UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Comparación de consumo energético en algoritmos de llaves asimétricas en redes IoT

TESIS

Que para obtener el título de

Ingeniera en Telecomunicaciones

P R E S E N T A

Tamayo González Cinthya Celina

DIRECTOR DE TESIS

Dr. Luis Francisco García Jiménez

Ciudad Universitaria, Cd. Mx., 2020

Agradecimientos

Agradezco especialmente a mi abuelo, por su constante apoyo que me permitió llegar a cumplir esta meta.

También agradezco a los demás miembros de mi familia que siempre me impulsaron a seguir adelante, a mis amigos y compañeros de ingeniería por todo su cariño, especialmente a Abraham, Alejandro, Estefania, Elizabeth y Rodrigo.

Asimismo, las aportaciones y recomendaciones que surgieron durante la revisión de este trabajo para mejorarlo por parte los sinodales. Por otro lado agradezco el apoyo brindado por parte del proyecto DGAPA-PAPIIT IA105520

Índice general

Índice de figuras

Índice de tablas

Resumen

Internet de las cosas (IoT) es un paradigma que permite que objetos cotidianos interactúen entre sí y puedan comunicarse globalmente mediante el uso de Internet. Muchos de estos objetos, suelen ser dispositivos alimentados por baterías de baja capacidad de carga que requieren un uso eficiente de energía con el fin de prolongar la vida útil de la red.

La información que se transmite en este tipo de dispositivos puede ser vulnerada si no se utiliza un protocolo que garantice la privacidad de la información. Por ello, el cifrado de datos es una tarea crucial en la comunicación entre estos dispositivos. Uno de los algoritmos de cifrado más utilizados es RSA, un protocolo de llave pública que garantiza la privacidad en el envío de la información. Sin embargo, actualmente el tamaño de las llaves se recomienda que sea de al menos $2048\,\text{bits}$ [1]. Esto lo hace costoso para dispositivos que suelen tener pocos recursos en procesamiento, memoria y energía, ya que la complejidad computacional de RSA demanda el uso continuo del dispositivo. Por consiguiente, el consumo excesivo de potencia ocasionaría una degradación en el tiempo de vida de la batería.

Últimamente se han desarrollado algoritmos de cifrado basados en teoría de fractales que proveen la misma seguridad que RSA, pero con llaves de menor tamaño. Un ejemplo es el algoritmo basado en el fractal de Mandelbrot, el cual basa su fortaleza en seleccionar números complejos que pertenecen al conjunto de Mandelbrot. Este tipo de algoritmos requieren de llaves de menor longitud y un procesamiento más sencillo que los hacen muy atractivos para el uso de redes de sensores, ya que ofrecen un sistema robusto contra ataques pero con un menor procesamiento que se traduce en un menor consumo energético en las redes IoT.

En esta tesis se implementa el algoritmo de llave pública basado en el fractal de Mandelbrot y se compara con el algoritmo RSA con el fin de cuantificar cuál de ellos es más factible de implementar en dispositivos que tienen recursos limitados de memoria, energía y procesamiento. Específicamente, se implementan ambos algoritmos en el dispositivo ESP8266; un equipo muy utilizado en redes de sensores por su conectividad, costo económico y capacidad de procesamiento.

Capítulo 1

Introducción

Las redes IoT están compuestas por dispositivos que tienen recursos limitados de memoria, batería y capacidad de procesamiento. Por esta razón, los recursos de estos dispositivos se deben utilizar de una manera óptima. Uno de los ámbitos donde se planifica con especial atención los recursos de estos dispositivos es en el uso de la energía, dado que muchos de estos dispositivos son alimentados con baterías de baja capacidad de carga.

Otro aspecto importante en la comunicación de una red, es la seguridad de la información, puesto que si no se protege la comunicación con un algoritmo de cifrado, cualquier persona ajena a este sistema puede saber el contenido de la comunicación. Uno de los algoritmos de cifrado más usados es RSA, un algoritmo de llave asimétrica que basa su fortaleza en la complejidad computacional de factorizar un número primo muy grande. Actualmente se recomienda que las llaves utilizadas sean de al menos $2048\,\text{bits}$. Sin embargo, utilizar este tipo de algoritmos en dispositivos IoT con memoria y unidades de procesamiento considerablemente limitados puede no ser óptimo. Especialmente, porque la complejidad computacional de RSA demanda el uso continuo del dispositivo IoT. Por consiguiente, el consumo excesivo de potencia ocasionaría una degradación en el tiempo de vida de la batería.

Últimamente, se han desarrollado algoritmos de cifrado basados en fractales que buscan ser una alternativa. Su fortaleza se encuentra en la recursividad infinita que construye el fractal. Algunos ejemplos son el algoritmo basado en el fractal de Mandelbrot que utiliza números complejos para calcular las llaves o el algoritmo basado en el triángulo de Sierpinski, que coloca la llave en los diferentes triángulos que lo conforman.

Esta tesis tiene el propósito de hacer una comparativa en términos de eficiencia de energía y tiempo de procesamiento entre los algoritmos RSA y Mandelbrot para determinar cuál de ellos es más factible de usar en un dispositivo IoT. Específicamente, para lograr este objetivo, se implementan ambos algoritmos en el lenguaje de alto nivel MicroPython en el dispositivo IoT *ESP8266*. Este dispositivo es muy usado en las redes IoT por su capacidad de conectarse con diferentes tipos de sensores que miden temperatura, humedad, movimiento, entre otros y conectarse a través de su módulo Wifi para enviar esta información a un registro en la nube, por correo electrónico o mostrar la información en las redes sociales Facebook y Twitter. También permite una conexión remota con este dispositivo a través de la nube para controlar motores o pequeños robots, lamparas y puertas inteligentes [2].

1.1. Meta

Comparar cuál de los algoritmos entre RSA y el fractal de Mandelbrot tiene una mejor eficiencia energética y a su vez un menor tiempo de procesamiento en el dispositivo IoT ESP8266.

1.2. Hipótesis

El uso de cifrado basado en fractales representa una disminución del tiempo de procesamiento en comparación con el cifrado basado en el algoritmo de RSA y por ende una mejora en

la eficiencia de energía.

1.3. Metodología

El desarrollo de este proyecto consta de las siguientes etapas:

1. Programar el algoritmo RSA en el lenguaje Python y comprobar su correcto funcionamiento.
2. Programar el algoritmo basado en el fractal de Mandelbrot en el lenguaje Python y comprobar su correcto funcionamiento.
3. Instalar MicroPython en el dispositivo ESP8266 e implementar ambos algoritmos con el fin de medir la eficiencia energética en cada uno.
4. Realizar el análisis comparativo entre RSA y Mandelbrot y obtener resultados.

1.4. Contribución

Comparar dos algoritmos de cifrado asimétrico con el fin de cuantificar cuál de ellos es más factible de usar en dispositivos que tienen pocos recursos energéticos y de procesamiento.

1.5. Descripción del contenido

En el Capítulo 2, se describen los tipos de algoritmos de cifrado según sus características y se explica brevemente el funcionamiento de los algoritmos más utilizados.

En el Capítulo 3, se describe el funcionamiento del algoritmo RSA y se ejemplifica el proceso de cifrado y descifrado.

En el Capítulo 4, se describen las propiedades del fractal de Mandelbrot y el fractal de Julia. Posteriormente, se explica el algoritmo de llave pública de Mandelbrot y se ejemplifica el cifrado y descifrado.

En el Capítulo 5, se explica de forma breve las características de MicroPython y el dispositivo IoT ESP8266. Posteriormente, se desarrollan las pruebas y se explican los resultados obtenidos.

En el capítulo 6, se presentan las conclusiones, la verificación de la hipótesis y las perspectivas de investigación.

Capítulo 2

Antecedentes

Con el auge de los sistemas de comunicación, surgió la necesidad de proteger la información, ya que terceros no autorizados pueden interceptar la comunicación y ver su contenido. Para ofrecer esta seguridad se originan los algoritmos de cifrado. La palabra criptografía tiene sus raíces en los vocablos griegos *kryptos* que significa ocultar y *grafos* que significa escritura, de esta manera su unión hace referencia a *escritura oculta* [3]. Actualmente conocemos como criptografía a la ciencia que provee confidencialidad, integridad y autenticación en la comunicación.

Usualmente se clasifican los métodos de cifrado por sus características o por cómo cifran los textos. La clasificación más común se basa en el tipo de llaves que utilizan para cifrar y descifrar. En base a este criterio se tienen dos tipos de cifrado, simétricos y asimétricos que se explican a continuación [4].

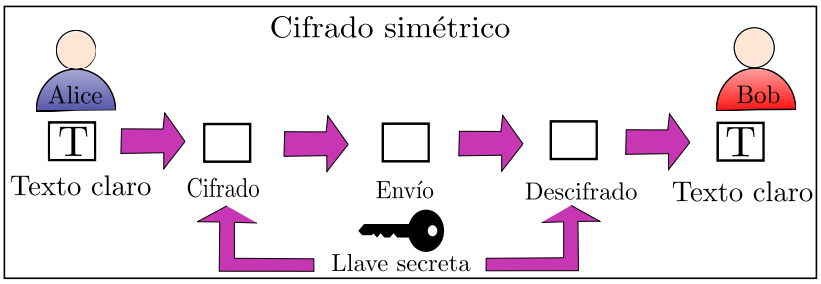
2.1. Cifrado simétrico

El cifrado simétrico tiene la característica de usar la misma llave para cifrar y descifrar el mensaje, para ello el transmisor y el receptor deben tener conocimiento de la llave, lo que implica la necesidad de un canal seguro para el intercambio de la misma. Como se observa en la figura 2.1 (a), donde el transmisor (Alice) aplica un algoritmo de cifrado al texto en claro usando la llave compartida. Cuando el mensaje cifrado llega al receptor (Bob), éste usa la llave compartida y el algoritmo inverso para descifrar el mensaje. La principal desventaja del cifrado simétrico es la dependencia de la llave compartida, ya que si uno de ellos pierde o revela el secreto, la seguridad de la información queda comprometida. Por esta razón, es recomendable cambiar frecuentemente la llave compartida. Algunos de los algoritmos simétricos conocidos son DES, 3DES, RC4, Blowfish y AES.

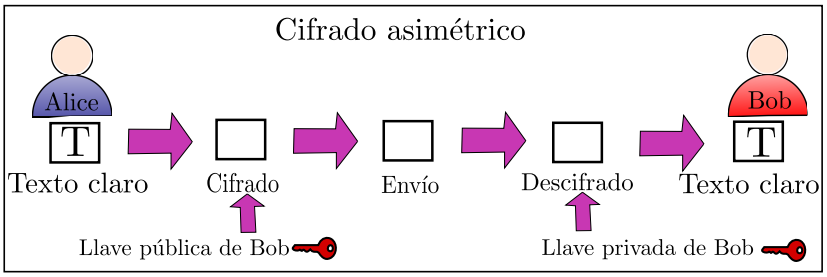
DES [5] fue desarrollado por IBM y aceptado como estándar en 1974 por la National Institute of Standars and Technology (NIST). DES toma el texto plano y lo divide en bloques de $64\,\text{bits}$. Cada bloque de texto plano se divide en 2 bloques de $32\,\text{bits}$. Uno de estos bloques se expande a $48\,\text{bits}$ para ser combinado con la llave original de $48\,\text{bits}$ mediante una XOR. El resultado de esta operación entra a una caja llamada S que comprime los $48\,\text{bits}$ a un bloque de $32\,\text{bits}$. Estos $32\,\text{bits}$ entran a una caja llamada P que realiza permutaciones entre los bits. El resultado de la caja P se mezcla con el bloque original de $32\,\text{bits}$ de texto plano mediante una XOR. Todo este proceso se repite 16 veces. Sin embargo, en 1988, la Electronic Frontier Foundation (EFF) publicó que había sido capaz de romper DES con una máquina especializada que tardaba menos de 3 días.

3DES [6] fue desarrollado por Tuchman en 1978 y tiene su origen como una variante más fuerte de DES. Esencialmente consiste en aplicar tres veces DES con la misma o con diferentes llaves. La ventaja de 3DES es que puede tener la fuerza de un cifrado con una llave de $168\,\text{bits}$, en lugar de una llave simple de $56\,\text{bits}$.

RC4 [7] fue desarrollado por Ron Rivest en 1987. Este cifrado calcula una subllave con un generador pseudoaleatorio que toma como semilla la llave principal. Posteriormente cifra el



(a) Cifrado simétrico.



(b) Cifrado asimétrico.

Figura 2.1: Tipos de cifrado.

mensaje mediante una operación XOR y la subllave calculada. Este algoritmo se hizo público en 1994 y es utilizado en el cifrado WEP.

AES [8] fue desarrollado por Vincent Rijmen y John Daemen con el fin de tener un reemplazo de DES. Su nombre original era Rijndael pero fue cambiado por la NIST en 2001. Este algoritmo divide el texto plano en bloques de 128 bits. Puede tener llaves de longitud variable que van desde 128 , 192 y 256 bits. Dependiendo la longitud de la llave, AES realiza diferentes rondas para cifrar la información. 10 rondas para 128 bits, 12 rondas para 129 bits y 14 rondas para 256 bits. Por ejemplo, para una llave de 128 bits, el texto plano es acomodado en una matriz de 4×4 conocida como matriz de estado. Las operaciones a esta matriz se realizan con palabras de 32 bits. El cifrado empieza con la ronda 0, conocida como AddRoundKey, que es una XOR entre la matriz de estado y la clave principal. Posteriormente se realizan las siguientes 10 rondas. Para cada ronda se calcula una subclave generada a partir de la clave principal. En las primeras 9 rondas se realizan las operaciones conocidas como SubBytes, ShiftRows, MixColumns y AddRoundKey. En la última ronda solo se realizan las operaciones SubBytes, ShiftRows y AddroundKey.

Blowfish [9] fue desarrollado y publicado por Bruce Schneier en 1993 como un algoritmo de cifrado simétrico, con la característica de ser más rápido que DES. Este algoritmo toma bloques de 64 bits de texto plano que puede cifrar con llaves cuyo tamaño varían desde los 32 hasta los 448 bits. La clave principal es utilizada para generar 18 subclaves que conforman el arreglo P. Similar a DES, Blowfish aplica 16 rondas para cifrar. El texto plano de 64 bits es dividido en 2 bloques de 32 bits, conocidos como XL y XR, respectivamente. En cada ronda se calcula un nuevo XL y un XR. Para el caso de XL se aplica una XOR al bloque XL anterior con la función π. Mientras que la nueva XR se crea mediante la operación conocida como F. El resultado de este proceso se le aplica una XOR con la XR anterior.

Otro ejemplo de cifrado simétrico pero esta vez basado en fractales es el algoritmo basado en el fractal de Sierpinski [10], toma el texto plano y lo inserta de manera recursiva bit a bit en los triángulos externos del fractal, mientras que la llave la inserta en los triángulos centrales bit a bit. Posteriormente, recorre cada uno de los bits del mensaje y dependiendo su valor realiza un desplazamiento al rededor del triángulo exterior, si es 0 gira hacia la derecha y si es 1 hacia la izquierda. De forma similar recorre los valores de la clave y dependiendo su valor realiza una XOR o XNOR, si es 0 o 1, respectivamente. El texto cifrado se obtiene de tomar la información mediante una secuencia recursiva previamente compartida por el transmisor y el

receptor.

2.2. Cifrado asimétrico

El cifrado asimétrico también se le conoce como cifrado de llave pública. Para este tipo de cifrado tanto el transmisor como el receptor tienen dos llaves complementarias; una llave pública que cualquiera puede conocer y una llave privada que solamente debe ser conocida por el propietario. Estas características se muestran en el diagrama de cifrado asimétrico en la figura 2.1(b). Para cifrar el mensaje, el transmisor (Alice) toma la llave pública del receptor (Bob) y cifra su mensaje con esta llave, posteriormente el receptor (Bob) usa su llave privada para descifrar el mensaje enviado por Alice. Actualmente, se recomienda que las llaves tengan al menos una longitud de $2048\,\text{bits}$, lo que implica una cantidad considerablemente grande de posibilidades (2^{2048}) si se hace un ataque por fuerza bruta. No obstante, la seguridad no solo recae en la longitud de las claves, si no también en la complejidad algorítmica, ya que se sabe que estos problemas no tienen una solución polinomial conocida. Algunos de estos problemas son:

- Factorización de un número primo muy grande.
- Logaritmo discreto, es decir, obtener el exponente al que ha sido elevado un número.
- Calcular los elementos utilizados en funciones equivalentes a la suma en un grupo de curvas elípticas.

El cifrado asimétrico elimina la necesidad de un canal seguro para la transmisión de una llave compartida. Sin embargo, este tipo de cifrado generalmente usa algoritmos con llaves muy grandes que conlleva a una mayor complejidad computacional. Algunos de los algoritmos más usados de llave pública son RSA [11] y DSA [12].

DSA fue presentado por la NIST en 1991 y adoptado como estándar por el gobierno de los Estados Unidos en 1993 para su uso en firma electrónica basado en llaves públicas. También dentro de los algoritmos de cifrado asimétrico se encuentran los algoritmos basados en fractales, como el algoritmo de llave pública del fractal de Mandelbrot [13]. En el Capítulo 3 y 4 se describe y ejemplifica el algoritmo RSA y Mandelbrot, respectivamente.

Se han realizado análisis comparativos entre los algoritmos previamente mencionados [14, 15]. Los resultados de estas pruebas muestran que RC4 y Blowfish son más rápidos en su proceso de cifrado que DES, a su vez DES es más rápido que AES tanto en cifrado como en descifrado. También observaron que RSA es el algoritmo que más tiempo requiere en cifrado. Por otro lado, la cantidad de memoria que utiliza AES para cifrar un texto es casi tan costoso como RSA, a pesar de que en las pruebas se utilizaron 256 bits para AES y 1024 bits para RSA.

El algoritmo basado en el triángulo de Sierpinski presenta casi la misma fortaleza que el algoritmo DES [10]. Mientras que DSA es más rápido que RSA en cifrado [15]. Finalmente, el algoritmo basado en el fractal de Mandelbrot es más rápido en su proceso de descifrado comparado con RSA [13].

Capítulo 3

RSA

En este capítulo se explica el procedimiento de generación de llaves, cifrado y descifrado en RSA. Además, se presenta un ejemplo numérico del funcionamiento del algoritmo.

El algoritmo de cifrado RSA [11] fue publicado en 1977, recibe este nombre al tomar la primera letra de los apellidos de los creadores Ron Rivest, Adi Shamir y Len Adelman. Posteriormente fue patentado por el MIT en 1983 y después del vencimiento de esta patente, se convirtió en un método de cifrado popular. La fortaleza de este algoritmo se basa en la dificultad de factorizar un número, resultante de la multiplicación de dos números primos, actualmente se recomienda el uso de números primos de al menos $2048\,\text{bits}$. A continuación se describe el algoritmo.

Primero se deben elegir dos números primos p y q, es decir que cada número solo tenga como divisor al 1 y así mismo. Después se calcula el número n, producto de p y q.

$$n = p \cdot q.$$

Posteriormente, se calcula la función multiplicativa de Euler $\phi(n)$. Para un número X dado, esta función define el número de enteros positivos menores o iguales que X que son primos relativos de X. La función se calcula como: $\phi(X) = X - 1$. Descomponiendo n en p y q dentro de la función $\phi(n)$ se tiene:

$$\phi(n) = (p-1)(q-1).$$

Posteriormente, se elige un número e que sea mayor que 1 pero menor que $\phi(n)$, es decir, que sea primo relativo de $\phi(n)$. Es necesario que e y $\phi(n)$ tengan como máximo común divisor a 1, de esta forma se asegura que exista el inverso multiplicativo de e. Una vez asegurada esta condición, se debe calcular d, que es el inverso multiplicativo de e módulo n. Dado la condición de que e y $\phi(n)$ tienen como máximo común divisor a 1, se puede expresar a su relación en una combinación lineal:

$$e \cdot u + \phi(n) \cdot v = 1.$$

Se buscan los números u y v que cumplan esta ecuación, para finalmente calcular d

$$d = (u) mod \phi(n).$$

La tupla d y n conforman la llave privada, mientras que e y n conforma la llave pública. Suponiendo un transmisor (Alice) que quiere mandarle un mensaje M al receptor Bob, Alice obtiene el mensaje cifrado C elevando M al exponente e de Bob en modulo n, siendo esta la llave pública de Bob:

$$C = M^e (mod n). \tag{3.1}$$

Cuando Bob recibe el mensaje cifrado, para descifrar el mensaje debe realizar la operación

$$M = C^d mod n. \tag{3.2}$$

Con su llave privada puede descifrar el mensaje dado que e es el inverso multiplicativo módulo n de d, es decir

$$(M^e mod n)^d mod n = M.$$

La fortaleza de este algoritmo radica en que si alguien que no es Bob recibe el mensaje, necesita conocer la llave privada de Bob (d_{Bob}). Para ello, el atacante debe calcular el inverso multiplicativo de e. Sin embargo, no se publica la función $\phi(n)$ como parte de la llave pública. Por lo que el atacante debe descomponer los factores primos de la función $\phi(n)$, que conlleva un número exponencial de posibilidades que aumentan conforme el tamaño de la llave aumenta.

Para ejemplificar el algoritmo de RSA, se eligen dos números primos $p = 17$ y $q = 23$, multiplicando estos números se obtiene el número $n = 17 \cdot 23 = 391$. Con estos valores se calcula el valor de la función $\phi(n)$:

$$\phi(n) = (p-1)(q-1),$$

$$\phi(n) = (16)(22) = 352.$$

A continuación se debe elegir un número e tal que sea primo relativo de $\phi(n)$ y que $e < \phi(n)$, además que el máximo común divisor entre ellos sea 1; de los números posibles se selecciona al número 59. La primer condición la cumple ya que $59 < \phi(n)$. Para verificar que cumple la otra condición se utiliza el algoritmo de Euclides [16] con $e = 59$ y $\phi(n) = 352$, para ello se expresa a $\phi(n)$ como una combinación lineal:

$$\phi(n) = e \cdot q + r$$

$$352 = 59 \cdot q + r$$

El número q debe ser un número entero cuya multiplicación con e se aproxime al valor de $\phi(n)$ y esta multiplicación sumada con el número restante r debe ser igual a $\phi(n)$. Para el ejemplo se utiliza $q = 5$ y $r = 57$, con estos valores se tiene la primera ecuación del algoritmo de Euclides:

$$352 = 59 \cdot 5 + 57, \tag{3.3}$$

Para la siguiente ecuación el valor del número por el que se multiplica q, en este caso e, se coloca en lado izquierdo de la ecuación, mientras el valor del número r se utiliza como el número que va a multiplicar a q, como se muestra a continuación:

$$59 = 57 \cdot q + r,$$

al igual que en la ecuación 3.3, se encuentran los valores de q y r que satisfacen la ecuación. Esta vez $q = 1$ y $r = 2$:

$$59 = 57 \cdot 1 + 2, \tag{3.4}$$

se repite el procedimiento utilizado para la ecuación 3.4 hasta que $r = 0$:

$$57 = 2 \cdot q + r,$$

para esta ecuación los valores que cumplen la igualdad son $q = 28$ y $r = 1$:

$$57 = 2 \cdot 28 + 1, \tag{3.5}$$

se establece la siguiente ecuación:

$$2 = 1 \cdot q + r,$$

finalmente, para esta ecuación $q = 2$ y $r = 0$:

$$2 = 1 \cdot 2 + 0. \tag{3.6}$$

Dado que se llega a una ecuación con $r = 0$, termina el algoritmo. El máximo común divisor es el valor de r de la penúltima ecuación, para este caso corresponde a la ecuación 3.5 con $r = 1$. Con esto queda verificado que son primos relativos con máximo común divisor de 1.

Para el cálculo de la llave privada d, se toman las ecuaciones 3.3, 3.4 y 3.5, estas ecuaciones se expresan de forma que todos los números sean variables a excepción de la q y se despeja el valor correspondiente a r.

De la ecuación 3.3 se obtiene:

$$X_{352} = 5X_{59} + X_{57},$$

$$X_{352} - 5X_{59} = X_{57}. \tag{3.7}$$

De la ecuación 3.4:

$$X_{59} = X_{57} + X_2,$$

$$X_{59} - X_{57} = X_2. \tag{3.8}$$

De la ecuación 3.5:

$$X_{57} = 28X_2 + X_1,$$

$$X_{57} - 28X_2 = X_1. \tag{3.9}$$

Se sustituye el valor de X_2 (ecuación 3.8) en la ecuación 3.9:

$$X_{57} - 28(X_{59} - X_{57}) = X_1,$$
$$X_{57} - 28X_{59} + 28X_{57} = X_1,$$

$$29X_{57} - 28X_{59} = X_1. \tag{3.10}$$

Posteriormente, se sustituye el valor de X_{57} (ecuación 3.7) en la ecuación 3.10

$$29(X_{352} - 5X_{59}) - 28X_{59} = X_1,$$
$$29X_{352} - 145X_{59} - 28X_{59} = X_1,$$

$$29X_{352} - 173X_{59} = X_1. \tag{3.11}$$

Recordando que $X_{352} = 352$, $X_{59} = 59$ y $X_1 = 1$, y a su vez estos valores corresponden a $\phi(n) = 352$ y $e = 59$, entonces:

$$29\phi(n) - 173e = 1, \tag{3.12}$$

dado que son operaciones mod $\phi(n)$, implica que $(29\phi(n))_{mod\phi(n)} = 0$, y por tanto, la ecuación 3.12 se puede expresar de la forma:

$$-173e = 1$$

Esto quiere decir que -173 es el inverso multiplicativo de e, por lo tanto $d = -173$. Sin embargo, d no puede ser negativo, ya que no pertenece a los valores de $\phi(n)$, por lo que el inverso multiplicativo se obtiene al sumar 352:

$$d = -173 + 352 = 179.$$

Se puede comprobar que $(d \cdot e) mod n = 1$, lo que implica que d es el inverso multiplicativo de e.

Para ejemplificar el cifrado, se considera $M = 88$ como mensaje, que es letra X en código ASCII, se obtiene:

$$C = M^e (mod n),$$
$$C = 88^{59} mod 391 = 296,$$

para este ejemplo la información cifrada es 296, y el receptor puede obtener la información en claro mediante:

$$M = C^d mod n,$$
$$M = 296^{179} mod 391 = 88.$$

Capítulo 4

Algoritmo basado en fractales

En este capítulo se presentan las características generales de los fractales. Específicamente, se presentan el fractal de Julia y el fractal de Mandelbrot. Posteriormente, se explica el funcionamiento del algoritmo de llave asimétrica basado en el fractal de Mandelbrot y se presenta un ejemplo numérico de cifrado y descifrado.

4.1. Fractales

El matemático Benoit Mandelbrot fue una de las primeras personas en utilizar el término fractal como un objeto con fragmentos de orientación y tamaño variable, cuya estructura básica se repite a diferentes escalas para formar una figura en un plano más grande [17], por ejemplo algunos de los fractales son representados en el plano complejo. La palabra fractal tiene su origen en el latín del vocablo *fractus*, que significa fragmentado [18]. Algunas de sus características principales de un fractal son [19]:

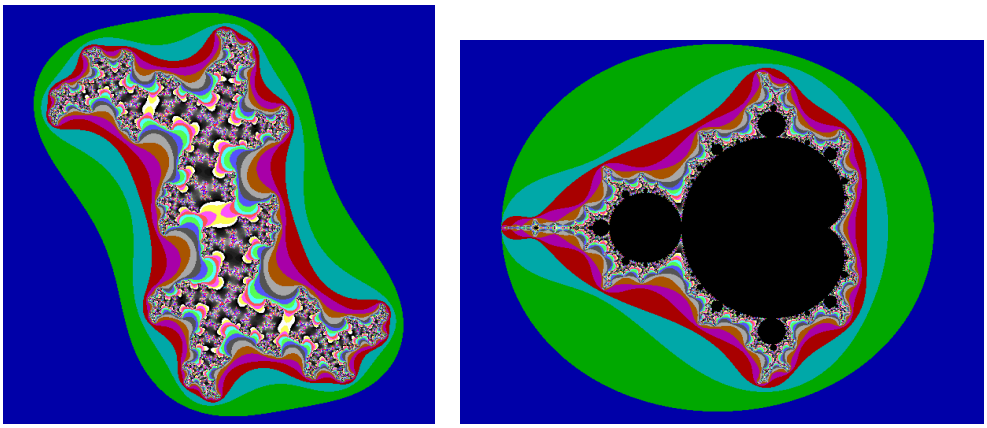
- Es descrito a través de un algoritmos recursivo.
- Es autosemejante.
- La dimensión fractal es mayor a la topológica.

Existen dos tipos de fractales, lineales y los no lineales [20]. Los fractales lineales son aquellos que se construyen a partir de una figura que es la unidad del fractal, esa figura se repite con diferentes escalas y en diferentes direcciones hasta tomar un cierto tamaño. Algunos ejemplos de este tipo de fractales son el triángulo de Sierpinski, la alfombra de Sierpinski y la curva de Von Koch.

Los fractales no lineales son aquellos que se representan en el plano complejo. Dos de los fractales no lineales más conocidos son el fractal de Julia y el de Mandelbrot. En 1918 el matemático francés Gaston Julia definió un conjunto de números complejos que se le conoce como el conjunto de Julia [18], llegó a la definición de este conjunto debido a que estaba investigando sobre las propiedades de la iteración de funciones polinómicas. Su trabajo se enfocaba en encontrar cuencas de atracción de los ceros de un polinomio, cuyo resultado por el método de Newton proporciona una función de iteración. Julia se calcula mediante la ecuación:

$$Z_n = Z_{n-1}^2 + c; c, Z_n \in C; n \in Z. \tag{4.1}$$

Para cada número del plano complejo, crea una sucesión donde c es un número complejo fijo; la frontera de esta función graficada en el plano complejo define la silueta de una figura irregular. Los números que se encuentran dentro de esta figura son aquellos números con divergencia finita, mientras que los que se encuentran afuera tienden al infinito. Si se grafican los puntos que pertenecen al conjunto se obtiene el fractal de Julia (ver la figura 4.1(a)). El conjunto de Julia es un conjunto invariante, acotado, cerrado y no vacío.



(a) Fractal de Julia. (b) Fractal de Mandelbrot.

Figura 4.1: Gráficas de fractales no lineales [21].

El conjunto de Mandelbrot está basado en el conjunto de Julia, en esencia son similares, utilizan la misma función con la diferencia de que c es variable, por tanto el conjunto de Mandelbrot se conforma por los valores de c donde la función no converge a infinito, el fractal que se forma de este conjunto se puede observar en la figura 4.1(b).

Actualmente se han encontrado diferentes aplicaciones a los fractales como la resolución de ecuaciones diferenciales, el tratamiento digital de imágenes y el cifrado.

4.2. Algoritmo de llave pública basado en el fractal de Mandelbrot

La figura 4.2 muestra el proceso de creación de llaves, cifrado y descifrado del fractal de Mandelbrot. Primero, se selecciona un entero x y un complejo c que pertenece al fractal de Mandelbrot, esta información se intercambia entre un transmisor (Alice) y un receptor (Bob). Posteriormente, Alice selecciona un número entero n tal que $x < n$ y un complejo e que pertenece al fractal de Mandelbrot. La tupla (n,e) forma la llave privada de Alice. De igual forma, Bob selecciona un número entero k tal que $x < k$ y un número complejo d que pertenece al fractal de Mandelbrot. La tupla (d,k) forma la llave privada de Bob.

Una vez que Alice generó su llave privada, debe generar su llave pública mediante la ecuación:

$$Z_n = Z_{(n-1)} \cdot c \cdot e, Z_0 = c, \tag{4.2}$$

donde la tupla (n,e) es la llave privada de Alice y c es el número compartido al inicio de la comunicación. En otras palabras, Alice genera su llave pública ($Z_n e$) al iterar n veces la ecuación 4.2, e inicializar $Z_0 = c$. De igual forma, Bob genera su llave pública mediante la siguiente ecuación.

$$Z_k = Z_{(k-1)} \cdot c \cdot d, Z_0 = c, \tag{4.3}$$

donde la tupla (k,d) es la llave privada de Bob y c es el número compartido al inicio de la comunicación. De la misma manera que Alice, Bob genera su llave pública ($Z_k d$) al iterar k veces la ecuación 4.3.

A diferencia de RSA, donde cifrar un mensaje solo requiere la llave pública del receptor, en Mandelbrot cada parte de la comunicación debe crear una llave de cifrado cuyas entradas son la llave pública de la parte receptora y la llave privada del transmisor. Esta llave se crea mediante el fractal de Julia. Este proceso se describe en dos partes. Primero, Alice usa la llave pública de Bob ($Z_k d$) y su llave privada (n,e) para iterar la función de Julia:

$$Z_n = Z_{(n-1)} \cdot c \cdot e, Z_0 = Z_k d, \tag{4.4}$$

donde c es el número compartido al inicio de la comunicación. Es importante notar que en esta ocasión la función de Julia se inicializa con $Z_0 = Z_k d$ (la llave pública de Bob). A este proceso lo denotaremos como $(Z_k d)_n e$. Posteriormente, la llave de cifrado de Alice se genera de la siguiente forma:

$$U = c^{n-x} \cdot (Z_k d)_n e, \tag{4.5}$$

donde c y x son los valores compartidos al inicio de la comunicación. De la misma forma, Bob itera la siguiente ecuación:

$$Z_k = Z_{(k-1)} \cdot c \cdot d, Z_0 = Z_n e, \tag{4.6}$$

donde c es el valor compartido y se inicializa la función de Julia con la llave pública de Alice ($Z_0 = Z_n e$). A este proceso lo denotaremos como $(Z_n e)_k d$. Posteriormente, Bob obtiene su llave de cifrado mediante:

$$W = c^{k-x} \cdot (Z_n e)_k d. \tag{4.7}$$

Dado la relación entre el fractal de Mandelbrot y el fractal de Julia, resulta que la llave de cifrado de Alice (U) y la llave de cifrado de Bob (W) son iguales. Esto quiere decir que tanto Alice como Bob de manera independiente obtienen una misma llave secreta. Por lo que cifrar un mensaje se reduce a una suma y descifrar un mensaje se reduce a una resta. Supongamos que Alice quiere enviar un mensaje M a Bob. El mensaje cifrado se calcula mediante:

$$V = U + M. \tag{4.8}$$

Cuando Bob recibe el mensaje cifrado, solo tiene que restar:

$$M = V - W. \tag{4.9}$$

Para ejemplificar este algoritmo, se elige el número entero $x = 3$ y el complejo $c = 0{,}7126 + 0{,}9991j$. Alice elige el número entero $n = 4$ y el complejo $e = 0{,}8082 + 0{,}9996j$ del conjunto de Mandelbrot. Por su parte Bob elige el entero $k = 6$ y el complejo $d = 0{,}6172 + 0{,}9985j$ del conjunto de Mandelbrot. Se observa que con los números elegidos se cumplen las condiciones $x < n$ y $x < k$.

Mediante la ecuación 4.2, Alice genera su llave pública al iterar la función de Mandelbrot cuatro veces, el resultado de este proceso es:

$$Z_n e = 15{,}5817155634 - 7{,}36807821097j.$$

De la misma forma Bob genera su llave pública al iterar la ecuación 4.3 seis veces. El resultado de este proceso es:

$$Z_k d = 34{,}7790313912 - 13{,}9125763419j.$$

Ahora Alice debe tomar la llave pública de Bob y calcular $(Z_k d)_n e$ al iterar cuatro veces la ecuación 4.4, donde $Z_0 = Z_k d$ (la llave pública de Bob). El resultado de este proceso es:

$$(Z_k d)_n e = 176,704431656 + 150,278755171j.$$

Posteriormente, la llave de cifrado de Alice (U) se obtiene al multiplicar $c^{4-3} \cdot (Z_k d)_n e$ (ver ecuación 4.5). La llave de cifrado de Alice es:

$$U = -24{,}223926 + 283{,}634039j.$$

De la misma forma Bob calcula el valor de $(Z_n e)_k d$, al iterar la ecuación 4.6 seis veces donde $Z_0 = Z_n e$ (la llave pública de Alice). El resultado de este proceso es:

$$(Z_n e)_k d = 176{,}704431656 + 150{,}278755171j.$$

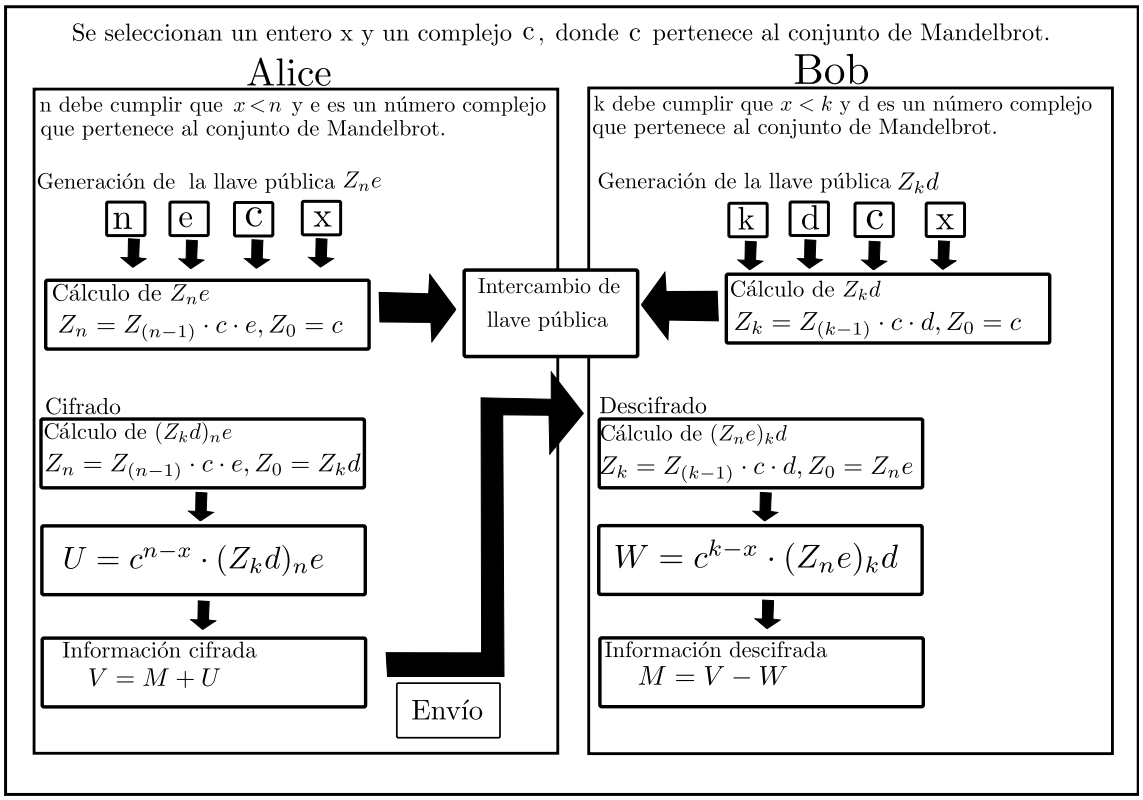


Figura 4.2: Diagrama de cifrado utilizando el fractal de Mandelbrot [13].

Posteriormente, la llave de cifrado de Bob (W) se obtiene al multiplicar $c^{6-3} \cdot (Z_ne)_kd$ (ver ecuación 4.7). La llave de cifrado de Bob es:

$$W = -24{,}223926 + 283{,}634039j.$$

Se puede observar que U y W son iguales. Suponiendo un mensaje $M = 65 + 0j$, el mensaje cifrado es:

$$V = M + U,$$

$$V = 40{,}776074 + 283{,}634039j.$$

Para descifrar el mensaje, Bob solo debe restar:

$$M = V - W,$$

$$M = 40{,}776074 + 283{,}634039j - (-24{,}223926 + 283{,}634039j),$$

$$M = 65 + 0j.$$

Capítulo 5

Implementación y resultados

En este capítulo se explica el entorno de MicroPython y las características básicas del ESP8266. Posteriormente, se explica el desarrollo de los experimentos y se hace un análisis de los resultados obtenidos.

5.1. MicroPyhton

MicroPython es una implementación ligera y optimizada de Python 3 que incluye algunas de las bibliotecas estándar de Python. Fue diseñado para ser utilizado en microcontroladores o en sistemas embebidos, con la intención de que sea fácil transferir un código.

MicroPython no tiene los alcances de Python, por el contrario, es un lenguaje más limitado, debido a que la memoria de los microcontroladores usualmente son de poca capacidad y no son capaces de soportar todas las bibliotecas [22, 23].

Otra cualidad importante de MicroPython es que es *open-source*, un proyecto abierto donde la comunidad puede reportar problemas, proponer soluciones y mejoras al código. Algunos de los dispositivos para los cuales MicroPython tiene soporte son ESP8266, ESP32, WiPy, CC3200, Pyboard, Teenesy, Black STM32F407VET6 y HydraBus. Específicamente, para la placa ESP8266, MicroPython tiene la biblioteca *esp*, que contiene funciones específicas para el manejo de este dispositivo [24].

5.1.1. Consideraciones de Micropython

Como se mencionó anteriormente, al ser un lenguaje optimizado para microcontroladores, Micropython no tiene los alcances que tiene Python y al momento de usarlo se deben tomar en cuenta las limitaciones de este lenguaje. La primera consideración es que Micropython no tiene soporte para manejo de números complejos, por lo que se tuvo que implementar una clase donde el objeto complejo está formado por dos atributos, uno representa la parte real y otro la parte imaginaria. También se crearon las funciones para la aritmética de números complejos.

Otro punto a considerar es la aritmética de punto flotante que es necesaria para calcular las operaciones entre complejos que tienen parte entera y parte decimal. El problema al realizar operaciones con decimales en lenguajes con soporte de punto flotante es que son inexactos al representar la parte decimal de los números. Python y MicroPython, representan a los decimales como una suma de fracciones en base dos. Es decir, si se busca representar el número $0{,}125$, el lenguaje lo representa como la suma de $\frac{1}{10} + \frac{2}{100} + \frac{5}{1000}$. El problema se presenta cuando se utiliza un número con el que no es posible tener una representación exacta con sumas de fracciones en base dos, en este caso se calcula una aproximación. Para contrarrestar este inconveniente existen varios métodos diseñados para la aritmética de punto flotante, pero para esta tesis se utilizó uno específicamente. Este método consiste en redondear todos los números de punto flotante con la función *round* propia del lenguaje Python y MicroPython. Esta función toma los valores de punto flotante y se le indica hasta que decimal redondear.

Para esta tesis se utilizó 5 decimales para cada punto flotante. Sin embargo, esta opción tiene limitantes y puede generar errores cuando se usan llaves muy grandes.

5.2. ESP8266

La placa ESP8266 es una placa diseñada entre otros usos, como un dispositivo para proyectos IoT, cuenta con las siguientes características [25]:

- Tamaño reducido.
- Conexión con Wifi a través de interfaces SPI/SDIO o UART.
- Amplificador de potencia.
- Amplificador de recepción de bajo ruido.
- 16 puertos GPIOS para conexión con sensores externos.
- Rango de frecuencias de $2.4\,\mathrm{GHz}$ a $2.5\,\mathrm{GHz}$.
- Voltaje promedio entre $3.3\,\mathrm{V}$ a $3.6\,\mathrm{V}$
- Frecuencia del CPU de 80MHz
- RAM total: 96 KB, disponible después de instalar Micropython: 32KB

Por defecto no tiene instalado MicroPython, sin embargo se puede encontrar un tutorial en su página oficial [24]. Este dispositivo ha sido utilizado en diferentes proyectos IoT como control para puertas inteligentes [26], control de sistemas de energía híbrida [27], sistema de monitoreo de temperatura en tiempo real conectado a la nube [28], sistema de riego inteligente [29], prototipos de detector de fuego [30], entre otros. Además de que ha sido utilizado para analizar la seguridad en redes IoT utilizando algoritmos como Elliptic Curve Diffie-Hellman (ECDH) en intercambio de llaves de forma segura [31].

5.3. Experimentos

Como se mencionó anteriormente en los algoritmos de llave asimétrica, la longitud de las llaves es de suma importancia, ya que entre más grande es la llave, más seguridad proporciona. Esto se debe a que el número de combinaciones posibles crece exponencialmente. Por lo que un ataque por fuerza bruta será más difícil.

En el algoritmo de Mandelbrot, el número de llaves posibles es igual a 2^n, donde n es el número de bits de la llave. Por el contrario, en RSA la llave pública debe cumplir dos requisitos, que el máximo común divisor con $\phi(n)$ sea 1 y e debe ser un número primo. Mediante la ecuación 5.1 [32] se puede calcular la cardinalidad de números primos en el intervalo de 1 a n:

$$Primos = \frac{n}{ln(n) - 1}. \tag{5.1}$$

La tabla 5.1 muestra el número de llaves posibles con respecto al tamaño de la misma. Por ejemplo, para una llave de $32\,\mathrm{bits}$, el espacio de búsqueda en Mandelbrot es de 4294967296 llaves posibles, mientras que en RSA son 202777307. La última columna de la tabla muestra la proporción entre el espacio de búsqueda de Mandelbrot con respecto a RSA. Por ejemplo, para una llave de $32\,\mathrm{bits}$, la proporción es $21{,}18$, es decir, que el espacio de búsqueda de Mandelbrot es $21{,}18$ veces más grande que RSA, lo que implica que un ataque en Mandelbrot mediante fuerza bruta será 21 veces más lento.

La metodología para contabilizar la eficiencia energética en el dispositivo IoT ESP8266 se describe a continuación.

Se implementó el algoritmo RSA y el algoritmo de cifrado basado en el fractal de Mandelbrot en el dispositivo IoT ESP8266 con el fin de medir el tiempo que requiere el dispositivo en cada parte del proceso. Para ello, las pruebas se dividieron en tres:

Tamaño de la llave (bits)	Espacio de búsqueda fractal [Mandelbrot]	Espacio de búsqueda [RSA]	Proporción (Mandelbrot/RSA)
8	256	56	4.57
16	65536	6495	10.09
32	4294967296	202777307	21.18
64	18446744073709551616	4.2541836182174e+17	43.36
128	3.40282366920938e+38	3.8790624011581e+36	87.72
192	6.27710173538668e+57	4.7523465693120e+55	132.084
256	1.15792089237316e+77	6.5624780614204e+74	176.44
512	1.3407807929942e+154	3.788679120215e+151	353.89

Tabla 5.1: Tabla de comparación entre las llaves de RSA y Mandelbrot.

- Generación de llaves.
- Cifrado.
- Descifrado.

Para ambos algoritmos se utilizó como mensaje la letra A en ASCII, el número 65 para RSA y el número $65 + 0j$ en el fractal de Mandelbrot. El tamaño de la llave se incrementa de bit en bit. Para cada experimento, se mide el tiempo en relación con el tamaño de la llave. Para el caso de RSA se aumenta la longitud de la llave e y se eligen dos números primos p y q, de tal forma que $\phi(n)$ vaya en proporción al tamaño de la llave. Para el caso de Mandelbrot se aumenta la llave n y se dejan fijos los números $x = 2$, $k = 6$, $e = 0{,}808192577 + 0{,}9996157444j$, $d = 0{,}617154692 + 0{,}998465031j$ y $c = 0{,}712577442 + 0{,}999136002j$.

Con el propósito de obtener datos estadísticos, cada cambio en la longitud de la llave se repitió 10 veces. El tamaño de las llaves se inicializó en $7\,\text{bits}$, ya que el espacio de búsqueda en RSA para una llave de $7\,\text{bits}$ es de 33 posibles llaves y se eligieron 10 de ellas aleatoriamente para comenzar las pruebas.

5.4. Resultados

La figura 5.1(a) muestra el tiempo de procesamiento de generación de llaves (públicas y privadas) para ambos algoritmos. Para el caso de RSA el valor de n se calcula mediante la multiplicación de los números p y q (elegidos previamente), posteriormente se elige un número e primo cuyo máximo común divisor con $\phi(n)$ es 1. Finalmente se calcula el inverso multiplicativo de e (d).

Por su parte, en el algoritmo de Mandelbrot se calcula la llave pública variado la longitud de n. En la figura 5.1(a) se puede observar como RSA es un poco más rápido que Mandelbrot. La diferencia de tiempo se mantiene relativamente constante en ambos algoritmos. También, se puede observar que la generación de llaves para el caso de $7\,\text{bits}$ requiere de alrededor de $50\,\text{ms}$ y para una llave de $10\,\text{bits}$ el tiempo requerido es de alrededor de $450\,\text{ms}$, es decir, para ambos algoritmos el aumento de $3\,\text{bits}$ en la llave representó un incremento de aproximadamente 5 veces el tiempo de generación de llaves. El aumento más notable se muestra de 9 a $10\,\text{bits}$ dado que se eleva de $200\,\text{ms}$ aproximadamente hasta $450\,\text{ms}$, mientras que el aumento de 7 a $8\,\text{bits}$ representó cerca de $50\,\text{ms}$.

Es importante notar, que los resultados no muestran llaves de más de $10\,\text{bits}$ de longitud. Esto se debe a que al aumentar la llave a más de 10 bits, el dispositivo en el algoritmo de RSA mostraba un error de desbordamiento de memoria de $16\,\text{KB}$. Se sabe que el interprete de Micropython requiere de al menos $8\,\text{KB}$ de RAM para la ejecución de un script simple. Por lo que después del error, se midió la cantidad de memoria disponible mediante la biblioteca *gc* y el dispositivo contaba con $2\,\text{KB}$. Cabe resaltar, que el algoritmo de Mandelbrot no presentó ningún problema de desbordamiento inclusive con llaves de longitud superior a los $512\,\text{bits}$. Sin embargo, se decidió no presentar más datos del fractal, ya que es una comparativa entre dos algoritmos de llave asimétrica.

La figura 5.1(b) muestra el tiempo de cifrado. RSA toma la llave pública de Bob y cifra el mensaje con la ecuación 3.1. Por su parte Mandelbrot debe calcular su llave de cifrado con la ecuación 4.5. Se puede ver en la figura que RSA tiene pequeños incrementos en el tiempo a comparación del tiempo que necesita el fractal de Mandelbrot. Por ejemplo, el tiempo de cifrado de RSA no rebasa los $100\,\mathrm{ms}$, por el contrario, el tiempo que requiere Mandelbrot es de alrededor de $100\,\mathrm{ms}$ para $7\,\mathrm{bits}$ y termina por encima de los $700\,\mathrm{ms}$, un incremento de 7 veces su tiempo inicial. Siendo su incremento más significativo de 9 a $10\,\mathrm{bits}$ y el menos significativo de 7 a $8\,\mathrm{bits}$.

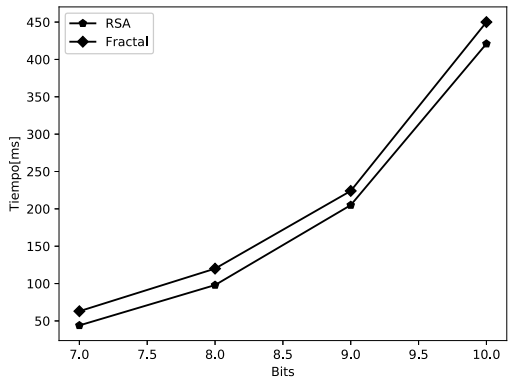
Finalmente, el tiempo de descifrado se muestra en la figura 5.1(c). RSA toma la llave privada junto con el mensaje cifrado y descifra con la ecuación 3.2. Por otro lado, el fractal de Mandelbrot descifra con la ecuación 4.9. Se puede ver en esta figura que el tiempo de descifrado para RSA es considerablemente más grande que en Mandelbrot. Esto se debe a que Mandelbrot solo debe hacer una resta, mientras que RSA debe elevar el mensaje a un exponente del tamaño de su llave y después hacer una operación módulo. Mandelbrot mantiene su tiempo por debajo de los $100\,\mathrm{ms}$, caso contrario con el descifrado de RSA cuyo tiempo empieza arriba de los $200\,\mathrm{ms}$ y llega hasta los $1400\,\mathrm{ms}$. Similar a las tendencias observadas en las fases anteriores, RSA tiene su menor incremento de 7 a $8\,\mathrm{bits}$ con aproximadamente $200\,\mathrm{ms}$, mientras que hay una diferencia muy grande de los 9 a $10\,\mathrm{bits}$ de al rededor de $600\,\mathrm{ms}$.

La figura 5.1(d) muestra el tiempo total de las tres etapas. Se puede observar que RSA es ligeramente más lento en comparación con Mandelbrot, además que la diferencia de tiempo incrementa conforme aumenta el tamaño en bits de la llave. Por lo que podemos esperar que esta diferencia sea cada vez más grande conforme aumenta la longitud de la llave. Para el caso de $7\,\mathrm{bits}$ la diferencia de tiempos entre los algoritmos es de alrededor de $300\,\mathrm{ms}$, sin embargo en la última medición se muestra que RSA necesita $500\,\mathrm{ms}$ más para realizar las etapas de generación de llaves, cifrado y descifrado.

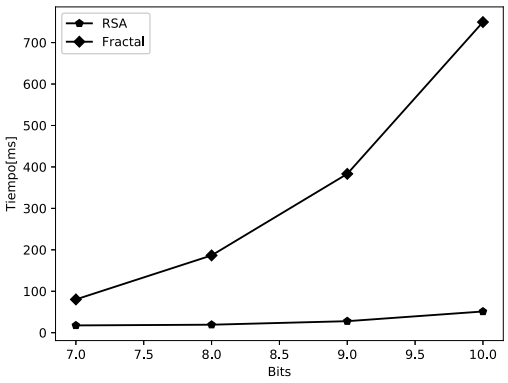
Como se mencionó anteriormente, la generación de llaves en RSA satura la memoria del dispositivo ESP8266, sin embargo este procedimiento se realiza solo una vez en el proceso de comunicación. Es decir, no es un proceso que se realice en cada envío o recepción de datos, por lo que las llaves se pueden calcular de manera externa al dispositivo IoT. La figura 5.2 muestra la generación de llaves en una máquina de escritorio con un procesador de $2.1\,\mathrm{GHz}$ y $8\,\mathrm{GB}$ en RAM. Se puede ver en esta figura como al principio el tiempo de generación de llaves en ambos algoritmos es muy parecido, sin embargo conforme aumenta el tamaño de la llave, el tiempo en RSA se incrementa considerablemente en comparación a Mandelbrot. Para esta prueba, las mediciones se realizaron desde los 8 hasta los $30\,\mathrm{bits}$, incrementando la llave de dos bits en dos bits. Se puede ver en la figura que para $30\,\mathrm{bits}$ RSA requiere más de $12\,000\,\mathrm{s}$, mientras que Mandelbrot le toma alrededor de $8000\,\mathrm{s}$, es decir, más de $4000\,\mathrm{s}$ de diferencia. Además cabe resaltar que RSA incrementa más de $10\,000\,\mathrm{s}$ de 28 a $30\,\mathrm{bits}$, mientras que el algoritmo de Mandelbrot aumenta al rededor de $6000\,\mathrm{s}$. En ambos algoritmos se observa un incremento muy grande de su última medición con respecto de la penúltima.

5.5. Relación de tiempo y eficiencia energética

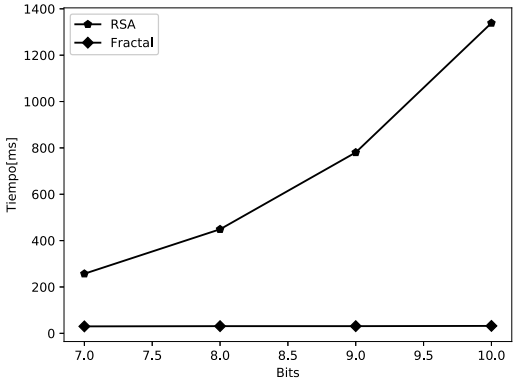
Los resultados de la sección anterior muestran una disminución en el tiempo de procesamiento en el fractal de Mandelbrot en comparación con RSA. Sin embargo, no se cuantificó por cuánto tiempo es mejor un algoritmo que otro. Para ello, se calculó la diferencia de tiempo entre RSA y Mandelbrot para el proceso de generación de llaves, cifrado y descifrado (ver figura 5.1(d). La figura 5.3, muestra la diferencia de tiempos entre RSA y Mandelbrot. Se puede ver en esta figura que para una llave de $7\,\mathrm{bits}$ existe una diferencia de poco más de $100\,\mathrm{ms}$, mientras que para una llave de $10\,\mathrm{bits}$ existe una diferencia de casi $600\,\mathrm{ms}$. Es decir, que el incremento en $3\,\mathrm{bits}$ de la llave representa casi 6 veces el tiempo inicial. Todo esto implica que una mejora en el tiempo, representa una mayor eficiencia en el consumo del dispositivo. Esto a su vez representa una mejor utilización en los recursos del dispositivo IoT.



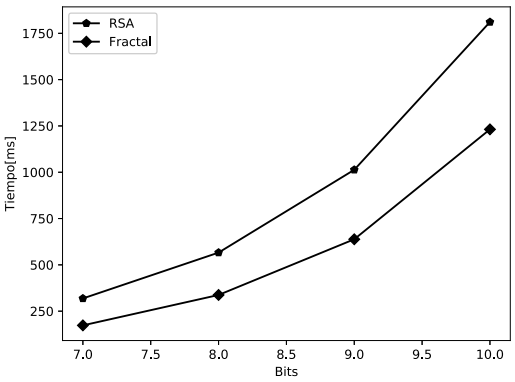
(a) Tiempo de generación de llaves.



(b) Tiempo de cifrado.



(c) Tiempo de descifrado.



(d) Gráfica comparativa del tiempo total.

Figura 5.1: Comparación entre RSA y algoritmo basado en el fractal de Mandelbrot.

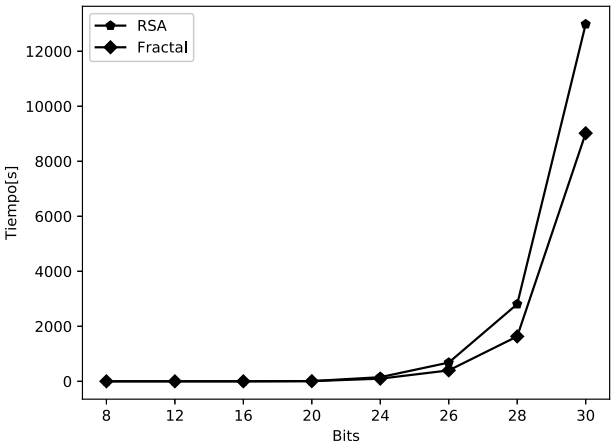


Figura 5.2: Tiempo de generación de llaves externamente.

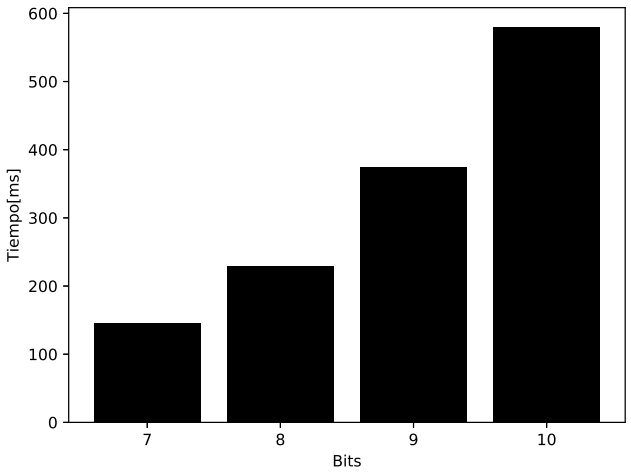


Figura 5.3: Diferencia de tiempo entre RSA y Mandelbrot.

Capítulo 6

Conclusiones

6.1. Conclusiones generales

Esta tesis presenta una comparación entre dos algoritmos de llave asimétrica (RSA y Mandelbrot) con el fin de elegir cuál de ellos es más factible para dispositivos que tienen recursos considerablemente limitados. Para ello, se implementó ambos algoritmos en el dispositivo ESP8266 utilizando el lenguaje de alto nivel MicroPython. Las pruebas se dividieron en tres fases: generación de llaves, cifrado y descifrado. Se observó que Mandelbrot requiere de más tiempo en la generación de llaves comparado con RSA para llaves de hasta $24\,\mathrm{bits}$. Sin embargo, para llaves más grandes, las pruebas muestran que el tiempo que requiere RSA es considerablemente más grande que Mandelbrot (ver figura 5.2). En cuanto a la etapa de descifrado, Mandelbrot es mucho más rápido que RSA. Esto se debe a que Mandelbrot solo requiere hacer una resta de números complejos, por el contrario RSA requiere de una potenciación y una operación módulo de números primos muy grandes. Esto se traduce en un mayor tiempo de procesamiento y por ende consumo de batería, debido a que cada segundo de procesamiento representa un decaimiento en la carga de la batería. Específicamente, las pruebas muestran una complejidad algorítmica $O(n^2)$ en RSA y de $O(1)$ en Mandelbrot en cuanto al descifrado (ver figura 5.1(c)).

En cuanto al cifrado, Mandelbrot requiere de mayor tiempo de procesamiento en relación con RSA, ya que genera una llave de cifrado extra que RSA no requiere. Sin embargo, en la mayoría de las comunicaciones, no se requiere cambiar la llave cada vez que se va enviar un mensaje. Por lo que este proceso solo se realiza una vez. Incluso, las llaves se pueden generar externamente al dispositivo IoT y solo usar el dispositivo IoT para cifrado y descifrado de la información. Bajo esta consideración, el algoritmo de Mandelbrot sería más factible de implementar en dispositivos de pocos recursos energéticos, ya que tanto el cifrado como el descifrado solo requieren una suma o una resta de números complejos, respectivamente. Además, cabe mencionar que el cifrado y descifrado basado en el fractal de Mandelbrot no presentó problemas ante el incremento de la longitud de la llave, por lo cual se puede decir que este algoritmo es más apto para ser implementado en dispositivos IoT. Sin embargo, la aritmética de punto flotante en Mandelbrot juega un papel muy importante, ya que el error acumulado en la generación de llaves U y W puede llevar a errores en el descifrado del mensaje.

6.2. Verificación de la hipótesis

Retomando la hipótesis presentada en la sección 1.2:

"El uso de cifrado basado en fractales representa una disminución de consumo energético en comparación con el cifrado basado en el algoritmo de RSA."

Para verificar la hipótesis previa, en esta tesis se implementó el algoritmo RSA y el algoritmo basado en el fractal de Mandelbrot con el objetivo de medir el tiempo que cada uno de ellos utiliza para generar sus llaves, cifrar y descifrar. Los resultados muestran que, tomando

en cuenta el tiempo total (cifrado, descifrado y generación de llaves), se requiere de un menor tiempo con el algoritmo de Mandelbrot a comparación de RSA. Esto implica que el uso del fractal de Mandelbrot representa una disminución en tiempo, que se ve reflejado en una disminución del consumo energético (ver figura 5.1(d)). También cabe resaltar que uno de los procesos que más tiempo consume en el algoritmo basado en el fractal de Mandelbrot es la generación de la llave U y W. Por lo que si estas llaves se calculan externamente al dispositivo, el tiempo de cifrado se reduciría enormemente. Esto implicaría que cifrar o descifrar se reduce a sumas o restas de números complejos. Por el contrario, en RSA cifrar o descifrar equivale a elevar un número a un exponente con modulo $\phi(n)$, lo que implica más operaciones y mayor cantidad de memoria, especialmente cuando las llaves van creciendo de longitud. Lo anterior se puede reflejar en un ahorro en recursos como memoria, tiempo y energía. Más aún, las pruebas muestran que la tendencia de descifrado en RSA es $O(n^2)$, mientras que en Mandelbrot es $O(1)$ (ver figura 5.1(c)). De igual forma, si las llaves en el cifrado se generan externamente al dispositivo, la tendencia de Mandelbrot sería $O(1)$, mientras que RSA seguiría siendo cuadrática. Por todas estas razones concluimos que un algoritmo como Mandelbrot es una mejor opción en dispositivos IoT.

6.3. Perspectivas de investigación

Como perspectiva de investigación se considera implementar ambos algoritmos en algún lenguaje de bajo nivel que permita una implementación más óptima, como ensamblador o lenguaje C. Esto hará posible el uso de llaves con mayor longitud, ya que estos lenguajes suelen aprovechar mejor los recursos de memoria.

Por otro lado, se plantea implementar ambos algoritmos en un microcontrolador diferente, utilizando nuevamente MicroPython para comparar los resultados de consumo energético con un hardware diferente.

Bibliografía

[1] N. Smart,"ECRYPT II Yearly Report on Algorithms and Keysizes(2011-2012)". Technical report 7th Framework Programme, European Commission, 2012. https://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf

[2] M. Schwartz, "Internet of Things with ESP8266". Packt Publishing, UK , 2016.

[3] S. Rajsbaum,"Criptografía". Instituto de Matematicas, Universidad Nacional Autónoma de México, 2005.

[4] V. Delgado, R. Palacios, "Introducción a la Criptografía: Tipos de Algoritmos". Escuela Técnica Superior de Ingeniería(ICAI), Universidad Pontificia Comillas, 2006.

[5] National Bureau of Standards, Data Encryption Standard, FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., Jan 1977.

[6] National Bureau of Standards, Data Encryption Standard, FIPS-Pub.46-3. National Bureau of Standards,U.S. Department of Commerce, Washington D.C.,Oct 1999.

[7] R. Rivest: "The RC4 Encryption Algorithm". RSA Data. Security, Inc., March, 1992.

[8] J. Daemen, V. Rijme, "AES Proposal: Rijndael ".Version 2, Sep 1999.

[9] B. Schneier: "Description of a New Variable-LengthKey, 64-Bit Block Cipher (Blowfish)". Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, pp.191-204,December 1993.

[10] P. Jhansi, S. Durga, "Symmetric Encryption Using Sierpinski Fractal Geometry". Communications in Computer and Information Science, vol 157. Springer, Berlin, Heidelberg, 2011.

[11] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM, Vol. 21,pp.120–126, 1978.

[12] National Bureau of Standards, Data Encryption Standard, FIPS-Pub.186-2. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., 2000.

[13] M. Alia, A. Samsudin,"A New Public-Key Cryptosystem Based On Mandelbrot and Julia Fractal Sets". Asian Journal of Information Technology, Malaysia, 2007.

[14] D. Palanivel, "Comparative Study on Data Encryption Algorithms in Cloud Platform". International Journal of Engineering Research & Technology, Vol. 6 , October 2017.

[15] B. Oguntunde, S. Arekete , M. Odim, O. Olakanmi,"A Comparative Study of Some Traditional and Modern Cryptographic Techniques ".International Journal of Engineering and Management Research, Volume 7, Issue 6, December 2017.

[16] J. Guo, C. Wang, "Bit-Serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in GF". Department of Electrical Engineering, National Tsing Hua University Hsinchu, Taiwan, Republic of China, 1997.

[17] B. Mandelbrot, "The Fractal Geometry of Nature". W.H. Freeman, 1977.

[18] M. Sastre,"Geometría Fractal". Universidad Politécnica de Madrid, Facultad de Informática.

[19] G. Peña , "Introducción al Estudio de las Imágenes Fractales". Universidad Nacional del Sur, Argentina, 2012.

[20] P. Valdés , "Introducción a la Geometría Fractal". Universidad del Bío-Bío,Chile, Facultad de Educación y Humanidades, 2016.

[21] N. Giffin ,"Fractint". TRIUMF project at the University of British Columbia Campus in Vancouver B.C. Canada, 2006.

[22] D. George, "Porting of MicroPython to LEON Platforms". TEC-ED & TEC-SW Final Presentation Days, Cambridge, United Kingdom, June 2016.

[23] Adafruit Learning System, "MicroPython Basics: What is MicroPython?". Adafruit Industries ,January 2019.

[24] MicroPython, "MicroPython". https://micropython.org/

[25] Espressif Systems, "Datasheet ESP8266EX". Version 6.3, 2019.

[26] I. Muhammad,"Smart Door Locks Based on Internet of Things Concept with mobile Backend as a Service". Jurnal Electronics, Informatics, and Vocational Education (ELINVO), Volume 1, No. 3, November 2016.

[27] P. Srivastava, M. Bajaj and A. S. Rana,"IOT based controlling of hybrid energy system using ESP826". IEEMA Engineer Infinite Conference (eTechNxT), New Delhi, 2018, pp. 1-5.

[28] S. Saha and A. Majumdar, "Data centre temperature monitoring with ESP8266 based Wireless Sensor Network and cloud based dashboard with real time alert system". Devices for Integrated Circuit (DevIC), Kalyani, 2017, pp. 307-310.

[29] P. Srivastava, M. Bajaj and A. S. Rana,"Overview of ESP8266 Wi-Fi module based Smart Irrigation System using IOT". Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), Chennai, 2018, pp. 1-5.

[30] T. Listyorini, R. Rahim,"A prototype fire detection implemented using the Internet of Thingsand fuzzy logic".World Transactions on Engineering and Technology Education, Vol.16, No.1, 2018.

[31] R. K. Kodali and A. Naikoti, "ECDH based security model for IoT using ESP8266". International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil, 2016, pp. 629-633.

[32] C. Caldwell , "How Many Primes Are There?". The University of Tennesse , 2006. https://primes.utm.edu/howmany.html