsa.html

- Milanov, E. (2009). *The RSA Algorithm*. Universidad de Washington. Seattle, Washington: Estados Unidos de América. Recuperado el 23 de octubre de 2019 de: https://sites.math.washington.edu/ morrow/336_09/papers/Yevgeny.pdf

- García, J. (2019). *How does RSA work?*. Hackernoon. Recuperado el 23 de octubre de 2019 de: https://hackernoon.com/how-does-rsa-work-f44918df914b

- García, C. (2019). *Notas del curso de criptografía: RSA*. Facultad de Ciencias - UNAM. Ciudad de México: México.