View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Instituto Tecnológico de Celaya: E-Journals

IMPLEMENTACIÓN DE LA FUNCIÓN SHA3-3 EN UNA ARQUITECTURA ARM

Alfonso Francisco de Abiega Leglisse

Instituto Politécnico Nacional. ESIME Culhuacan

ponchohunt1980@gmail.com

Gina Gallegos García

Instituto Politécnico Nacional, ESIME Culhuacan

ggallegosg@ipn.mx

Resumen

Una función hash es un algoritmo matemático que garantiza la integridad de los datos ya que independientemente de la longitud de los datos de entrada, genera una nueva serie de caracteres con una longitud fija y única para cada una de las entradas que se le proporciona. SHA-3 es una función hash completamente nueva, la cual se diferencia completamente de las versiones anteriores como son SHA-1 y SHA-2. Esta función también es conocida como Keccak y no sólo presenta las salidas clásicas de su antecesor (224, 256, 338 y 512 bits) sino que tiene un paquete de funciones que permite que las salidas sean mucho más grandes para aumentar la seguridad del algoritmo. En este trabajo se presenta la implementación del algoritmo SHA-3 en el lenguaje de programación Python. Los resultados obtenidos a través de las pruebas efectuadas en diferentes plataformas, como lo son un ordenador y una placa embebida Raspberry pi 1 modelo B (dispositivo embebido de bajos recursos computacionales), muestran una diferencia en el tiempo de procesamiento al ejecutar el algoritmo. Con lo cual se pudo comprobar que los dispositivos con recursos limitados pueden estar protegidos ante ataques que violen la integridad de los mismos.

Palabra(s) Clave: Criptografía, Embebidos, Post-cuántica, Sha-3.

Abstract

A hash function is a mathematical algorithm that guarantees the integrity of the data since, regardless of the length of the input data, it generates a new series of characters with a fixed and unique length for each of the inputs that is provided to it. SHA-3 is a completely new hash function, which is completely different from previous versions such as SHA-1 and SHA-2. This function is also known as Keccak and not only presents the classic outputs of its predecessor (224,256,338 and 512 bits) but also has a package of functions that allows the outputs to be much larger to increase the security of the algorithm. The implementation of the SHA-3 algorithm in the Python programming language. The results obtained through tests carried out on different platforms, such as a computer and a Raspberry pi 1 embedded board model B (embedded device with low computational resources), show a difference in the processing time when executing the algorithm. Thus, it could be verified that devices with limited resources can be protected against attacks that violate the integrity of the devices.
Keywords: *Cryptography, Embedded, Post-quantum, Sha-3.*

1. Introducción

Las funciones hash forman parte de varias aplicaciones de seguridad de la información, entre las que destacan la generación y verificación de firmas digitales, generación de llaves y la creación de bits pseudoaleatorios, por mencionar las más importantes.

La familia de las funciones hash SHA-3 está compuesta por seis funciones. Cuatros de ellas son funciones criptográficas hash llamadas SHA3-224, SHA3-256, SHA3-348 y SHA3-512 y las otras dos funciones son de salida variable, conocidas como XOFs por sus siglas en inglés, llamadas SHAKE128 y SHAKE256, respectivamente. Las funciones de salida variable son diferentes a las funciones hash pero es posible utilizarlas de forma similar, con la posibilidad de adaptarse a las necesidades criptográficas de manera individual.

Las cuatro funciones hash SHA-3 que se muestran en el FIPS-202, complementa las funciones hash especificadas en la familia de SHA-1 y SHA-2. En el caso de

las funciones con salida variable, son funciones, que toman como parámetros de entrada datos binarios, en las cuales es posible obtener como salida, una cadena de longitud que depende de las necesidades o de la aplicación de la misma. Estas funciones son llamadas SHAKE128 y SHAKE256, para los cuales, los sufijos 128 y 256 indican la fortaleza, en términos de seguridad, que tienen estas funciones. En contraste con los sufijos para las funciones hash anteriormente mencionadas, que indican las longitudes de la salida de la función digestiva.

Las seis funciones SHA-3 están diseñadas para proporcionar propiedades especiales, tales como resistencia a las colisiones, así como ataques de pre-imagen y de segunda pre-imagen. Cada función emplea, en la construcción, el mismo tipo de permutación, que el componente principal, el de tipo esponja. Estas permutaciones se especifican como la instancia de una familia de permutaciones, denominada KECCAK-p, para proporcionar flexibilidad, modificar su tamaño y parámetros de seguridad en el desarrollo de cualquier modo adicional a futuro.

Actualmente, se tienen implementaciones de criptografía post-cuántica en sistemas embebidos de propósito específico como, por ejemplo, una matriz de puertas programables (FPGA), microcontroladores AVR, circuito integrado para aplicaciones específicas (ASIC). Sin embargo, no se tienen implementaciones criptográficas, de este tipo de criptografía, en embebidos de gama baja o gama media como, por ejemplo, Edison, Nvidia y Raspberry Pi; lo cual tiene como consecuencia que las propuestas de solución que hacen uso de estos últimos, sean susceptibles ante ataques cuánticos.

Con base en lo anterior, en este trabajo se presenta la implementación de un algoritmo post-cuántico, capaz de ser ejecutado en un sistema embebido de gama media, utilizando los resultados de las simulaciones hechas en los dispositivos de gama alta, para disminuir la probabilidad de éxito de un ataque cuántico dentro de un escenario definido.

2. Metodología

La metodología que se llevó a cabo para el desarrollo de esta investigación está conformada por una serie de cuatro pasos: análisis de la funcionalidad de las

funciones SHA-3, adecuación de una biblioteca que ejecuta la funcionalidad de la SHA-3, elaboración de un programa que mide los tiempos de procesamiento de cada una de las funciones de la SHA-3, ejecución de pruebas de funcionalidad de la SHA-3 en una arquitectura x64 y en ARM y ejecución de pruebas de funcionalidad de la SHA-1 y SHA-2 contra la SHA-3 en una arquitectura ARM. Estos pasos se describen a continuación y se ilustran en la figura 1.

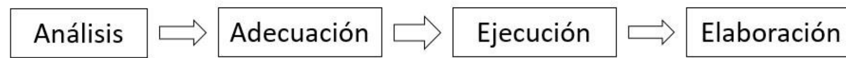

Figura 1 Metodología de trabajo.

Análisis de la funcionalidad

Las permutaciones KECCAK-p están diseñadas para ser los componentes principales de una variedad de funciones criptográficas, incluyendo funciones con llave para el servicio de autenticación y/o el servicio de cifrado. Las seis funciones que conforman la familia de SHA-3 pueden considerarse como modos de operación de la permutación KECCAK-p, la cual se especifica con dos parámetros. El primero es la longitud fija de los datos que se permutan, llamada anchura de la permutación. El segundo, es el número de iteraciones de una transformación interna, denominado ronda.

La ronda de una permutación KECCAK-p consiste en una secuencia de cinco transformaciones llamadas "*The step mapping*". El rango para esta permutación KECCAK-p está comprendida por b bits, para la cual existen siete valores definidos que se muestran en la tabla 1.

Tabla 1 Valores definidos para una permutación KECCAK-p.

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

Donde:

b = Longitud del estado de la permutación KECCAP-p

w = b/25

l = $\log^2$ (b/25)

Si S muestra una cadena que representa el estado, entonces cada uno de sus bits son numerados de 0 a b – 1, con base en la ecuación 1.

$$S = S[0] \parallel S[1] \parallel \cdots \parallel S[b-2] \parallel S[b-1] \tag{1}$$

Es necesario representar los estados de entrada y salida de la permutación como cadenas de bits de longitud b y se presenten los estados de entrada y salida de "*The step mapping*" como matrices de 5 x 5 x w. De ahí, que si A muestra una matriz de 5 x 5 x w que representa el estado, entonces sus índices son los triples enteros (x,y,z) para los cuales 0≤x≤5, 0≤y≤5, 0≤z≤w. El bit que corresponde a (x,y,z) se denota por A[x,y,z]. Con base en lo anterior, la matriz de estado es una representación de la forma tridimensional que muestra la figura 2.

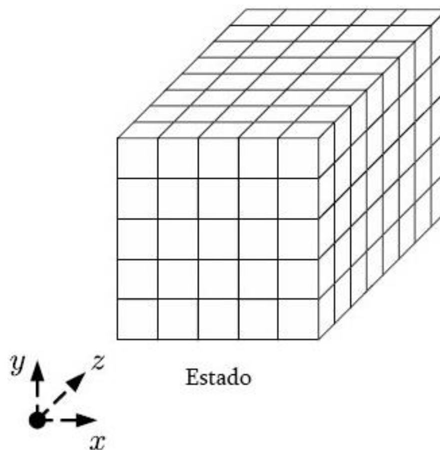

Figura 2 Matrices de estado.

Las submatrices bidimensionales se denominan hojas, planos y rebanadas y las subcadenas unidimensionales se llaman filas, columnas y carriles, como se muestran en la figura 3.

Para convertir una cadena de bits a una matriz de estados, se tiene que S es una cadena de bits de longitud b que representa el estado de la permutación KECCAP-p[b,nr]. La matriz de estado correspondiente, denotada por A se define con base en la ecuación 2: En donde para toda tripleta de enteros (x,y,z) tal que $0 \leq x \leq 5$, $0 \leq y \leq 5$, $0 \leq z \leq w$ se tiene que.

$$A[x, y, z] = S[w(5y + x) + z] \tag{2}$$

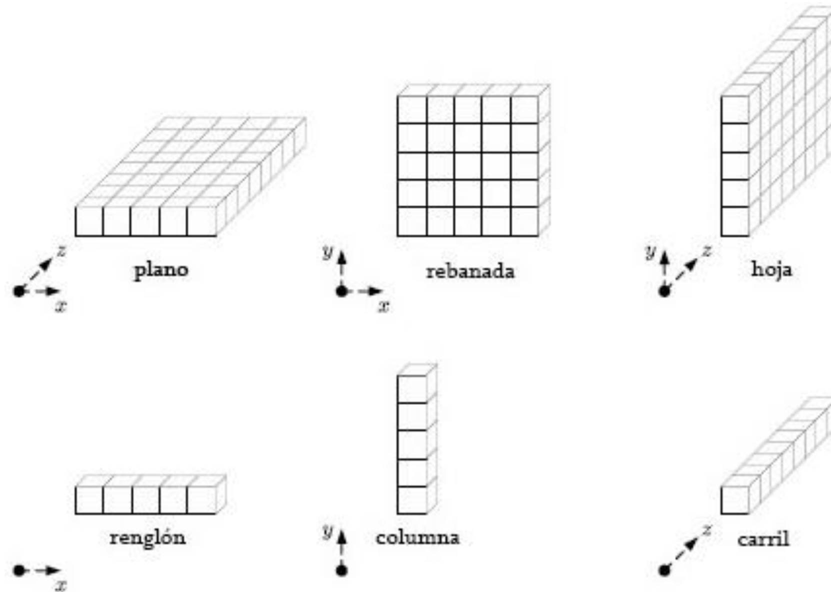

Figura 3 Submatrices bidimensionales y subcadenas unidimensionales.

De igual manera, para convertir una matriz de estado en una cadena de bits, se tiene que A es una matriz de estado. La representación de la cadena correspondiente, denotada por S, puede ser reconstruida a partir de los carriles y planos de la matriz A y se define con base en la ecuación 3, donde para cada par de enteros (i,j) tal que $0 \leq i \leq 5$ y $0 \leq j \leq 5$, define la cadena Lane(i,j).

$$Lane(i,j) = A[i,j,0] \parallel A[i,j,1] \parallel A[i,j,2] \parallel \cdots \parallel A[i,j,w-2] \parallel A[i,j,w-1]. \quad (3)$$

El etiquetado completo de las coordenadas x, y, z para la matriz de estados se muestra en la figura 4.

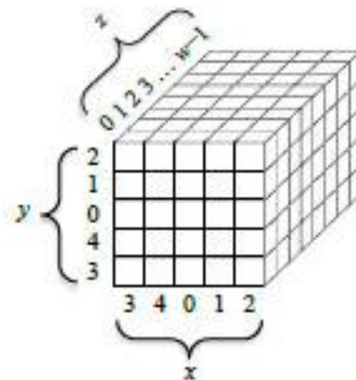

Figura 4 Etiquetado de la matriz de estado.

The Step Mapping

Los cinco "*The Step Mapping*" que comprenden una ronda de la permutación KECCAP-p son denotados por las letras griegas θ, ρ, π, χ, y ι. Los algoritmos, que se muestran a continuación, toman a la matriz de estado, denotada por A, como la entrada y devuelve otra matriz de estado actualizada, denotada por A’, como la salida. El tamaño del estado es un parámetro que no se incluye en la notación ya que b siempre se especifica cuando se invocan los "*Step Mapping*".

El algoritmo para Theta (θ) es mostrado en la figura 5:

- Entrada: Matriz de estado A
- Salida: Matriz de estado A’:
  1. Para todo par (x,z) dado que $0 \leq x \leq 5$, $0 \leq z \leq w$, se tiene que:
     $C[x,z] = A[x,0,z] \oplus A[x,1,z] \oplus A[x,2,z] \oplus A[x,3,z] \oplus A[x,4,z]$
  2. Para todo par (x,z) tal que $0 \leq x \leq 5$, $0 \leq z \leq w$, se tiene que:
     $D[x,z] = C[(x-1) mod\ 5, z] \oplus C[(x+1) mod\ 5, (z-1) mod\ w]$
  3. Para toda tripleta (x,y,z) tal que $0 \leq x \leq 5$, $0 \leq y \leq 5$, $0 \leq z \leq w$, se tiene que:
     $A'[x,y,z] = A[x,y,z] \oplus D[x,z]$

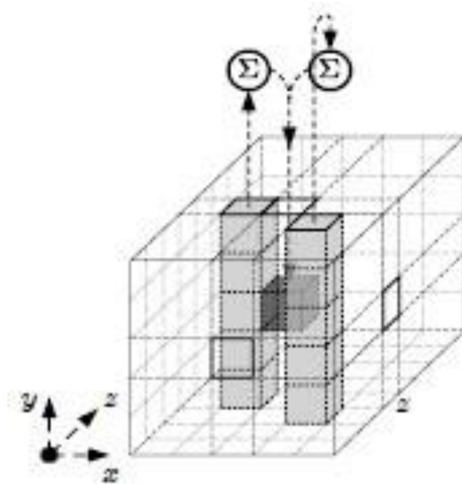

Figura 5 Step Maping-Theta.

El algoritmo para Rho (ρ) es la figura 6:

- Entrada: Matriz de estado A

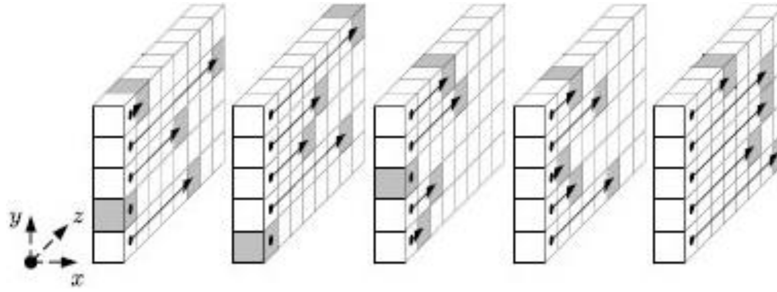

Figura 6 Step Maping-Rho.

- Salida: Matriz de estado A':
    1. Para toda z tal que $0 \leq z \leq w$, se tiene que:
       $A'[0,0,z] = A[0,0,z]$
    2. (x, y) = (1, 0)
    3. Para t de 0 a 23:
        a. Para toda z tal que $0 \leq z \leq w$, se tiene que:
           $A'^{[x,y,z]} = A[x, y, (z - (t+1)(t+2)/2) mod\ w]$
        b. $(x, y) = (y, (2x + 3y) mod\ 5)$
    4. Regresar A'

El algoritmo para Pi (π) es la figura 7:

- Entrada: Matriz de estado A
- Salida: Matriz de estado A'
    1. Para toda tripleta (x,y,z) tal que $0 \leq x \leq 5$, $0 \leq y \leq 5$, $0 \leq z \leq w$, se tiene que:
       $A'[x, y, z] = A[(x + 3y) mod\ 5, x, z]$
    2. Regresar *A'.*

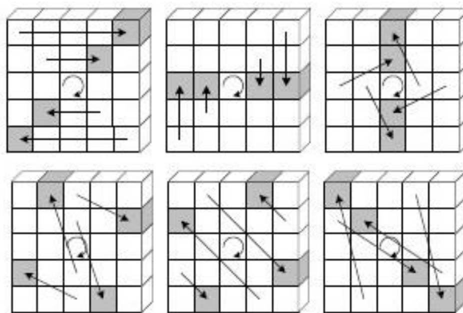
Figura 7 Step Maping-Pi.

El algoritmo para Ji (χ) es la figura 8:

- Entrada: Matriz de estado A
- Salida: Matriz de estado A`:
    - ✓ Para toda tripleta (x,y,z) tal que $0 \leq x \leq 5$, $0 \leq y \leq 5$, $0 \leq z \leq w$, se tiene que:

$$A'^{[x,y,z]} = A[x,y,z] \oplus \left((A[(x+1)mod\ 5,y,z] \oplus 1) \cdot A[(x+2)mod\ 5,y,z]\right)$$

    - ✓ Regresar *A'.*

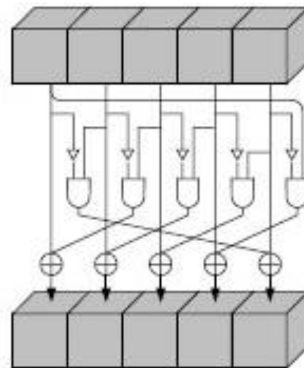

Figura 8 Step Maping - Ji.

Para los algoritmos para Iota (ι) se tiene que el propósito es modificar algunos bits de Lane(0,0) de manera que dependa del índice redondo i_r. "*The Step Mapping*" de ι depende de dos algoritmos. Para el primero, el parámetro de salida determina el valor de l+1 bits de uno de los carriles llamado la constante redonda, designada por RC. Sus especificaciones se detallan a continuación:

- Algoritmo 1.
    - ✓ Entrada: Entero t
    - ✓ Salida: Bit de la constante redonda rc(t):
        - ❖ Si *t mod 255 = 0* entonces regresar 1
        - ❖ R = 10000000
        - ❖ Para cada i de 1 a t mod 255, hacer:
        - ❖ $R = 0 \parallel R;$
        - ❖ $R[0] = R[0] + R[8];$
        - ❖ $R[4] = R[4] + R[8];$

- $R[5] = R[5] + R[8];$
- $R[6] = R[6] + R[8];$
- $R = Trunc_S[R]$
- $R[0]$

- Algoritmo 2.
  - ✓ Entrada: Matriz de estado A e índice redondo i_r
  - ✓ Salida: Matriz de estado A'
    - Para toda tripleta (x,y,z) tal que $0 \leq x \leq 5$, $0 \leq y \leq 5$, $0 \leq z \leq w$, se tiene que
    - $A'^{[x,y,z]} = A[x, y, z]$
    - $RC = 0^w$
    - Para cada j de 0 a l, hacer que $RC[2^j - 1] = rc(j + 7i_r)$
    - Para toda z tal que $0 \leq z \leq w$, se tiene que
    - $A'[0,0,z] = A'[0,0,z] \oplus RC[z]$
    - Regresar *A'*
- Permutación KECCAK-p[b,nr]. Dado una matriz de estado A y un índice redondo i_r, la función redonda Rnd es la transformación que resulta de aplicar "*The Step Mapping*" θ, ρ, π, χ y ι en ese orden, con base en la ecuación 4:

$$Rnd(A, i_r) = \iota\left(\chi\left(\pi\left(\rho(\theta(A))\right)\right), i_r\right) \quad (4)$$

Las permutaciones KECCAK-p[b,nr] consiten en n_r iteraciones de la función Rnd como se muestra en el siguiente algoritmo:

  - ✓ Entrada: Cadena S y número de rondas n_r
  - ✓ Salida: Cadena S':
    - Convertir la cadena S en la matriz de estado A
    - Para toda i_r de $2l + 12 - n_r - 1$, hacer que $A = Rnd(A, i_r)$
    - Convertir la matriz de estado A en la cadena S'
    - Regresa *S'*

Adecuación de una biblioteca que ejecuta la funcionalidad de la SHA-3

Hoy en día, se están desarrollando códigos para utilizar las fortalezas que presentan las funciones de la familia SHA-3. El proyecto de investigación "Open Quantum Safe", ha puesto al alcance de los estudiantes una biblioteca de funciones. Con base en ésta biblioteca se programó una aplicación, en código Phyton, capaz de ejecutarse en un sistema embebido.

La biblioteca se adecuó para poder ejecutarse dentro de la arquitectura ARM. La adecuación se hizo a nivel código, ya que se tenía que modificar uno de los valores que poseía un constructor para que pudiera hacer uso de la función SHA-3 y de todas las opciones que brinda la biblioteca. El código que se utilizó para la adecuación es el siguiente:

```
hashlib.__builtin_constructor_cache['sha3_512'] = sha3.sha3_512
hashlib.new('sha3_512')
<sha3.SHA3512 object at 0x10b381a90>
```

Elaboración de un programa que mide los tiempos de procesamiento de cada una de las funciones de la SHA-3

El programa diseñado a partir de la biblioteca anteriormente mencionada tiene como propósito la obtención de las mediciones en los tiempos de procesamiento, que tardan tanto el embebido como la computadora personal, para resolver las operaciones de las funciones hash de la familia de SHA-3.

El código muestra las líneas de código con las cuales se genera el valor hash de un espacio en blanco y da como resultado el valor hash y el tiempo que tarda el ordenador en procesar esta instrucción:

```
print("***** SHA3 512  *****")
start_time = time.time()
print(hashlib.sha3_512 = time.time() - start_time
print("***** SHA3 224 *****")
start_time = time.time()
print(hashlib.sha3_224 = time.time() - start_time
print("***** SHA3 256 *****")
start_time = time.time()
print(hashlib.sha3_256 = time.time() - start_time
print("***** SHA3 384 *****")
start_time = time.time()
print(hashlib.sha3_384 = time.time() - start_time
```

Ejecución de pruebas de funcionalidad de la SHA-3 en una arquitectura x64 y una ARM y ejecución de pruebas de funcionalidad de la SHA-1 y SHA-2 contra la SHA-3 en una arquitectura ARM

Se hicieron una serie de pruebas para determinar la rapidez de la ejecución del algoritmo SHA-3 dentro de dicha arquitectura. Las pruebas realizadas consistieron en ejecutar cada uno de los miembros de dicha familia, con la finalidad de obtener una comparación de los tiempos de procesamiento que se tienen en una computadora con arquitectura x64. De ahí, que el ordenador que se utiliza es una computadora portátil con un procesado i7 de tercera generación, con velocidad de 2.9 GHz y 8 GB en RAM.

El sistema embebido empleado es una tableta Raspberry Pi 1 modelo B que cuenta con un procesador de un solo núcleo a 700 MHz de velocidad y con 256 MB de memoria RAM. En este embebido se utiliza el sistema operativo Raspbian, dado que es el operativo que la compañía creadora, proporciona para sus dispositivos.

La primera prueba que se hizo fue la comparación de los tiempos de ejecución, medida en segundos, de las funciones SHA-3 224, SHA-3 256, SHA-3 384 y SHA-3 515, para observar qué tiempos tardan en ejecutarse dentro de las dos arquitecturas, con un espacio como la entrada de la función. Posteriormente se repitió la misma prueba, pero cambiando la entrada por una palabra aleatoria. Finalmente, se repitió nuevamente el experimento, pero con un archivo que contenía un millón de letras A como entrada de la función.

La segunda prueba que se realizó fue para obtener la comparación de los tiempos de ejecución de la función, pero ahora en las funciones de salida variable SHAKE128 y SHAKE256, con salidas de tamaño 512, 1024, 2048, 4096 y 8192 bits, respectivamente. Para finalizar, se realizó una prueba de comparación de tiempo de procesamiento en el mismo embebido, comparando a las funciones antecesoras que son SHA-1 y SHA-2 contra SHA-3, con la finalidad de observar, si existe alguna diferencia en los tiempos de ejecución utilizados para resolver las funciones hash.

Los parámetros de salida de estas pruebas permitieron determinar, por medio de un promedio de tiempos, qué tan posible o no es la ejecución de la familia de funciones SHA-3 dentro de sistemas embebidos.

3. Resultados

Los resultados obtenidos por las pruebas antes mencionadas se detallan a continuación:

- Para la ejecución 1, donde se comparan los tiempos de ejecución obtenidos para el procesamiento de las funciones SHA-3 224, SHA-3 256, SHA-3 384 y SHA-3 515, la entrada es "un espacio". El resultado se muestra en la tabla 2.

Tabla 2 Resultados del experimento 1 ronda 1.

Entrada: Espacio			
Tipo	PC	RASPBERRY	Veces más lento
SHA-3 224	0.000036	0.001895	52.63888889
SHA-3 256	0.000202	0.001833	9.074257426
SHA-3 384	0.000078	0.001075	13.78205128
SHA-3 512	0.000092	0.013202	143.5
Promedio	0.000102	0.00450125	44.12990196

- La ejecución 2 es similar a la anterior, pero con la diferencia de que la entrada, ahora es una palabra tomada de forma aleatoria. Los resultados de la prueba se muestran en la tabla 3.

Tabla 3 Resultados del experimento 1 ronda 2.

Entrada: Palabra			
Tipo	PC	RASPBERRY	Veces más lento
SHA-3 224	0.000018	0.000689	38.27777778
SHA-3 256	0.000016	0.000684	42.75
SHA-3 384	0.000016	0.000698	43.625
SHA-3 512	0.000059	0.000853	14.45762712
Promedio	0.00002725	0.000731	26.82568807

- La ejecución 3 es realizada con un archivo que contiene un millón de letras A. Los resultados se muestran en la tabla 4.

Tabla 4 Resultados del experimento 1 ronda 3.

Entrada: Un millón A			
Tipo	PC	RASPBERRY	Veces más lento
SHA-3 224	0.005812	1.576884	271.3152099
SHA-3 256	0.006284	1.654963	263.361394
SHA-3 384	0.008364	2.100136	251.0923003
SHA-3 512	0.016931	3.837384	226.6483964
Promedio	0.00934775	2.29234175	245.229253

- La ejecución 4 consiste en la comparación de los tiempos de ejecución del procesamiento que resulta en las funciones de salida variable con valores de 512, 1024, 2048, 4096 y 8192 bits respectivamente y con un espacio como valor de entrada para la función. Los resultados son expuestos en la tabla 5.

Tabla 5 Resultados del experimento 1 ronda 4.

Tipo	PC	RASPBERRY	Veces más lento
SHAKE128 - 512	0.000106	0.002282	21.52830189
SHAKE128 – 1024	0.000062	0.002734	44.09677419
SHAKE128 – 2048	0.000068	0.002218	32.61764706
SHAKE128 – 4096	0.000125	0.003046	24.368
SHAKE128 – 8192	0.000131	0.004483	34.22137405
SHAKE256 – 512	0.000049	0.005367	109.5306122
SHAKE256 – 1024	0.000047	0.001067	22.70212766
SHAKE256 – 2048	0.000055	0.008259	150.1636364
SHAKE256 – 4096	0.000075	0.001954	26.05333333
SHAKE256 – 8192	0.000107	0.003071	28.70093458
Promedio	0.0000825	0.0034481	49.39827414

- La ejecución 5 repite los mismos parámetros de la prueba anterior, pero con la diferencia de que el valor de entrada de las funciones corresponde a una palabra tomada de forma aleatoria. La tabla 6 muestra los resultados.

Tabla 6 Resultados del experimento 1 ronda 5.

Tipo	PC	RASPBERRY	Veces más lento
SHAKE128 - 512	0.000093	0.000958	10.30107527
SHAKE128 – 1024	0.000023	0.00078	33.91304348
SHAKE128 – 2048	0.000024	0.00106	44.16666667
SHAKE128 – 4096	0.000037	0.001877	50.72972973
SHAKE128 – 8192	0.000048	0.007162	149.2083333
SHAKE256 – 512	0.000017	0.001025	60.29411765
SHAKE256 – 1024	0.000017	0.000957	56.29411765
SHAKE256 – 2048	0.000022	0.001242	56.45454545
SHAKE256 – 4096	0.000032	0.00213	66.5625
SHAKE256 – 8192	0.000048	0.009615	200.3125
Promedio	0.0000361	0.0026806	72.82366292

- La ejecución 6 repite los mismos parámetros de las dos pruebas anteriores, pero el valor de entrada de la función es un archivo de datos que contiene un millón de letras "A". Los resultados se muestran en la tabla 7.

Tabla 7 Resultados del experimento 1 ronda 6.

Tipo	PC	RASPBERRY	Veces más lento
SHAKE128 - 512	0.004848	2.210871	456.0377475
SHAKE128 – 1024	0.004806	1.524076	317.119434
SHAKE128 – 2048	0.004887	1.391101	284.6533661
SHAKE128 – 4096	0.004824	1.423986	295.1878109
SHAKE128 – 8192	0.004822	1.461029	302.9923268
SHAKE256 – 512	0.005995	1.754203	292.6110092
SHAKE256 – 1024	0.005933	1.738503	293.0225855
SHAKE256 – 2048	0.006027	1.736042	288.0441347
SHAKE256 – 4096	0.004804	1.72537	359.1527893
SHAKE256 – 8192	0.004511	1.736697	384.9915761
Promedio	0.0051457	1.6701878	327.381278

- La ejecución 7 consiste en comparar las funciones SHA-1 y SHA-2 contra la función SHA-3 con un espacio como valor de entrada de las funciones. Los resultados son mostrados en la tabla 8.

Tabla 8 Resultados del experimento 2 ronda 7.

Tamaño	SHA-1 y SHA-2	SHA-3	Veces más lento
160	0.000971	-	Sin comparación
224	0.001488	0.001895	1.273521505
256	0.000474	0.001833	3.867088608
384	0.000448	0.001075	2.399553571
512	0.000446	0.013202	29.60089686
Promedio	0.000714	0.00450125	9.285265136

- La ejecución 8 es similar a la prueba anterior, pero con la diferencia de que el valor de entrada es una palabra escogida de manera aleatoria. La tabla 9 muestra los resultados.

Tabla 9 Resultados del experimento 2 ronda 8.

Tamaño	SHA-1 y SHA-2	SHA-3	Veces más lento
160	0.000414	-	Sin comparación
224	0.000325	0.000689	2.12
256	0.000305	0.000684	2.242622951
384	0.000328	0.000698	2.12804878
512	0.000309	0.000853	2.760517799
Promedio	0.00031675	0.000731	2.312797383

- Por último, la ejecución 9 muestra los tiempos de ejecución de la Prueba 8, pero con un archivo que contiene un millón de letras A como valor de entrada. Los resultados se pueden ver en la tabla 10.

Tabla 10 Resultados del experimento 2 ronda 9.

Tamaño	SHA-1 y SHA-2	SHA-3	Veces más lento
160	0.131596	-	Sin comparación
224	0.136864	1.576884	11.52153963
256	0.179939	1.654963	9.197355771
384	0.289569	2.100136	7.25262718
512	0.281525	3.837384	13.6307042
Promedio	0.22197425	2.29234175	10.4005567

4. Discusión

Conforme a los resultados obtenidos en la sección anterior se puede observar lo siguiente:

Los resultados del primer experimento de las ejecuciones uno, dos y tres, que son las funciones SHA-3 224, SHA-3 256, SHA-3 384 y SHA-3 512, muestran los siguientes valores respectivamente: 44.1, 26.8 y 245.2; estas cifras demuestran qué tantas veces es más lento un embebido sobre un ordenador personal a la hora de la ejecución del código. Las comparaciones fueron hechas en valores de microsegundos por lo que los resultados más altos son de segundos, como muestra en la gráfica de la figura 9.

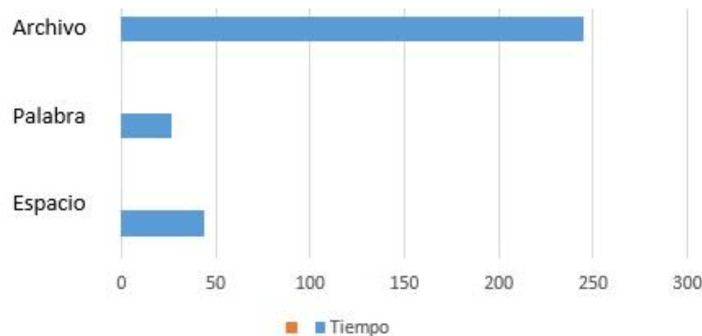

Figura 9 Resultados de experimento 1, ronda 1, 2 y 3.

El valor más alto representa el proceso de un archivo que contiene un millón de caracteres y el tiempo utilizado para ejecutar el proceso es de entre 1 y 3

segundos por lo que, a nivel computacional, es posible ejecutar una función hash post-cuántica en un embebido de bajos recursos computacionales.

Los resultados para las ejecuciones cuatro, cinco y seis, que son las funciones de salida variable SHAKE128 y SHAE256, muestran los siguientes valores: 49.3, 72.8 y 327.3. Los valores se muestran en la gráfica de la figura 10.

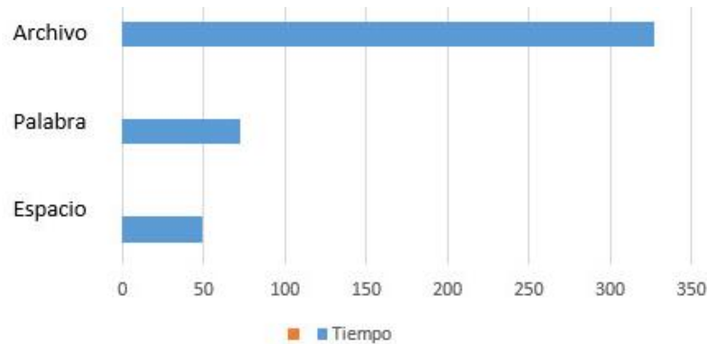

Figura 10 Resultados de experimento 1, ronda 4, 5 y 6.

El valor más alto representa el proceso de un archivo, igual que en las ejecuciones anteriores, y el tiempo utilizado para ejecutar el proceso es de un segundo por lo que es más que posible tener funciones hash de salida variable dentro de los embebidos de bajos recursos. Las salidas pueden ser tan grandes como el usuario las requiera y el dispositivo podrá tener el resultado de la función hash.

Por último, el resultado del experimento 2 de las ejecuciones siete, ocho y nueve, que es la comparación entre SHA-1 y SHA-2 contra SHA-3 dentro del mismo embebido, tiene los siguientes valores: 9.2, 2.3 y 10.4. Los resultados se muestran en figura 11.

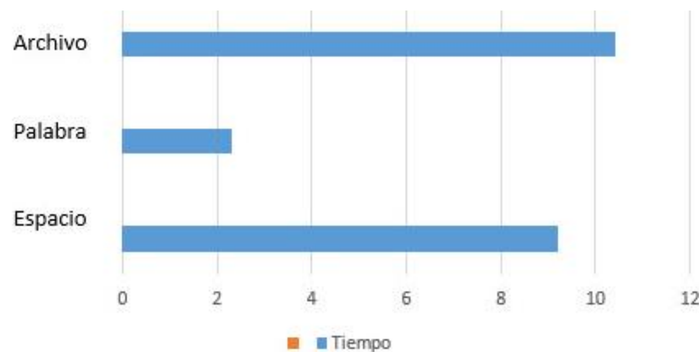

Figura 111 Resultados de experimento 2, ronda 7, 8 y 9.

Los valores obtenidos en este último experimento demuestran que una función post-cuántica puede ser ejecutada de manera exitosa dentro de los embebidos de bajos recursos computacionales. Al hacer la comparación con las funciones SHA-1 y SHA-2, que actualmente se utilizan dentro de los medios digitales, es posible observar que los valores de ejecución de la función SHA-3 no son tan altos, por lo que es posible utilizar e implementar aplicaciones que utilicen estas funciones hash en vez de sus antecesoras.

5. Bibliografía y Referencias

[1] Ghani Aziz, Mochamad Vicky, et al. (2013, noviembre). Hash MD5 Function Implementation at 8-bit Microcontroller. Para el 2013 Joint International Conference on Rural Information & Communication Technology and Electric-Vehicle Technology. Bandung, Indonesia.

[2] Kahri, Fatma, et al. (2013, marzo). An FPGA implementation of the SHA-3: The BLAKE Hash Function. Para el 10th International Multi-Conference on Systems, Signal & Devices. Hammamet, Túnez.

[3] Kahri, Fatma, et al. (2015, marzo). Efficient FPGA Hardware Implementation of Secure Hash Function SHA-256/Blake-256. Para el 12th International Multi-Conference on Systems, Signal & Devices. Mahdia, Túnez.