Implementación y Análisis de rendimiento de Algoritmos Criptográficos para aplicaciones en Sistemas Embebidos

María Sol Gonzalez[1], Graciana Roldán[2], Federico Gabriel D'Angiolo[3], Fernando Asteasuain[4]

[1] Universidad Nacional de Avellaneda
Mariasolgonzalez94@gmail.com
[2] Universidad Nacional de Avellaneda
groldan@undav.edu.ar
[3] Universidad Nacional de Avellaneda
fdangiolo@undav.edu.ar
[4] Universidad Nacional de Avellaneda
fasteasuain@undav.edu.ar

Resumen. El presente trabajo describe la implementación y análisis de algoritmos criptográficos en Sistemas Embebidos. Los algoritmos bajo estudio son: AES y RSA. El objetivo es comparar y determinar cuál de estos algoritmos resulta ser el más eficiente al momento de aplicarlos en una arquitectura de 32 bits como las implementadas en las placas de desarrollo basadas en microcontroladores ARM. Esta aplicación permite gran adaptación a sistemas industriales.

1 Introducción

Actualmente la transmisión y recepción de datos es vital en muchos sistemas, sobre todo en los industriales, y en consecuencia su seguridad resulta de gran importancia. Una de las principales herramientas para proteger la transmisión de los datos es la *criptografía,* consiste en ocultar la información mediante técnicas de cifrado, resultando solo comprensible para el destinatario, quien podrá obtener la información original.

Como sucede con la mayoría del equipamiento industrial, los sistemas embebidos están pensados para ser seguros a nivel físico. Cuando estos sistemas embebidos son colocados en una red surgen los inconvenientes **[1]**; y a razón de esto el presente trabajo busca implementar criptografía actual: AES y RSA, dentro de un sistema embebido.

Tal como se menciona en el trabajo *"Lightweight Cryptography for Embedded Systems – A comparative Analysis"* **[2]**, en el contexto de los sistemas embebidos el problema radica en la naturaleza de los dispositivos, ya que disponen de

recursos limitados. Por esto, resulta importante determinar cuál algoritmo es el más efectivo en tiempo de procesamiento y gasto computacional, de lo contrario se estarían desperdiciando recursos que el sistema embebido podría utilizar para otra labor específica **[3]**.

Habiendo comentado la justificación de este estudio, a continuación, se describe la distribución de temas en los que se basa este trabajo. En la sección dos, se describen los algoritmos, para tener una base teórica de su funcionamiento. En la sección tres se detalla el desarrollo del trabajo, fundamentado en la sección anterior. Por último, en la sección cuatro se presentan los resultados obtenidos, justificados por el desarrollo.

2 Descripción de algoritmos

2.1 AES

El *Estándar de cifrado avanzado (AES*) o también conocido como *"Rjandael",* es el cifrado simétrico (Fig.1) más utilizado en la actualidad. Por definición, este tipo de algoritmos aplican la misma clave tanto para cifrar como descifrar; y para su intercambio requieren de un canal de comunicación seguro.

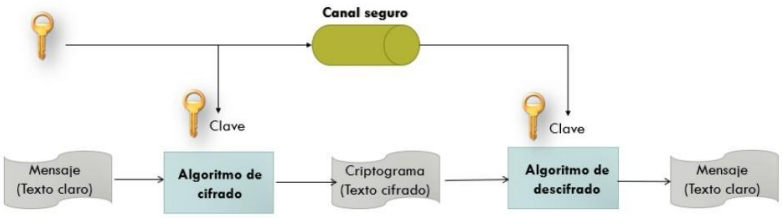

Fig. 1. Esquema general de funcionamiento del algoritmo simétrico AES.

Como se observa en la figura anterior (Fig.1), el algoritmo de cifrado recibe como entradas: un mensaje o texto claro y una clave, y genera como salida un criptograma o texto cifrado. Internamente este algoritmo (Fig.2) divide el texto claro en bloques ***X***, cada uno de los cuales se cifra aplicando la clave ***K*** para generar cada bloque cifrado ***Y***, que luego se concatenan para formar el texto cifrado final. AES propone una longitud fija de bloque ***X*** de 128 bits y longitud variable para la clave ***K*** de 128, 192 o 256 bits.

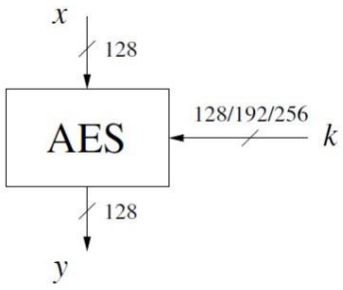

Fig. 2. Parámetros de entrada/salida del algoritmo de cifrado AES.

El algoritmo de cifrado AES utiliza redes de sustitución-permutación. El objetivo principal de estas redes es aplicar ***N*** rondas de sustituciones y permutaciones a cada bloque, con la intervención de una subclave diferente en cada ronda. Estas subclaves se generan a partir de la clave ***K***. En la siguiente tabla (Tabla 1) se pueden visualizar los diferentes parámetros de AES **[4]** según la longitud de la clave:

Tamaño de la clave K (key size) (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Tamaño de bloque de texto claro X (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Número de rondas N	10	12	14
Tamaño de la subclave por ronda (words/bytes/bits)	4/16/128	4/16/128	4/16/128

Tabla 1. Parámetros del algoritmo de cifrado AES según el tamaño de clave

En cada ronda se aplica un conjunto de operaciones tanto al bloque ***X*** como a la clave ***K***. Cada bloque ***X*** de 128 bits o 16 bytes, se estructura como una matriz cuadrada de 4 x 4 bytes, conocida como matriz de estado inicial. Luego de cada ronda se consigue un estado intermedio, hasta llegar en la última iteración al bloque cifrado ***Y***.

Las operaciones indicadas anteriormente se organizan en tres capas*(layers)* **[4]**:

1. **Byte Substitution layer (S-Box) / Forward substitute byte transformation / SubBytes:** Cada elemento de la matriz de estado es sustituido por un valor constante de una matriz fija de 16 x 16 bytes. Para la búsqueda se fracciona cada byte del estado en dos dígitos hexadecimales: el primero representa la fila y el segundo la columna, y con esos índices se selecciona el elemento de permutación.

2. **Diffusion layer**: Proporciona difusión sobre todos los bytes de la matriz de estado. Incluye dos operaciones:

a. **ShiftRows:** Consiste en la rotación cíclica de cada una de las filas de la matriz de estado. La primera fila no se altera, la segunda fila realiza un desplazamiento a la izquierda, la tercera fila se desplaza dos veces a la izquierda, y por último la cuarta fila realiza un desplazamiento a la izquierda de tres posiciones.

b. **MixColumn:** Dentro de esta operación los cuatro bytes de cada columna son multiplicados dentro del campo de Galois por una matriz determinada.

3. **Key Addition layer / AddRoundKey:** Se combina la matriz de estado con la subclave correspondiente utilizando una XOR a nivel de bits.

Para cada ronda, la subclave se deriva de la clave principal mediante la aplicaciónde una operación de expansión. Inicialmente la clave *K* de 128 bits o 16 bytes se organiza como una matriz de 4 x 4 bytes, y en el proceso de generación de las subclaves "se expande" con nuevas columnas hacia la derecha:

- Al generar una columna cuya posición es múltiplo de cuatro, se parte de la columna anterior y aplican tres operaciones: una rotación de arriba hacia abajo, una permutación SubBytes y finalmente una XOR en base a una matriz constante propia de las subclaves.
- Al definir el resto de las columnas, que no son múltiplo de cuatro, sólo se aplica la operación XOR entre la columna anterior y la que se encuentra cuatro posiciones previas.

Para cifrar cada bloque X, se aplican estas tres capas en N rondas. Obtenidos todos los bloques Y, se unen para dar lugar al criptograma final.

2.1 RSA

El algoritmo RSA *(Rivest, Shamir y Adleman)* es un algoritmo de cifrado asimétrico (Fig. 3), es decir, utiliza una clave de cifrado y otra clave de descifrado. Se dice que la clave de cifrado (Fig.3 - K_U**)** es la "clave pública" porque justamente se comparte, con la particularidad que su intercambio no requiere necesariamente de un canal seguro. Por otro lado, la clave de descifrado debe permanecer secreta y se conoce como "clave privada" (Fig.3 - K_R**)** .

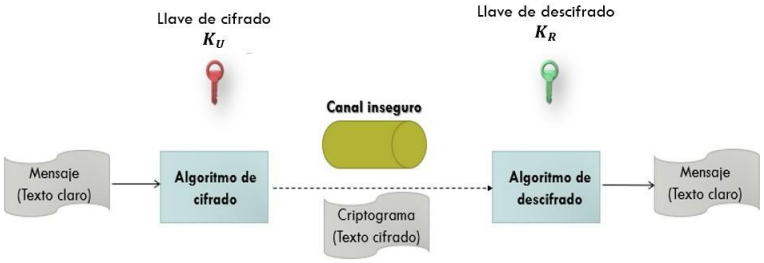

Fig. 3. Esquema general de cifrado asimétrico

La complejidad de estos algoritmos está relacionada con la resolución de un problema matemático de difícil solución en un tiempo razonable, lo cual permite que sea poco probable de quebrantar. En este sentido, RSA basa su funcionamiento en operaciones matemáticas con número primos grandes: ***p*** y ***q***. A partir de estos valores, se generan la clave pública *KU* y la clave privada *KR,* siguiendo los pasos detallados a continuación:

Generación Clave	
Seleccionar *p, q*	P y q primos, $p \neq q$
Calcular *n*	$n = p * q$
Calcular $\phi(n)$	$\phi(n) = (p-1)(q-1)$
Seleccionar entero *e*	$mcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calcular *d*	$d \ mod \ \phi(n) = 1$
Clave publica	$KU = \{e,n\}$
Clave privada	$KR = \{d,n\}$

Tabla 2. Proceso de generación de claves en RSA

Sea el texto claro ***M*** (expresado como entero) y la clave pública *KU = {e,n}*, el texto cifrado **C** (también de tipo entero) se consigue con la operación:

Cifrado	
Texto claro:	$M<n$
Texto cifrado:	$C = M^e mod(n)$

Tabla 3. Proceso de cifrado RSA

Sea el texto cifrado **C** (representado como entero) y *KR = {d,n}* la clave privada, el proceso de descrifrado se aplica con la siguiente operación:

Descifrado	
Texto cifrado:	C
Texto claro:	$M = C^d mod(n)$

Tabla 4. Proceso de descifrado RSA

Con RSA el cifrado es por bloques y su tamaño está limitado por el valor de *n* (Tabla 2); de este modo el texto claro ***M*** y el texto cifrado ***C*** son enteros comprendidos en *0* y *n-1* **[6]**.

3. Desarrollo

3.1 Sistema embebido implementado

El sistema embebido funciona a partir de un sensor de temperatura, DS18B20, que toma las correspondientes muestras en °C, el cual fue seleccionado debido a su amplio rango de medición que varía entre -55°C a +125°C, y por emplear un bus de comunicación 1Wire . Estos datos los envía a Raspberry Pi 3 Model B, que se encarga de aplicar el cifrado. El motivo de utilizar una Raspberry Pi, es porque en ella se puede embeber un sistema operativo, lo cual, facilita el desarrollo de los análisis criptográficos en ella.

La comunicación entre el sensor y la Raspberry Pi es a través del protocolo 1Wired, ya que la ventaja de este protocolo radica en que se utiliza un único conductor para realiza la comunicación, posibilitando un diseño sencillo de la placa.

Una vez alojados los datos de temperatura en la Raspberry Pi Model B, se ejecuta un script que encripta los datos en AES o RSA. Luego, estos datos son enviados a través del protocolo HTTP a un host remoto, el cual, posee un script para desencriptar los datos de temperatura (Fig. 4)

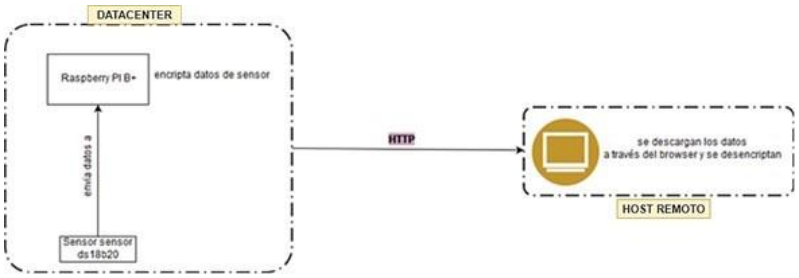

Fig. 4. Esquema general de cifrado asimétrico.

3.2 Implementación de algoritmos en Python

Ambos algoritmos se desarrollaron en Python, bajo la forma de una biblioteca criptográfica.

3.2.1 Implementación de AES

Se implementó el algoritmo AES con longitud de clave 128 bits. En la Fig. 5 se observa que en primer lugar se ingresa el texto claro y luego se realiza una ronda inicial, en donde se aplica el proceso de *AddRoundKey()* y a partir de la password (clave) se inicia el proceso de generación de subclaves (*Create_SubKeys ()*).

Dentro de la función *Main_AES ()*, se encuentran todas las llamadas a las demás subfunciones, y se itera con las rondas hasta *Wr=10*. Se realizan nueve (9) rondas principales, en las que se aplican las cuatro operaciones básicas del cifrado AES: *SubBytes(), ShiftRows(), MixColumns()* y, *AddRoundKey()*. Todas ellas están incluidas dentro de la función *AES_Cycle()*. En la última ronda *(Wr = 10),* se aplican las operaciones de *SubBytes(), ShiftRows(),* y *AddRoundKey()*. El texto cifrado se maneja a nivel de ASCII, para realizar las operaciones, las cuales, guardan el resultado en un array.

Una de las operaciones más complejas dentro del algoritmo AES, es la generación de subclaves. La función que se encarga de realizar esta operación se denomina *Create_SubKeys()*.

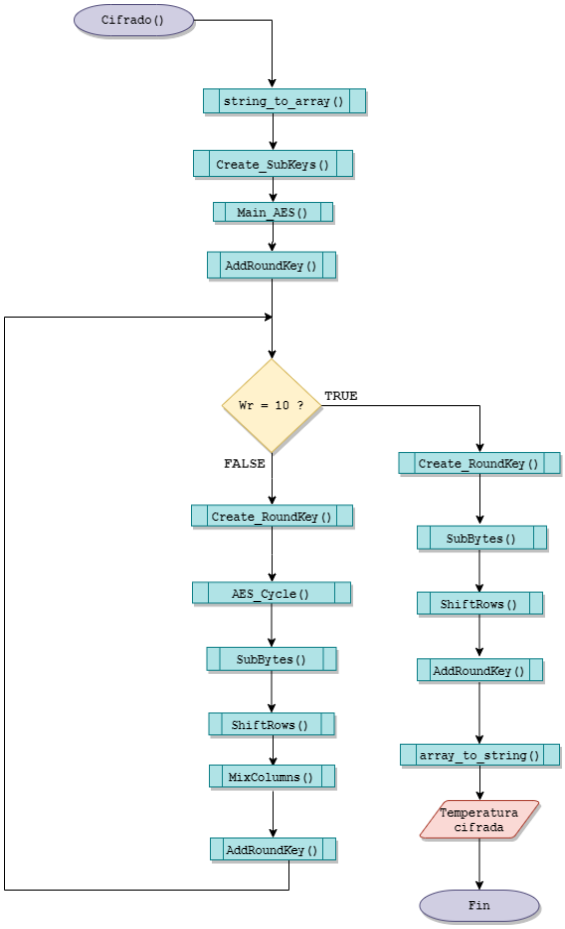

Fig. 5. Diagrama de flujo de AES implementado en Python.

3.1.2 Implementación de RSA

Se desarrolló el algoritmo RSA con longitud de clave 2048 bits (Fig, 6). En principio se ingresa la temperatura a cifrar y los números primos correspondientes vinculados con las claves pública y privada. Los valores ingresados *(p, q,phi,e,d)* pasan por una serie de validaciones. Lo principal es verificar que tanto **p** como **q** sean primos. Una vez que se verifica que los valores sean correctos y cumplan con las propiedades del algoritmo, se realiza la operación de cifrado (*Cifrado_RSA()*). Esta última, llama a la función *define_k(),* que calcula el tamaño en bits de los bloques a cifrar.

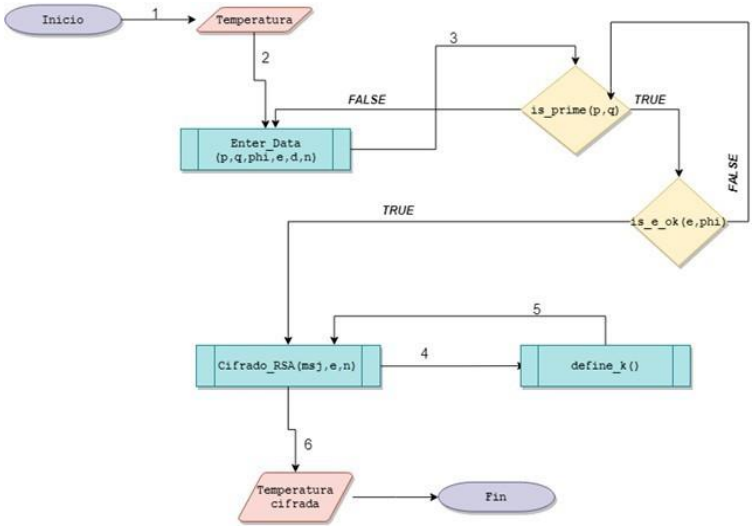

Fig. 6. Diagrama de flujo de RSA implementado en Python.

4. Resultados

Para evaluar el costo computacional de ambos algoritmos en el sistema embebido propuesto, según el paper "*A Hybrid Approach for Intelligent Communication and Performance Analysis over DSRC VANET*" **[8]** se realizó una parametrización sobre el rendimiento de cada uno según la siguiente expresión matemática:

$$RC = \frac{\sum input}{\sum tiempo\ de\ cifrado} \quad (1)$$

$$RD = \frac{\sum input}{\sum tiempo\ de\ descifrado} \quad (2)$$

Siendo:

RC: Rendimiento de cifrado (bytes por segundo)
Input: tamaño del archivo a cifrar (bytes)
Tiempo de cifrado: tiempo que consume el input en cifrar (segundos)
RD: Rendimiento de descifrado (bytes/segundo)
Tiempo de descifrado: tiempo que consume el input en descifrar (segundos)

Cabe destacar que los tiempos asociados al procesamiento de cada algoritmo son genéricos y, por tanto, solo se compartirá aquellas pruebas que tuvieron mejor desempeño y un resultado más estable.

En la Fig. 7, se puede ver el gráfico de rendimiento para cada algoritmo. De aquí se puede ver que algoritmo de cifrado simétrico (AES) posee un mayor rendimiento frente a RSA, ya que este último es más costoso por su complejidad en operaciones matemáticas.

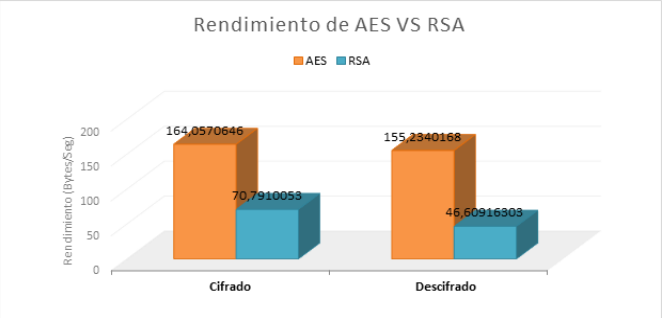

Fig. 7. Rendimiento de AES y RSA para proceso de cifrado y descifrado.

En cuanto a la media en tiempo de procesamiento de los algoritmos RSA y AES en ambos métodos (cifrado, descifrado), se obtuvo el siguiente resultado:

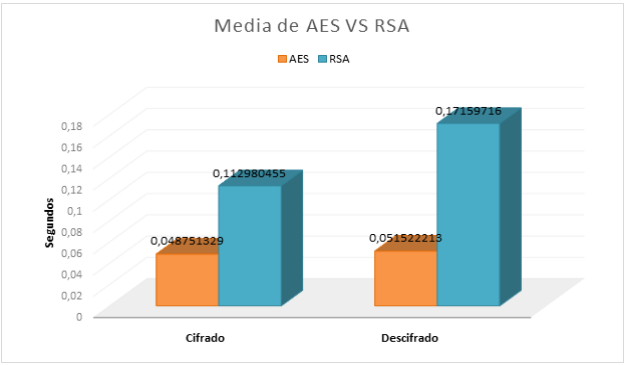

Fig. 8. Media de tiempo de consumo *(seg.)* de AES y RSA para proceso de cifrado y descifrado.

En la Fig.8, se aprecia un aumento mucho mayor en tiempo en cuanto al proceso de descifrado RSA, esto se debe a que el tamaño de bloques *K* para RSA 1024 en la función de descifrado es igual a 257 bytes.

5. Conclusiones

Las conclusiones que se han obtenido a lo largo del desarrollo indican que el algoritmo simétrico AES posee un mejor rendimiento en cuanto al tiempo empleado para las operaciones de cifrado y descifrado. Esto se debe a que la seguridad del

algoritmo RSA se basa en las operaciones matemáticas de exponenciación de números primos, mientras que el algoritmo AES basa su seguridad en la sustitución, permutación y transformaciones lineales ejecutadas en bloque de datos de 16 bytes. El uso de la criptografía asimétrica, en este caso RSA, dará como resultado un alto costo de recursos y tiempos que, si bien se obtuvieron en el rango de los segundos, la suma de cada uno de estos podría significar un tiempo de procesamiento y uso mucho más radical que el uso de el algoritmo asimétrico AES. Sin embargo, se debe tener en cuenta que los algoritmos simétricos y asimétricos no son sencillos de comparar, ya que depende del ámbito donde se va a aplicar. Si bien los algoritmos de clave pública son más recientes no implica que reemplazarán a los algoritmos simétricos. No hay un algoritmo que sea completamente mejor que otro, cada uno ofrece sus ventajas relativas dependiendo de la implementación en la que se aplica. A la hora de optar por un algoritmo simétrico o asimétrico, primero se debe evaluar el tipo de aplicación. Los algoritmos de clave única establecen su predominio a la hora de cifrar grandes cantidades de datos, dado que el volumen de cómputo asociado al empleo de la clave pública podría hacer inviable la aplicación. Por este motivo, cuando se tiene una limitación a nivel hardware, implementar un algoritmo simétrico es más factible que uno de asimétrico, el cual implica un mayor costo al aplicar operaciones matemáticas complejas. En cuanto a la implementación de un algoritmo de cifrado en el sistema embebido de sensado, según los datos obtenidos y las comparaciones, el algoritmo AES se adapta mejor al sistema, ya que el tiempo de consumo es mucho menor al del algoritmo RSA. AES es uno de los más eficientes algoritmos en cuanto a adaptación en hardware y software.

Referencias

[1] Incibe-cert. (2018). https://www.incibe-cert.es. Obtenido de incibe-cert: https://www.incibe-cert.es/blog/introduccion-los-sistemas-embebidos

[2] Manifavas C., H. G. (21 de March de 2014). Lightweight Cryptography for Embedded Systems – A comparative Analysis. (T. E. Dept. of Industrial Informatics, Ed.) DOI: https://doi.org/10.1007/978-3-642-54568-9_21 ISBN:978-3-642-54568-9

[3] Harry Manifavas. Konstantinos Fysarakis. George Hatzivasilis. Konstantinos Rantos: Lightweight Cryptography for Embedded Systems. Conference Paper · September 2013. Issn: 0302-9743

[4] Christof, P. -J. (2010). Understanding Cryptography. New York, EE.UU: Springer. ISBN: 978-3-642-04100-6

[5] Williams, S. (2005). Cryptography and Network Security (Fourth Edition ed.). ISBN-13: 978-0-13-187316-2

[6] Williams, S. (2003). Fundamentos de Seguridad en Redes (2º edición ed.). Madrid: Pearson Education. ISBN: 84-205-4002-1

[7] Tanenbaum, A. S. (2003). Redes de Computadoras (cuarta ed.). Pearson Education. ISBN: 970-26-0162-2.

[8] Gambhir, N., & Sharma, P. (2017). A Hybrid Approach for Intelligent Communication and Performance Analysis over DSRC VANET. IEEE. Paper ID:153.