Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Tipo de artículo: Artículo de revisión
Temática: Seguridad informática
Recibido: 03/02/2020 | Aceptado: 06/04/2020

La aritmética modular como mecanismo de seguridad y vulnerabilidad en el sistema criptográfico RSA

Modular arithmetic as a security and vulnerability mechanism in the RSA cryptographic system

Hubner Janampa Patilla[1*] https://orcid.org/0000-0003-3110-194X

[1] Unidad de Investigación e Innovación, Facultad de Ingeniería, Universidad Nacional de San Cristóbal de Huamanga, Perú. hubner.janampa@unsch.edu.pe

*Autor para la correspondencia: hubner.janampa@unsch.edu.pe

RESUMEN

La aritmética modular es la base matemática para el mecanismo de seguridad del sistema criptográfico de clave pública asimétrica RSA. Se mostrará los fundamentos matemáticos de la aritmética modular para el cifrado y descifrado de un texto plano, se definió la base sobre el uso de las operaciones aritméticas de adición, multiplicación y exponenciación sobre la aritmética módulo-n. Se mencionará los criterios de generación de dos números primos p y q para calcular la clave pública y privada mediante el algoritmo RSA. Se mostrará la importancia de la cantidad de cifras significativas de estos números primos como un mecanismo de seguridad para evitar la factorización de los mismos, además de las vulnerabilidades y ataques a nivel de protocolo; a nivel matemático y de canal lateral, que el sistema RSA posee debido al uso de la aritmética

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

módulo-n. Se desarrolla e implementa en base al lenguaje de programación imperativo Python, la ejecución matemática del algoritmo RSA para la generación de las claves, el cifrado y el descifrado.
Palabras claves: aritmética modular; cifrado; descifrado; RSA; números primos

ABSTRACT

Modular arithmetic is the mathematical basis for the security mechanism of the cryptographic system of RSA asymmetric public key. We will show the mathematical foundations of modular arithmetic for the encryption and decryption of a plain text, we define the basis on the use of arithmetic addition operations, multiplication and exponentiation on module-n arithmetic. We will mention the criteria for generating two prime numbers p and q to calculate the public and private key using the RSA algorithm. We will show the importance of the number of significant figures of these prime numbers as a security mechanism to avoid factoring them, in addition to vulnerabilities and attacks at the protocol level; to mathematical and lateral channel level, which the RSA system has due to the use of modulo-n arithmetic. I know develops and implements based on the imperative programming language Python, the mathematical execution of the RSA algorithm for key generation, encryption and decryption.
Keywords: modular arithmetic; encryption; decryption; RSA; prime number

Introducción

Casi todos los algoritmos criptográficos, tanto cifrados simétricos como cifrados asimétricos son basados en la aritmética modular dentro de un número finito de elementos. La mayoría de los conjuntos infinitos que usamos son el conjunto de números naturales $\mathbb{N}$ o el conjunto de números reales $\mathbb{R}$. En este estudio

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

presentamos a la aritmética modular como un mecanismo de seguridad y vulnerabilidad sobre un conjunto finito de enteros $\mathbb{Z}$ para la criptografía asimétrica RSA.

La aritmética modular

Tal como define (Koscielny et al., 2013), sean a, b y n números naturales (a, b, $n \in \mathbb{N}$; $n = 0$). Si dos enteros, a y b, tienen el mismo resto cuando se dividen entre n, entonces se dice que son congruentes módulo n. Denotamos esto por $a \equiv b \; mod \; n$. En símbolos:

$$a \equiv b \; mod \; n \leftrightarrow (a \; mod \; n) = (b \; mod \; n)$$

La relación de congruencia es una relación de equivalencia, para todos los números naturales a, b, c, n la relación de congruencia, denotada por ≡ , tiene las siguientes propiedades:

1. reflexividad: $a \equiv a \; mod \; n$
2. simetría: $a \equiv b \; mod \; n \rightarrow b \equiv a \; mod \; n$
3. transitividad: $a \equiv b \; mod \; n \wedge b \equiv c \; mod \; n \rightarrow a \equiv c \; mod \; n$

Debido a las propiedades de las relaciones de equivalencia, cada relación de congruencia módulo n determina una partición del conjunto de números naturales en clases de equivalencia disjuntas. Estas clases están formadas por números naturales congruentes módulo n. Para cualquier clase de equivalencia de a, existe algún a_0 del conjunto $\{1,2, \ldots, n-1\}$, tal que $a \equiv a_0 \; mod \; n$. En la que a_0 se llama representación canónica de la clase de equivalencia de a (la clase $[a]$). El conjunto $\mathbb{Z}_n = \{0,1,2 \ldots, n-1\}$ se llama conjunto de números naturales módulo n (Koscielny et al., 2013).

Definición 1 (El anillo de entero $\mathbb{Z}_n$). Sea el conjunto $\mathbb{Z}_n = \{0,1,2 \ldots, n-1\}$, en la cual definimos dos operaciones $+$ y $-$ para todos los $a, b \in \mathbb{Z}_n$ de modo que (Paar and Pelzl, 2010) :

1. $a + b \equiv c \; mod \; n, (c \in \mathbb{Z}_n)$
2. $a \times b \equiv d \; mod \; n, (d \in \mathbb{Z}_n)$

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Definición 2 (Grupo de clase de residuos primos módulo n, $\mathbb{Z}_n^*$). Por lo tanto, los elementos del grupo de unidades de $\mathbb{Z}_n$ son representados por los números primos a n (Delfs and Kneb, 2007).

$$\mathbb{Z}_n^* \coloneqq \{[a] | 1 \leq a \leq n, mcd(a,n) = 1\}$$

Se llama el grupo de clase de residuo primario módulo n. La cantidad de elementos en $\mathbb{Z}_n^*$ (también llamado el orden de $\mathbb{Z}_n^*$) es el número de enteros en el intervalo $[1, n-1]$ que son primos para n.

Para un número natural n positivo fijo, se cumple la siguiente implicación:

$$a = q \times n + r \wedge 0 \leq r < n \rightarrow a \equiv r \, mod \, n$$

en la cual r es el resto de a módulo n.

Deje que $m \geq 1$ sea un entero, decimos que los enteros a y b son congruente módulo m, si su diferencia $a - b$ es divisible por m. Nosotros escribimos $a \equiv b \, mod \, m$ para indicar que a y b son congruentes módulo m. El número m se llama módulo (Hoffstein et al., 2008).

Tal como define (Kurose and Ross, 2013), las formulas de la aritmética modular para sumar y multiplicar se facilita con las siguientes igualdades:

$$[(a \, mod \, n) + (b \, mod \, n)] \, mod \, n = (a + b) \, mod \, n$$
$$[(a \, mod \, n) - (b \, mod \, n)] \, mod \, n = (a - b) \, mod \, n$$
$$[(a \, mod \, n) \times (b \, mod \, n)] \, mod \, n = (a \times b) \, mod \, n$$
$$[(a \, mod \, n)^d] \, mod \, n = (a^d) \, mod \, n$$

La suma, la resta y la multiplicación se comportan de la misma manera que para residuos, o que para la aritmética de enteros. Lo habitual es tener leyes de identidad, conmutativas y distributivas, de modo que el conjunto de clases de residuos forme un anillo, denotado por $\mathbb{Z}_N$, para el módulo N. Por lo tanto; las leyes sobre el anillo se definen como (Tilborg and Jajodia, 2011):

1. $N \equiv 0 \, (mod \, N)$

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

2. $A + 0 \equiv A\ (mod\ N)$
3. $1 \times A \equiv A\ (mod\ N)$
4. $si\ A \equiv B\ (mod\ N), entonces\ B \equiv A\ (mod\ N)$
5. $si\ A \equiv B\ (mod\ N) \wedge B \equiv C\ (mod\ N), entonces\ A \equiv C\ (mod\ N)$
6. $si\ A \equiv B\ (mod\ N) \wedge C \equiv d\ (mod\ N), entonces\ A\ +\ C \equiv B\ +\ d\ (mod\ N)$
7. $si\ A \equiv B\ (mod\ N) \wedge C \equiv d\ (mod\ N), entonces\ A \times C \equiv B \times d\ (mod\ N)$
8. $A + B \equiv B + A\ (mod\ N)$
9. $A \times B \equiv B \times A\ (mod\ N)$
10. $A + (B + C) \equiv (A\ +\ B) + C\ (mod\ N)$
11. $A \times (B \times C) \equiv (A \times B) \times C\ (mod\ N)$
12. $A \times (B + C) \equiv (A \times B)\ +\ (A \times C)\ (mod\ N)$

La noción de los números primos, tal como manifiesta (Pace, 2012), son números mayores que 1, que son exactamente divisibles solo por 1 y ellos mismos, tienen una larga historia en matemáticas. Los números primos corresponden estrechamente a la noción de números atómicos, números que no pueden ser construidos como un producto de números más pequeños. Aunque fueron estudiados por los griegos desde la antigüedad, se han encontrado aplicaciones prácticas muy recientemente. Hoy en día, los números primos juegan un papel crucial en los algoritmos de encriptación, y la compra segura en línea en una tienda virtual, no sería posible sin ellos.

Los enteros a y b son primos relativos si y sólo si, $mcd(a, b)\ =\ 1$, los enteros $a_1, a_2, a_3, \dots, a_n$ son primos relativos en pares si y sólo si, el $mcd(a_i, a_j)\ =\ 1$, para todos los enteros i y j con $1 \leq i, j \leq\ n$ e $i \neq j$.

Según (Dooley, 2018), un campo finito es una estructura algebraica con varias características. Primero, hay un conjunto de números que componen los elementos del campo. Utilizaremos un conjunto de enteros positivos $\{0,1,2,3, \dots, p-1\}$ donde p es un número primo o una potencia de un número primo. Habrá exactamente p elementos en el conjunto, esto se llama el orden del campo. También declararemos dos operaciones $+$ y $\times$, que generalmente se llaman suma y multiplicación, y las restringiremos de modo que si a, b están en el conjunto de enteros, entonces también lo está $a\ +\ b$; esto se llama la propiedad de la cerradura del álgebra, por lo cual garantizamos que esto funcionará realizando el módulo de las sumas y multiplicaciones de p.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Según (Smart, 2016), sobre el teorema de los números primos, antes de analizar estos algoritmos, debemos analizar algunas heurísticas básicas sobre números primos. Un famoso resultado en matemáticas, conjeturado por Gauss, después de un extenso cálculo a principios de 1800, es el teorema del número primo que se expresa como el teorema 1.

Teorema 1 (Teorema de los números primos). La función $\pi(X)$ cuenta con un número de primos menores que X, de donde tenemos la aproximación

$$\pi(X) \approx \frac{X}{\log(X)}$$

Esto significa que los números primos son bastante comunes. Como muestra, el número de primos menores que 2^{1024} es aproximadamente 2^{1014}. El teorema del número primo también nos permite estimar la probabilidad de un número primo aleatorio, si p es un número elegido al azar, entonces la probabilidad de que sea primo es aproximadamente $\frac{1}{\log(p)}$.

Según (Koblitz, 1994), hay muchas situaciones en las que uno quiere saber si un número n es primo en el criptosistema de clave pública RSA, y en varios criptosistemas basados en el problema del logaritmo discreto para campos finitos. Una primera interpretación de lo que esto significa es elegir un número entero impar grande n_0 usando un generador aleatorio de dígitos y luego probando $n_0, n_0 + 2, \ldots$ para la primalidad hasta que obtengamos el primer primo que es menor a n_0. Un segundo tipo es el uso de la prueba de primalidad para determinar si un entero especial de cierto tipo es primo.

El lema de Euclides manifiesta que para todos los enteros a, b y c, si $mcd\ (a, c) = 1$ y si $a|bc$, entonces $a|b$ (Epp, 2019).

Un método para calcular el máximo común divisor de dos enteros es mediante el algoritmo euclidiano, que es un algoritmo antiguo, conocido y eficiente para encontrar el máximo común divisor. El algoritmo euclidiano mostrado en el algoritmo 1, tabla 1, se basa en el hecho de que si $r = a \ (mod\ b)$, entonces:

$$mcd(a, b) = mcd(b, r)$$

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Para todos los enteros a y m, si $mcd(a,m) = 1$, entonces existe un entero s tal que $as \equiv 1\ (mod\ m)$. El entero s se llama el inverso de a módulo m. Una extensión del algoritmo de Euclides se define como, si a y b son primos relativos enteros, es decir $mcd(a;\ b) = 1$, entonces existen los enteros s y t tal que $a \times s + b \times t = 1$ (Epp, 2019).

Tabla 1 - Algoritmo de Euclides.

Algoritmo 1: Algoritmo de Euclides
Entrada: a, b. **Salida:** b, el máximo común divisor.
1: $Procedure\ (a,b)$ 2: $r \leftarrow a\ mod\ b$ 3: $while\ r \neq 0\ do$ 4: $a \leftarrow b$ 5: $b \leftarrow r$ 6: $r \leftarrow a\ mod\ b$ 7: $end\ while$ 8: $return\ b$ 9: $End\ Procedure$

El teorema de la cancelación para la congruencia modular se define para todos los enteros a, b, c y n con $n > 1$, si $mcd(c,n) = 1$ y $ac \equiv bc\ (mod\ n)$, entonces $a \equiv b\ (mod\ n)$. Para todos los enteros a, b y c, si $mcd(a,c) = 1$ y $ab|c$, entonces $a|b$ (Epp, 2019).

Ahora el pequeño teorema de Fermat se define si, p es cualquier número primo y a es cualquier entero tal que $p \nmid a$, entonces $a^{p-1} \equiv 1\ (mod\ p)$. Según (Joye and Tunstall, 2012), la operación principal del esquema de la firma RSA es una exponenciación modular en $(\mathbb{Z} = \mathbb{NZ})^*$. Es decir, una firma s se genera a partir de un mensaje m al computar $s = \mu(m)^d\ (mod\ N)$, donde d es la clave privada, N es el producto de dos números primos grandes, y μ es una función de relleno apropiada. Esta firma se puede verificar marcando si $\mu(m)$ es igual a $s^e\ (mod\ N)$. Definimos $d \equiv e^{-1}(mod\ \emptyset(N))$ donde $\emptyset$ es la función de Euler.

Según manifiesta (Biggs, 2008), la función $\emptyset$ se define para cualquier entero positivo n, como la cantidad de números enteros x en el rango $1 \leq x \leq n$, tal que su $mcd\ (x,n) = 1$, que se denota por $\emptyset(n)$. Se desprende de lo declarado que $\emptyset(n)$ es también el número de elementos invertibles de $\mathbb{Z}_n$. El conjunto de enteros en las operaciones de suma y multiplicación con módulo n, es un anillo que lo denotamos por $\mathbb{Z}_n$. En general, no

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

todos los elementos distintos de cero de $\mathbb{Z}_n$ tienen un inverso multiplicativo, un elemento distinto de cero $x \in \mathbb{Z}_n$ tiene un inverso multiplicativo $x^{-1} \in \mathbb{Z}_n$, si y solo si $mcd\ (x, n) = 1$.

Rivest-Shamir-Adleman (RSA)

Tal como afirma (Vaudenay, 2006), RSA es el primer criptosistema de clave pública que todavía es seguro y es usado, y que fue inventado por Ronald Rivest, Adi Shamir y Leonard Adleman, cuyas iniciales llevaron al nombre del criptosistema RSA. Fue publicado en 1978 según la referencia (Rivest et al., 1978), en la revista Communications de la ACM. Desde entonces, este algoritmo se ha adaptado, generalizado y transformado en varios estándares.

RSA es el cifrado asimétrico más conocido y utilizado, este se utiliza en una variedad de protocolos diferentes en el mundo de la seguridad informática, tal como el SSL[1], CDPD[2], y PGP[3]. RSA se ha usado en muchas aplicaciones diferentes hasta la fecha, y es probable que se use en muchas aplicaciones en el futuro (Daswani et al., 2007).

La aplicación de la aritmética modular a la criptografía RSA, define que para un texto plano, M, este se convierte en el texto cifrado, C, donde para e se cumple que el $mcd((p-1)(q-1), e) = 1$, donde e debe estar entre $1 < e < (p-1)(q-1)$, la ecuación que cifra el mensaje es la siguiente:

$$C = M^e\ (mod\ p \times q)$$

Para descifrar el mensaje, se necesita calcular un número d que es un inverso positivo al e módulo $(p-1)(q-1)$, esto se expresa como : $d \times e\ (mod\ (p-1)(q-1)) = 1$. La fórmula para obtener el texto plano, M, se muestra a continuación:

$$M = C^d\ (mod\ p \times q)$$

Según (Delfs and Kneb, 2007), el criptosistema RSA se basa en hechos de la teoría de números elementales, que se conocen desde hace 250 años. Para implementar un criptosistema RSA, se tienen que multiplicar dos números primos muy grandes y hacer público su producto n, que es parte de la clave pública, mientras que los factores

de n se mantienen en secreto y se utilizan como la clave secreta. La idea básica es que los factores de n no se pueden recuperar de n, de hecho, la seguridad de la función de cifrado RSA depende tremendamente de la dificultad de la factorización, pero esta equivalencia no está probada.

Según (Kranakis, 1986), fue solo recientemente que la criptografía de clave pública se convirtió en una precisa asignatura matemática, principalmente como respuesta a la creciente necesidad de seguridad en la transmisión de información a través de los medios electrónicos, la principal nueva idea era basar la seguridad de los criptosistemas en la intratabilidad del número.

Tal como afirma (Salomaa, 1996); el texto sin formato o texto nativo se codifica como un número decimal, o podemos utilizar igualmente la representación binaria si queremos, el número se divide en bloques de tamaño adecuado, los bloques están encriptados por separado. Un tamaño adecuado para los bloques es el número entero único i, que satisface la desigualdad $10^{i-1} < n < 10^{i}$; en algunos casos, uno puede elegir $i - 1$ como el tamaño del bloque, o asegurarse de que cada bloque sea menor que n, si la unicidad del descifrado es importante.

Tal como manifiesta (Daras and Rassias, 2015), en algunas aplicaciones, el mensaje m se cifra como $c \equiv m^e \ (mod\ N)$ y g^{m^e} se hace público. Recuperar m de g^{m^e}, o verificar las propiedades de m es el problema.

El criptosistema RSA es un criptosistema de clave asimétrica donde dos claves, una clave pública y una clave privada son usados. La clave pública se utiliza para el cifrado y la clave privada se utiliza para el descifrado (Nisha and Farik, 2017).

El criptosistema de clave pública RSA se resume en la tabla 2. La clave secreta de Bob es un par de primos grandes p y q. Su clave pública es el par (N, e) que consiste del producto $N = p \times q$ y un exponente de cifrado e que es primo relativo a $(p-1) \times (q-1)$. Alice toma su texto sin formato y lo convierte en un entero m entre 1 y N. Ella cifra m calculando la cantidad

$$c \equiv m^e \ (mod\ N)$$

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

El número entero c es su texto cifrado, que le envía a Bob. Bob resuelva la congruencia $x^e \equiv c\ (mod\ N)$ para recuperar el mensaje m de Alicia, porque Bob conoce la factorización $N = p \times q$. Eva, por otro lado, puede interceptar el texto cifrado c, pero a menos que ella sepa factorizar N, presumiblemente tiene dificultad para tratar de resolver la congruencia

$$x^e \equiv c\ (mod\ N)$$

En el algoritmo RSA existen dos componentes interrelacionados que son la elección de la clave pública y privada. Recordar que el algoritmo RSA es un cifrador de bloque en el que el texto nativo y el texto cifrado son enteros entre 0 y $n - 1$, para algún n. Para un texto nativo M, el cifrado C, y el descifrado M, se realiza

$$C = M^e\ mod\ n$$

$$M = C^d\ mod\ n = (M^e)^d\ mod\ n = M^{ed} mod\ n$$

Ambos; tanto el emisor como el receptor deben conocer el valor de n y el valor de e. Solo el receptor conoce el valor de d. El algoritmo de clave pública genera una clave pública $KU = \{e, n\}$, y una clave privada de $KR = \{d, n\}$, además este algoritmo de cifrado de clave pública debe cumplir las siguientes propiedades:

1. Es posible encontrar valores de e, d, y n tales que $M^{e \times d} = M\ mod\ n$, para todo $M < n$.
2. Es relativamente fácil calcular M^e y C^d para todos los valores de $M < n$.
3. Es impracticable determinar d dado e y n.

Según (Stallings, 2014), los dos primeros requisitos son fáciles de satisfacer. El tercer requisito se puede satisfacer con valores altos de e y n. Según (Hoffstein et al. ,2008), la seguridad de RSA depende de la siguiente dicotomía:

1. Partida: Sea p y q números primos grandes, y $N = p \times q$, y que e y c sean enteros.
2. Problema: Resuelva la congruencia $x^e \equiv (mod\ N)$ para la variable x.
3. Solución: Bob, que conoce los valores de p y q, puede resolver fácilmente x como descrito en la Proposición 1.
4. Dificultad: Eva, que no conoce los valores de p y q, no puede encontrar fácilmente x.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

5. Dicotomía: Resolver $x^e \equiv c\ (mod\ N)$ es fácil para una persona que posee cierta información adicional, pero aparentemente es difícil para todas las demás personas.

Proposición 1. Sea p y q primos distintos y que $e \geq 1$ y satisfaga

$$mcd(e, (p-1)\ \times (q-1))\ =\ 1$$

Sabiendo que e tiene un módulo inverso a $(p-1) \times (q-1)$, digamos $d \times e \equiv 1\ (mod\ (p-1) \times (q-1))$. Entonces la congruencia

$$x^e \equiv c\ (mod\ p \times q)$$

tiene la solución única

$$x \equiv c^d\ (mod\ p \times q)$$

La (Fig. 1), presenta el funcionamiento del algoritmo RSA, este algoritmo no se puede usar directamente, ya que se describe en un formato matemático, su configuración se establece la de la siguiente forma (Vaudenay, 2006):

1. Parámetro público: un entero par s.
2. Configuración: encuentre dos números aleatorios primos diferentes p y q de tamaño $\frac{s}{2}$ bits.

   Establecer $N\ =\ p \times q$. Elija un e aleatorio hasta que $mcd\ (e, (p-1)(q-1))\ =\ 1$. Establecer $d\ =\ e^{-1}\ mod\ ((p-1)(q-1))$.
3. Mensaje: un elemento $x \in \{1, \ldots, N-1\}$.
4. Clave pública: $Kp\ =\ (e; N)$.
5. Clave secreta: $Ks\ =\ (d; N)$.
6. Cifrado: $y\ =\ x^e\ mod\ N$.
7. Descifrado: $x\ =\ y^b mod\ N$.

Tabla 2 - Creación de las claves RSA, cifrado, y descifrado.

Bob	**Alicia**
Creación de Claves	

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Elija los números primos secretos p y q. Elige el exponente e para la encriptación con $mcd(e,(p-1)(q-1))=1$. Publicar $N=p\times q$ y e.	
Encriptación	
	Elija el texto sin formato m. Use la clave pública de Bob (N,e) para calcular $c\equiv m^e(mod\ N)$. Enviar el texto cifrado c a Alicia.
Desencriptación	
Calcular d satisfaciendo $e\times d\ \equiv 1\ (mod\ (p-1)(q-1))$. Calcule $m'\equiv c^d(mod\ N)$, entonces m' es igual al texto sin formato m.	

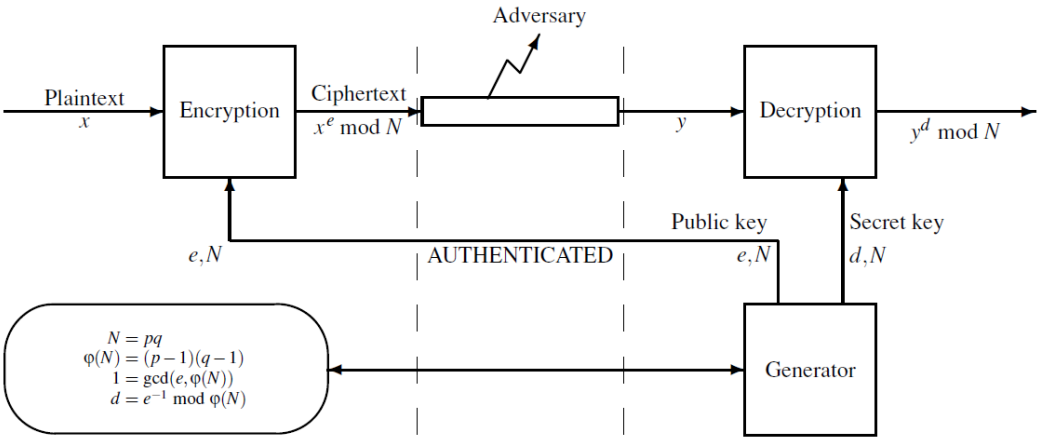

Fig. 1 - Criptosistema RSA simple.

Vaudenay, 2006.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

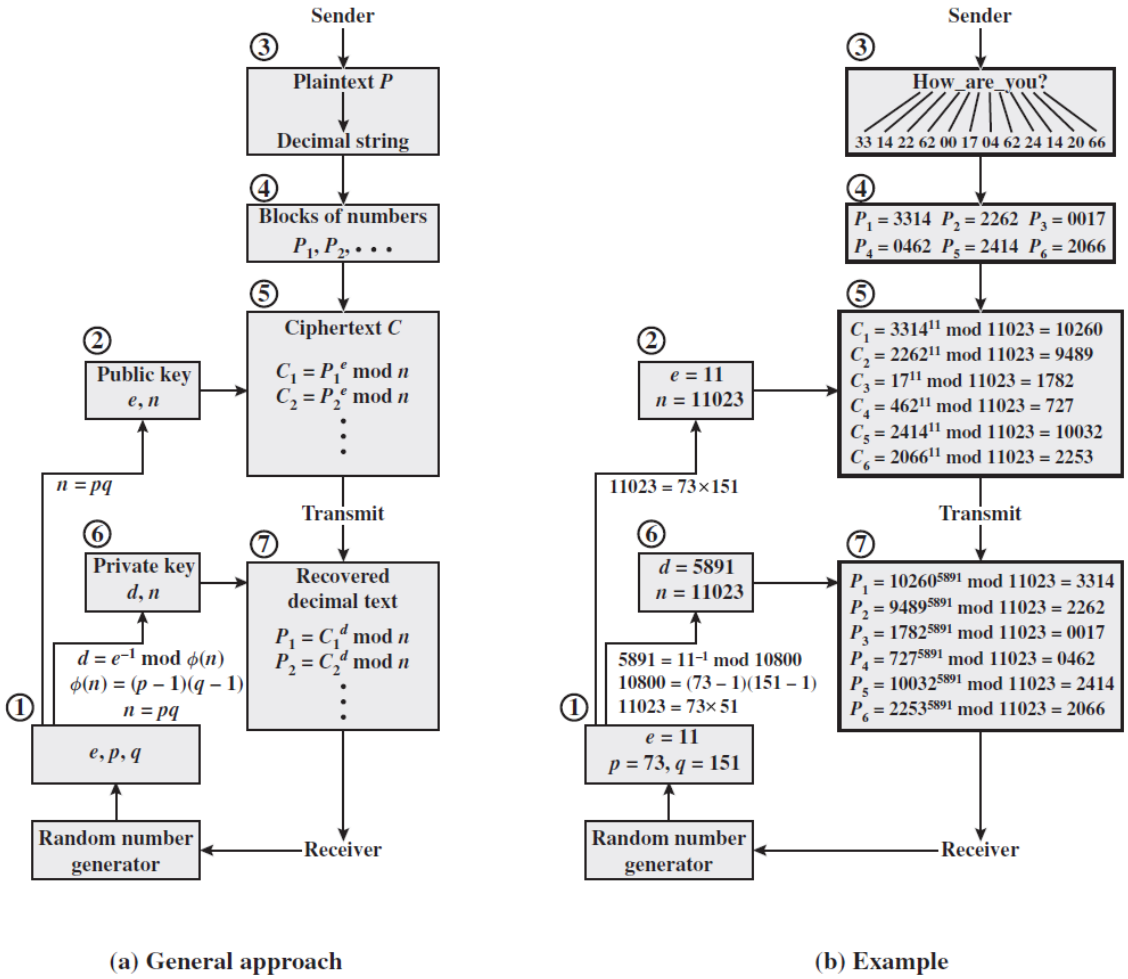

Fig. 2 - Procesamiento de bloques múltiples en RSA, (a) Enfoque general y (b) Ejemplo de ejecución (Stallings, 2014).

Tabla 3 - Números RSA.

Número RSA	Dígitos	Bits	Premio	Expirado	Equipo que resolvió
RSA-100	100	330		Abril, 1991	A.K. Lenstra
RSA-100	110	364		Abril, 1992	A.K Lenstra y M. Manasse
RSA-100	120	397		Junio, 1993	T. Denny y otros
RSA-100	129	426	$ 0	Abril, 1994	A.K Lenstra y otros
RSA-100	130	430		10 Abril, 1996	A.K Lenstra y otros
RSA-100	140	463		2 Febrero, 1999	H.J. te Riele y otros

RSA-100	150	496		16 Abril, 2004	K. Aoki y otros
RSA-100	155	512		22 Agosto, 1999	H.J. te Riele y otros
RSA-100	160	530		1 Abril, 2003	J. Franke y otros
RSA-100	174	576	$ 10 000	3 Diciembre, 2003	J. Franke y otros
RSA-100	193	640	$ 20 000	2 Noviembre, 2005	J. Franke y otros
RSA-100	200	663		9 Mayo, 2005	J. Franke y otros
RSA-100	212	704	$ 30 000	Premio retirado	
RSA-100	232	768	$ 50 000	Premio retirado	
RSA-100	270	896	$ 75 000	Premio retirado	
RSA-100	309	1024	$ 100 000	Premio retirado	
RSA-100	463	1536	$ 150 000	Premio retirado	
RSA-100	617	2048	$ 200 000	Premio retirado	

Koscielny et al., 2013.

Seguridad y ataques del sistema RSA

En la criptografía moderna a menudo emplea la frase del problema de la matemática computacionalmente difícil, un problema computacional se considera difícil o intratable o inviable si no existe un algoritmo conocido que resuelva todas las instancias del problema con un polinomio superior limitado a los recursos requeridos, en particular el tiempo polinómico. Cualquier criptosistema, cuya seguridad se basa en un problema insoluble se considera segura (Koscielny et al., 2013).

En 1991, se inició un concurso mundial, el RSA Factoring Challenge, organizado por la RSA Security para estimular a la comunidad científica a estudiar eficientemente los algoritmos para factorizar enteros grandes.

Se publicó una lista de números, que fueron los productos de dos primos. Estos números se llamaron números RSA. Se ofreció un premio en efectivo por factorizar algunos de ellos, tal como se muestra en la tabla 3, el concurso duró dieciséis años y se cerró oficialmente en mayo del 2007 (Koscielny et al., 2013).

Según (Johnsonbaugh, 2018), la seguridad del sistema RSA para cifrar se basa principalmente en el hecho de que, hasta ahora, no se cuenta con un algoritmo eficiente para factorizar enteros; es decir, no se conoce un algoritmo para factorizar enteros de d dígitos en tiempo polinomial, $O(d^k)$.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

La notación big-O se usa para el rendimiento y/o la complejidad de ejecución de un algoritmo, y la criptografía no es ajena a utilizar esta notación asintótica, más aún cuando se mide el rendimiento de este en un tiempo de ejecución exponencial, (Koblitz, 2004) define que para todo $n > n_0$, dos funciones $f(n)$ y $g(n)$ están definidos y toman valores positivos, y para cierta constante C, se satisface la desigualdad $f(n) \leq C \times g(n)$, entonces decimos que $f = O(g)$.

Tal como (Parthajit, 2018) manifiesta que factorizar el producto de dos números primos grandes es la forma directa de atacar a RSA, toda la seguridad de RSA depende de la dificultad de factorizar. Técnicamente hablando, dado n como un entero de b bits, toda la factorización de los algoritmos toman un tiempo del orden $O(2^{\frac{b}{k}})$, para todas las instancias de un k razonablemente pequeño, entonces afirmamos que RSA es seguro.

Los ataques contra RSA desde que se inventó en 1977, suelen explotar las debilidades en la forma en que se implementó RSA en sí, y podemos considerar tres tipos de ataques generales:

1. Ataques de protocolo
2. Ataques matemáticos
3. Ataques de canal lateral

Ataques de protocolo

Los ataques de protocolo aprovechan las debilidades en la forma en que se utiliza RSA. Ha habido varios ataques de protocolo a lo largo de los años. Entre los más conocidos están los ataques que explotan la maleabilidad de RSA.

Según (Paar and Pelzl, 2010), en criptografía se dice que un esquema es maleable, si el atacante es capaz de transformar el texto cifrado en otro texto cifrado que conduce a una transformación conocida del texto sin formato. Tenga en cuenta que el atacante no descifra el texto cifrado, sino que simplemente es capaz de manipular el texto sin formato de manera predecible. Esto se logra fácilmente en el RSA si el atacante

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

reemplaza el texto cifrado y por $s^e y$, donde s es un número entero y $y \equiv x^e mod\ n$. El receptor descifra el texto cifrado manipulado y obtiene el texto nativo x, multiplicado por un factor s, si calcula lo siguiente:

$$s^{ed} y \equiv s^{ed} x^{ed} \equiv sx\ mod\ n$$

Aunque el atacante no puede descifrar el texto cifrado, tales manipulaciones específicas todavía puede causar daño. Si x fuera una cantidad que debe transferirse, eligiendo $s = 2$, el atacante podría duplicar exactamente la cantidad de una manera que el receptor no la detecta.

Una posible solución a todos estos problemas es el uso de relleno, que incorpora una estructura aleatoria en el texto sin formato antes del cifrado y evita los problemas anteriores mencionados. Técnicas modernas como el cifrado asimétrico óptimo de relleno (Asymmetric Encryption Padding - OAEP) para rellenar mensajes RSA, son específicos y estandarizados para la criptografía de clave pública estándar # 1 (PKCS # 1)

Ataques matemáticos

Según (Paar and Pelzl, 2010), el mejor método criptoanalítico matemático que conocemos es factorizar el módulo. Si un atacante conoce el módulo n, la clave pública e y el texto cifrado y, y su objetivo es calcular la clave privada d, que tiene la propiedad que $e \times d = mod\ \emptyset(n)$; este simplemente podría aplicar el algoritmo euclidiano extendido y calcular d. Sin embargo, no conoce el valor de $\emptyset(n)$. En este punto viene la factorización, la mejor manera de obtener este valor es descomponer n en sus factores primos p y q. Si el atacante puede hacer esto, el ataque tiene éxito en tres pasos:

$$\emptyset(n) = (p-1)(q-1)$$

$$d - 1 \equiv e\ mod\ \emptyset(n)$$

$$x \equiv y^d\ mod\ n$$

Tal como expresa (Gómez, 2013), las claves RSA a menudo son generadas por un tercero confiable, se consideró una posibilidad que en los primeros tiempos de RSA se compartieran claves con el mismo módulo n, cada

una con un diferente exponente de cifrado como de descifrado. Sin embargo, esto lleva a una ruptura total del esquema RSA, no solo para un cifrado simple, sino también para otras versiones más potentes del mismo, esto permite que cualquier usuario de una de estas claves pueda recuperar la clave privada de cualquier otro usuario mediante el algoritmo probabilístico Las Vegas, y el algoritmo de recurrencia de Miller, ya que para las entradas (n, e, d), estos producen la factorización de n.

Según (Rubinstein, 2010), la forma más obvia de atacar a RSA es factorizar el módulo n. Si Eva puede aprender los primos p y q con $p \times q = n$, entonces ella puede calcular el exponente de descifrado d. Para hacer esto, supongamos que ella sabe n, p, q, y e. Recordemos que d es elegido tal que $e \times d \equiv 1 \; mod \; \emptyset(n)$, y que $\emptyset(n) = (p-1)(q-1)$. Como ella sabe p y q, ella puede calcular fácilmente $\emptyset(n)$. Luego usa el algoritmo euclidiano para encontrar d y a tal que $e \times d + a \times \emptyset(n) = 1$; solo d importa aquí, y ella puede ignorar a.

Tal como expresa (Lenstra et al., 2012) en su artículo demostró que es posible romper RSA con bastante frecuencia en la práctica al calcular muchos máximos común divisores. Para hacer esto, ellos recolectaron 4.7 millones de módulos RSA y calcularon los máximos común divisores de cada par de módulos. Siempre que el máximo común divisor de ese par era 1, no había nada en ese cálculo, pero cuando el máximo común divisor era mayor que 1, este será un factor de ambos módulos. Asumiendo que los módulos no son idénticos, el máximo común divisor debe ser un factor primo de ambos módulos; calcular el otro factor era entonces sencillo. Usando esta técnica, pudieron romper alrededor del 0.2% de las claves RSA.

Ataques de canal lateral

Estos ataques explotan la información sobre la clave privada que se filtra a través de canales físicos, como el consumo de energía y su comportamiento a través del tiempo. Para observar tales canales, el atacante generalmente debe tener acceso directo a la implementación del RSA, que puede ser mediante una celda telefónica o una tarjeta inteligente.

Según (Paar and Pelzl, 2010), estos ataques de canal lateral son un campo grande y activo de investigación en criptografía moderna, y se dan generalmente en la traza de la potencia de un microprocesador cuando ejecuta

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

el RSA, más precisamente, en el consumo de corriente eléctrica en el tiempo. El objetivo es extraer la clave privada d que se utiliza durante el descifrado del RSA. Este ataque específico se clasifica como análisis de potencia simple o SPA. Hay varias contramedidas disponibles para prevenir este ataque. Una simple es ejecutar un multiplicador con variables ficticias, después de una onda cuadrada que corresponde a un exponente bit 0. Esto da como resultado un perfil de potencia, y un tiempo de ejecución que es independiente del exponente.

Vulnerabilidad del sistema RSA

Según (Dong and Chen, 2012), el cifrado RSA es un algoritmo criptográfico ampliamente utilizado, y dado que no es poco común que el texto plano sea conocido parcialmente en la aplicación del RSA, ahora es ampliamente aceptado que el cifrado RSA debería evitar el uso de un exponente pequeño de cifrado. De otra manera, si existe $m < n^{\frac{1}{e}}$, se puede encontrar el mensaje m eficientemente extrayendo la raíz *e-ésima* en enteros del texto cifrado $c = m^e \, mod \, n$.

Tal como afirma (Koscielny et al. ,2013), la velocidad del sistema criptográfico RSA no es impresionante. En el caso de parámetros de 1024 bits, incluso cuando hay circuitos integrados especiales dedicados particularmente a las necesidades de cifrado RSA, el algoritmo sigue siendo más lento que DES por un factor de 100 a incluso 1000. La exponenciación modular, es decir, exponenciación de enteros módulo n, es la operación más costosa ya que para una base de k bits y un exponente de k bits la multiplicación $O(k)$ y su cuadratura deben ejecutarse.

Debido a su complejidad temporal, el algoritmo RSA rara vez se utiliza para el cifrado de datos, pero se puede aplicar para cifrar y transportar una clave de sesión para ser usado en la encriptación de datos por medio de algún algoritmo simétrico, como AES[7] o 3DES[8]. Por esta misma razón, el criptosistema RSA tiene aplicaciones limitadas en tarjetas con chip, ya que la mayoría de los microprocesadores no pueden realizar cálculos con claves largas de 1024 bits. En la práctica, generalmente se aplican pequeñas claves públicas que lo hacen más vulnerables, el descifrado y verificación de firmas es mucho más rápido en tales casos.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Tal como afirma (Johnsonbaugh, 2018), otra manera posible de interceptar y descifrar un mensaje sería tomar la raíz n de $c \ mod \ z$, donde c es el valor cifrado y a es el valor a transmitir. Como $c = a^n \ mod \ z$, la raíz n de $c \ mod \ z$ daría a, el valor descifrado, hasta ahora no hay implementaciones de un algoritmo para calcular las raíces n de $a^n \ mod \ z$.

La seguridad del sistema de cifrado RSA se basa principalmente en el hecho de que actualmente no existe un algoritmo eficiente conocido para factorizar enteros; es decir, actualmente no hay un algoritmo conocido para factorizar enteros de d bits en tiempo polinomial, $O(d^k)$.

Metodología computacional

El algoritmo de cifrado de clave pública RSA

El algoritmo RSA, empieza por seleccionar dos números primos, p y q, y calcular su producto n, que es el módulo para el cifrado y el descifrado. Luego necesitamos la cantidad $\emptyset(n)$, que se conoce como totalizador de Euler de n, que es el número de enteros positivos menores que n y primos relativos de n. Entonces, se selecciona un entero e que es primo relativo de $\emptyset(n)$, es decir, el máximo común divisor de e y $\emptyset(n)$, es 1. Finalmente, se calcula d tal que $d \times e \ mod \ \emptyset(n) = 1$. En la (Fig. 2), expresamos el procesamiento de bloques múltiples usando el algoritmo asimétrico RSA, su enfoque general y su ejecución.

Ejecución del algoritmo RSA

Generación de Claves

1. Seleccionamos $p = 37$ y $q = 71$, donde p y q deben ser ambos primos, $p \neq q$.
2. Calculamos $n = p \times q = 2627$.
3. Calculamos el totalizador de Euler $\emptyset(n) = (p - 1) \times (q - 1) = 36 \times 70 = 2520$.
4. Seleccionamos un entero e, de modo que $mcd(\emptyset(n), e) = 1$, entonces, $mcd(2520,17) = 1$, de donde se obtiene e = 17, donde $1 < e < \emptyset(n)$.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

5. Calculamos d, de manera que $d \times e \; mod \; \emptyset(n) \; = \; 1$, entonces, $d \times 17 \; mod \; 2520 \; = \; 1$, de donde se obtiene $d \; = \; 593$.
6. Clave pública $KU \; = \; \{e, n\} = \{17{,}2627\}$.
7. Clave privada $KR \; = \; \{d, n\} = \{593{,}2627\}$.

Generación de la encriptación

1. El texto nativo $M \; = \; m_1 m_2 m_3$ = 258566522, dividimos en 3 bloques de 3 dígitos cada uno:

   $m1 \; = \; 258$

   $m2 \; = \; 566$

   $m3 \; = \; 522$

   , donde $M < n$.
2. Texto cifrado por bloques: $C = c_1 c_2 c_3$, aplicado $c = m^e \; mod \; n$, se tiene:

$$c_1 \; = \; m_1^e \; mod \; n \; = \; 258^{17} mod \; 2627 \; = \; 813$$

$$c_2 \; = \; m_2^e \; mod \; n \; = \; 566^{17} mod \; 2627 \; = \; 1840$$

$$c_3 \; = \; m_3^e \; mod \; n \; = \; 522^{17} mod \; 2627 \; = \; 1619$$

Generación del desencriptado

1. Texto cifrado por bloques obtenido: $C \; = \; c_1 c_2 c_3$

$$c_1 \; = \; 813$$

$$c_2 \; = \; 1840$$

$$c_3 \; = \; 1619$$

2. Texto nativo por bloques: $M \; = \; m_1 m_2 m_3$, aplicando $m \; = \; c^d mod \; n$, se tiene:

$$m_1 \; = \; c_1^d \; mod \; n \; = \; 813^{593} \; mod \; 2627 \; = \; 258$$

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

$$m_2 = c_2^d \bmod n = 1840^{593} \bmod 2627 = 566$$
$$m_3 = c_3^d \bmod n = 1619^{593} \bmod 2627 = 522$$

donde el mensaje nativo es $M = m_1 m_2 m_3 = 258566522$.

El problema del módulo común en la seguridad del sistema RSA

Suponemos que dos entidades, Alicia y Bob usan claves públicas RSA con el mismo módulo n, pero con diferentes exponentes públicos e_a y e_b. El primer problema de seguridad que tenemos es que Alicia puede descifrar el mensaje enviado por Bob, esto sucede ya que dado un módulo n, un exponente público e_a, y la correspondiente clave privada d_a, es posible recuperar la factorización de $n = p \times q$. Entonces Alicia usa la factorización de n para recuperar $\emptyset(n) = (p-1) \times (q-1)$ y calcular la clave privada de Bob d_b con la ayuda de su exponente público e_b.

El segundo problema de seguridad es que, si existe un atacante activo como Eva, ella puede descifrar el mensaje enviado a Alicia y a Bob, siempre y cuando su $mcd(e_a, e_b) = 1$, esto sucede ya que si $mcd(e_a, e_b) = 1$, Eva puede calcular dos enteros x e y mediante la ecuación

$$e_a x + e_b y = 1$$

usando el algoritmo extendido de Euclides. Luego Eva usa los dos textos cifrados c_a y c_b, y calcula el texto plano m de la siguiente manera:

$$c_a^x \times c_b^y \equiv m^{e_a y} \times m^{e_a y} \equiv m^{e_a x + e_b y} \equiv m \bmod n$$

La vulnerabilidad de la encriptación RSA repetida

Este problema de vulnerabilidad consiste en que el criptoanalista realiza el cifrado repetido del texto cifrado, esto puede suceder incluso para un criptosistema de aspecto seguro, puesto que el texto sin formato se recupera, esto es un defecto fatal del esquema RSA. Sea un módulo RSA $n = p \times q = 35$, m un texto sin formato y c el correspondiente texto cifrado, se va comprobar que $E(c) = m^{e^2} mod\ n = m^{e^2} mod\ 35 = m$, para cualquier exponente público legítimo e, es decir, para cualquier e tal que $0 < e < \emptyset(35)$, y que su

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

$mcd(e, \emptyset(35)) = 1$, el cual muestra que este módulo conduce a un sistema criptográfico RSA completamente inseguro. Veamos como sucede esto, como $E(c) \equiv c^e \equiv (m^e)^e \; mod \; n = m^{e^2} mod \; n = m$, es suficiente demostrar que $e^2 \equiv 1 \; mod \; \emptyset(n)$, para todos los e legítimos, es decir, para todos los $e \in \mathbb{Z}^*_{\emptyset(35)}$ [9] .Tenemos que $\emptyset(35) = \emptyset(5) \times \emptyset(7) = 4 \times 6 = 24$, esta afirmación lo comprobamos con la tabla 4.

Esto se podría generalizar para montar el ataque cíclico al descifrar un texto cifrado c, con solo conocer la clave pública correspondiente $\{n, e\}$, esto sucede porque el cifrado RSA es una permutación de mensajes en el espacio $\{1,2,3, \ldots n-1\}$. Entonces existe un entero positivo k tal que $c^{e^k} \equiv c \; mod \; n$; en este caso también existe $c^{e^{k-1}} \equiv c \; mod \; n$, que nos lleva a encontrar el texto nativo m, esta particularidad lleva al ataque cíclico que se formula con el siguiente algoritmo 2, tabla 5.

Tabla 4 - Cuadrados de $\mathbb{Z}^*_{24}$.

Valor legítimo de e	$\boldsymbol{e^2 = mod\ 24}$
1	$1^2 \equiv 1 \equiv 1 \; mod \; 24$
5	$5^2 \equiv 25 \equiv 1 \; mod \; 24$
7	$7^2 \equiv 49 \equiv 1 \; mod \; 24$
11	$11^2 \equiv 121 \equiv 1 \; mod \; 24$
13	$13^2 \equiv 169 \equiv 1 \; mod \; 24$
17	$17^2 \equiv 289 \equiv 1 \; mod \; 24$
19	$19^2 \equiv 361 \equiv 1 \; mod \; 24$
23	$23^2 \equiv 529 \equiv 1 \; mod \; 24$

Baignères et al., 2006.

Tabla 5 - Ataque cíclico contra RSA.

Algoritmo 2: Ataque cíclico contra RSA
Entrada: Una clave pública RSA (n, e), un texto cifrado $c = m^e \; mod \; n$, y un valor N como límite superior para la cantidad de encriptaciones. **Salida:** La salida del texto plano m, en caso contrario falla.
1: $k \leftarrow 1$ 2: $s \leftarrow c$ 3: $for \; k = 1 \ldots N \; do$ 4: $\quad m \leftarrow s$ 5: $\quad s \leftarrow s^2 \; mod \; n$ 6: $\quad if \; s = c \; then$ 7: $\quad\quad salida \; de \; m \; y \; detener$

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

8: *end if*
9: *end for*
10: *Imprime y Falla y Termina*

Implementación del cifrado y descrifado RSA

Mostramos en la tabla 6, el código en Python de la generación de la clave pública y privada tomada de (Sweigart, 2013), observar que la clave pública y privada se encuentran en los ficheros *Clave_pubkey.txt* y *Clave_privkey.txt* respectivamente, y que el tamaño de la clave es de 1024 bits, además del cálculo de los valores de p, q, d y e.

Tabla 6 - Código para la generación de claves RSA.

Generación de Claves RSA - cifradoGenerarClavesRSA.py

```
import random, sys, os, rabinMiller, cryptomath
def main():
    print("Valores de RSA")
    makeKeyFiles("Clave", 1024)
    print("Ficheros creados.")
def generateKey(keySize):
    print("p primo")
    p = rabinMiller.generateLargePrime(keySize)
    print("p=", p)
    print("q primo")
    q = rabinMiller.generateLargePrime(keySize)
    print("q=",q)
    n = p * q
    print("n=pxq",n)
    print("e es primo relativo a (p-1)x(q-1)...")
    while True:
        e = random.randrange(2 ** (keySize - 1), 2 ** (keySize))
        if cryptomath.gcd (e, (p - 1) * (q - 1)) == 1:
            break
    print("e=", e)
    print("d que es modulus inverso de e...")
    d = cryptomath.findModInverse(e, (p - 1) * (q - 1))
    print("d=", d)
    publicKey = (n, e)
    privateKey = (n, d)
    print("Key public Key (n,e):", publicKey)
    print("Key private (n,d):", privateKey)
    return (publicKey, privateKey)
def makeKeyFiles (name, keySize):
    if os.path.exists("% s_pubkey.txt" % (name)) or os.path.exists("% s_privkey.txt" % (name)):
        sys.exit("Aviso: El archivo % s_pubkey.txt o % s_privkey.txt ya existe, use otro nombre o borre estos archivos y vuelva a ejecutar el
programa." % (name, name))
    publicKey, privateKey = generateKey(keySize)
```

```
    print()
    print("La llave public tiene e = % s y el modulus n = % s digitos respectivamente." % (len(str(publicKey[0])), len(str(publicKey[1]))))
    print("Escribiendo la llave public al fichero % s_pubkey.txt..." % (name))
    fo = open("% s_pubkey.txt" % (name), "w")
    fo.write("% s,% s,% s" % (keySize, publicKey[0], publicKey[1]))
    fo.close()
    print()
    print("La llave private tiene  d = % s  y el modulus n = % s digitos respectivamente." % (len(str(privateKey[0])), len(str(privateKey[1]))))
    print("Escribiendo la llave private en el fichero % s_privkey.txt..." % (name))
    fo = open("% s_privkey.txt" % (name), "w")
    fo.write("% s,% s,% s" % (keySize, privateKey[0], privateKey[1]))
    fo.close()
if __name__ == "__main__":
    main()
```

Ejecución del código cifradoGenerarClavesRSA.py
runfile('C:/Users/Codigo/cifradoGenerarClavesRSA.py', wdir='C:/Users/Codigo')
Reloaded modules: rabinMiller, cryptomath

Valores de RSA
p primo
p=
10844554708204014656513925882524768708299012709827377090474775998110236604037109420362995317179989823764073843188447130983373327192009628608693470294481720395762788099358510070408316023698609557033783573573843166943159991036959205046249228106452765137258494469949571724566830865206262990142409472780114333656 9
q primo
q=
12165691091124708046579542257651797228184620879814458247733928789144697076166194785113788177986566320470009315189356298145937223820381182907766142370559742691520718967893751502548155402160991503523350712430980005667080585606616638590336077757113555790602244514914666279572320330238072606939637142809199460055 3
n=pxq
13193150260081208895086999378245608590018838334115752454780895613030890303941637017158853400194188839782690388396310160801040442991648532503362481104607390974213202043917599521543912449722847815938370363851734273978284902684073082758853296993726398783026066504319537748897323993872982069126021240252109328182417018047709476579706912590504185447361841755635302223663423755708750301743959617257832276575494211450209247614705255217171812827347916112066432673009806445410997201167083130684186370569560594846820316760406355470411210125892584617088074144491981559315393670777558177693562526145316194323859579636279252265 7
e es primo relativo a (p-1) x(q-1) ...
e=
16100812069433029052610768630571151919152731669755571636786463619899692637558091574958576932016808439235757423592594227690882456511538344161145580459846336454138107803072602835434710277786958618299270015967116746356773166566458997703042158134936912771461307825081372994015951459386521152633974060657207072057
d que es modulus inverso de e...
d=
42948271411758709371405403060251235013229137727078235955285741652792679800191692025064969838576992696801787694813572325143422584531314879880414955052569746367206452950874911983839035728431265194903761214045656432454046744580054767837431726870027979010924332886141574210827937117061280792639937831707698479734668029843727157910531031418642127300452225250401169079084303720548521890068898411633801432461044739960077247434366142082826095936007097457392957898732097782024865478229023402297794099824153440494577874745162874292199843662447766534527642223490975888472085698943013485360168773039340699313388781715377637079 3
Key public Key (n, e):
(13193150260081208895086999378245608590018838334115752454780895613030890303941637017158853400194188839782690388839631016080104044299164853250336248110460739097421320204391759952154391244972284781593837036385173427397828490268407308275885329699372639878302606650431953774889732399387298206912602124025210932818241701804770947657970691259050418544736184175563530222366342375570875030174395961725783227657549421145020924761470525521717181282734791611206643 4
```
```

```
3267300980644541099720116708313068418637056956059484682031676040635547041121012589258461708807414449198155931539316707775581776935625261453161943238595796362792522657,161008120694330290526107686305711519191527316697555716367864636198996926375580915749585769320168084392357574235932594227690882456511538344161145580459846336454138107803072602835434710277869586182992700159671167463567731665664589977030421581349369127714613078250813729940159514593865211526339740606572070720 57)
Key private (n, d):
(131931502600812088950869993782456085900188383341157524547808956130308903039416370171588534001941888397826903888396310160801040442991648532503362481104607390974213202043917599521543912449722847815938370363851734273978284902684073082758853296993726398783026066504319537748897323993872982069126021240252109328182417018047709476579706912590504185447361841755635302223663423755708750301743959617257832276575494211450209247614705255217171812827347916112066432673009806445410997201167083130684186370569560594846820316760406355470411210125892584617088074144491981559315393167077755817769356252614531619432385957963627925226572,429482714117587093714054030602512350132291377270782359552857416527926798001916920250649698385769926968017876948135723251434225845313148798804149550525697463672064529508749119838390357284312651949037612140456564324540467445800547678374317268700279790109243328861415742108279371170612807926399378317076984797346680298437271579105130314186421273004522252504011690790843037205485218900688984116338014324610447399600772474343661420828260959360070974573929578987320977820248654782290234022977940998241534404945778747451628742921998436624477665345276422234909758884720856989430134853601687730393406993133887817153776327079 3)

La llave public tiene e = 617 y el modulus n = 309 digitos respectivamente.
Escribiendo la llave public al fichero Clave_pubkey.txt...

La llave private tiene d = 617 y el modulus n = 616 digitos respectivamente.
Escribiendo la llave private en el fichero Clave_privkey.txt...

Ficheros creados.
```

Mostramos en la tabla 7, el código de cifrado y descifrado del texto nativo para un tamaño de bloque de 128 bytes en el lenguaje Python, tomada de (Sweigart, 2013), observar que el texto cifrado se encuentra en el fichero *archivo_cifrado.txt.*

Tabla 7 - Código para el cifrado y descifrado RSA.

Cifrado y descifrado RSA - rsaCifradoDescifrado.py

```
import sys
DEFAULT_BLOCK_SIZE = 128
BYTE_SIZE = 256
def main():
    filename = "archivo_cifrado.txt"
    mode = "decrypt" # modo de encrypt o decrypt
    if mode == "encrypt":
        message ="Ser capaz de superar la seguridad no te convierte en un hacker"
        pubKeyFilename = "Clave_pubkey.txt"
        print("Cifrar y escribir a % s..." % (filename))
        encryptedText = encryptAndWriteToFile(filename, pubKeyFilename, message)
        print("Texto Cifrado:")
        print(encryptedText)
    elif mode == "decrypt":
        privKeyFilename = "Clave_privkey.txt"
        print("lectura de % s y descifrado..." % (filename))
        decryptedText = readFromFileAndDecrypt(filename, privKeyFilename)
        print("Texto descifrado:")
```

```
        print(decryptedText)
def getBlocksFromText(message, blockSize=DEFAULT_BLOCK_SIZE):
    messageBytes = message.encode("ascii")
    blockInts = []
    for blockStart in range(0, len(messageBytes), blockSize):
        blockInt = 0
        for i in range(blockStart, min(blockStart + blockSize, len(messageBytes))):
            blockInt += messageBytes[i] * (BYTE_SIZE ** (i % blockSize))
        blockInts.append(blockInt)
    return blockInts
def getTextFromBlocks(blockInts, messageLength, blockSize=DEFAULT_BLOCK_SIZE):
    message = []
    for blockInt in blockInts:
        blockMessage = []
        for i in range(blockSize - 1, -1, -1):
            if len(message) + i < messageLength:
                asciiNumber = blockInt // (BYTE_SIZE ** i)
                blockInt = blockInt % (BYTE_SIZE ** i)
                blockMessage.insert(0, chr(asciiNumber))
        message.extend(blockMessage)
    return "".join(message)
def encryptMessage(message, key, blockSize=DEFAULT_BLOCK_SIZE):
    encryptedBlocks = []
    n, e = key
    for block in getBlocksFromText(message, blockSize):
        encryptedBlocks.append(pow(block, e, n))
    return encryptedBlocks
def decryptMessage(encryptedBlocks, messageLength, key, blockSize=DEFAULT_BLOCK_SIZE):
    decryptedBlocks = []
    n, d = key
    for block in encryptedBlocks:
        decryptedBlocks.append(pow(block, d, n))
    return getTextFromBlocks(decryptedBlocks, messageLength, blockSize)
def readKeyFile(keyFilename):
    fo = open(keyFilename)
    content = fo.read()
    fo.close()
    keySize, n, EorD = content.split(",")
    return (int(keySize), int(n), int(EorD))
def encryptAndWriteToFile(messageFilename, keyFilename, message, blockSize=DEFAULT_BLOCK_SIZE):
    keySize, n, e = readKeyFile(keyFilename)
    if keySize < blockSize * 8:
        sys.exit("ERROR: el size del bloque es % s y el size de la clave es % s bits. El cifrado RSA requiere que el size del bloque sea igual o
mayor que el size de la clave. Disminuya el size del bloque o use diferentes teclas." % (blockSize * 8, keySize))
    encryptedBlocks = encryptMessage(message, (n, e), blockSize)
    for i in range(len(encryptedBlocks)):
        encryptedBlocks[i] = str(encryptedBlocks[i])
    encryptedContent = ",".join(encryptedBlocks)
    encryptedContent = "% s_% s_% s" % (len(message), blockSize, encryptedContent)
    fo = open(messageFilename, "w")
    fo.write(encryptedContent)
    fo.close()
    return encryptedContent
def readFromFileAndDecrypt(messageFilename, keyFilename):
    keySize, n, d = readKeyFile(keyFilename)
```

```
    fo = open(messageFilename)
    content = fo.read()
    messageLength, blockSize, encryptedMessage = content.split("_")
    messageLength = int(messageLength)
    blockSize = int(blockSize)
    if keySize < blockSize * 8:
        sys.exit("ERROR: el size del bloque es % s bits y el size de la clave es % s bits. El cifrado RSA requiere que el size del bloque sea igual
o mayor que el size de la clave" % (blockSize * 8, keySize))
    encryptedBlocks = []
    for block in encryptedMessage.split(","):
        encryptedBlocks.append(int(block))
    return decryptMessage(encryptedBlocks, messageLength, (n, d), blockSize)
if __name__ == "__main__":
    main()
```

Ejecución del código *rsaCifradoDescifrado.py*, y para el descifrado modificamos el código de la tabla 7 por *mode="decrypt"* según corresponda, la ejecución del cifrado y descifrado se visualiza en la tabla 8.

Tabla 8 - Ejecución RSA para el cifrado y descifrado.

Ejecución del código *rsaCifradoDescifrado.py*
runfile('C:/Users/Codigo/rsaCifradoDescifrado.py', wdir='C:/Users/Codigo') Cifrar y escribir a archivo_cifrado.txt... **Texto Cifrado:** 62_128_753866072064530531113776394180254090719722984300609127782146875484948583376054844642889910559562425963538586992098563150648392238682115191326642934047596210681580234932155399373813949082282457995872234374844381839437371227113137451795234501458863851848177503497956448888362723925697194051373281915974156016614147845345315680094604249724447396576099456200015805725256025909143544779882927717838395373965806593130919327717133835123425985464753790721231017773546560805852755666221788383788512136276921676732862093447956439955211944283700167490899848787635111521607494587496143909919485824179230909382962026738566499994 lectura de archivo_cifrado.txt y descifrado... **Texto descifrado**: Ser capaz de superar la seguridad no te convierte en un hacker

Resultados y discusión

De hecho, el mayor progreso que se ha hecho en la factorización en los últimos tiempos probablemente no habría sucedido si no fuera por el algoritmo RSA. A pesar de que la factorización se ha vuelto más fácil de lo que los diseñadores de RSA habían asumido, factorizar módulos de RSA más allá de cierto tamaño todavía está fuera del alcance. La longitud exacta que debe tener el módulo RSA es un tema de mucha discusión,

muchas aplicaciones RSA usan una longitud de 1024 bits por defecto. Hoy se cree que podría ser posible factorizar números de 1024 bits dentro de un periodo de unos 10 a 15 años o tal vez menos, y más aún las organizaciones de inteligencia podrían ser capaces de hacer esto posible incluso antes. Por lo tanto, se recomienda elegir los parámetros RSA en el rango de 2048 a 4096 bits para seguridad a largo plazo, si el valor del exponente es alto, entonces la seguridad del algoritmo RSA también es alta (Saranya et al., 2014).

Los Laboratorios RSA actualmente recomienda tamaños de clave de 1024 bits para uso corporativo y 2048 bits para claves extremadamente valiosas como el par de claves raíz utilizado por una autoridad de certificación, además varios estándares recientes especifican un mínimo de 1024 bits para uso corporativo. Es posible también que se cifre información menos valiosa utilizando una clave de 768 bits, ya que dicha clave sigue siendo segura y está fuera del alcance de todos los algoritmos de ruptura conocidos (Dooley, 2018).

La probabilidad de elegir aleatoriamente el mismo p o q repetidamente es muy improbable, en la programación de computadoras, la buena aleatoriedad proviene de algoritmos conocidos como generadores deterministas de bits aleatorios (DRBG). Los DRBG requieren datos de entrada denominados semillas de entropía, que provienen de una fuente (o fuentes) que generan eventos que no presentan ningún signo de similitud o previsibilidad de patrones. Si la fuente cumple con estos criterios simples, se sabe que es una buena fuente de entropía. Los algoritmos DRGB son revisados a fondo por investigadores de seguridad, matemáticos, universidades, así como por el Instituto Nacional de Estándares y Tecnología (NIST). Una buena fuente de entropía es aquella que pasará la prueba de evaluación de entropía del NIST, el cual creo un conjunto de pruebas estadísticas que ejecutará en su fuente de entropía. El código fuente de esta herramienta está disponible en https://github.com/usnistgov/SP800-90B_EntropyAssessment. Si una fuente pasa la prueba, esta se considera una buena para la generación segura de aleatoriedad, el cual se puede utilizar para generar claves de cifrado, o seleccionar aleatoriamente los valores de p y q (Labs, 2020).

Los sistemas de cifrado que implementan criptografía de clave pública tienen aproximadamente la misma seguridad como el de los sistemas de claves simétricas con longitudes equivalentes de clave de aproximadamente un tercio de longitud de la clave RSA. Los sistemas de cifrado de clave pública son sistemas

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

lentos, en algunos casos muy lentos, mientras que los sistemas simétricos como DES y AES usan operaciones informáticas muy simples como desplazamiento de bits y/o operaciones lógicas como el or exclusivo. Hasta la fecha todos los sistemas de clave pública desarrollados requieren operaciones matemáticas muy complicadas para cifrado, estas funciones matemáticas requieren mucho más tiempo de procesamiento de la CPU, que las simples operaciones requeridas para sistemas simétricos, es por eso que los sistemas de clave pública tienen limitaciones principalmente para el papel para el que fueron concebidos por primera vez, que es el de resolver el intercambio de claves.

Conclusiones

La seguridad del RSA no lo sabemos, tal vez hay un algoritmo de tiempo polinómico que pueda factorizar números o resolver el problema del logaritmo discreto, no conocemos ninguna forma de romper RSA que no sea factorizando el módulo, pero una vez más, nadie sabe si existen otros métodos que podrían romper el RSA que no sean equivalentes a la factorización.

Para evitar el ataque de la factorización, el módulo debe ser lo suficientemente grande, es decir se debe trabajar con módulos de 1024 bits o más, además de tener en cuenta que cuando se usa RSA, debemos rellenar primero el mensaje para evitar el ataque basado en la maleabilidad del RSA, y para evitar o bloquear la deducción práctica de la clave privada d mediante el ataque cíclico, debemos de utilizar primos seguros en el diseño de las claves.

La seguridad del algoritmo RSA radica en dos áreas, primero, si bien es fácil calcular $n = p \times q$, es muy difícil hacer lo contrario; es decir obtener p y q, dado n, y segundo, es extremadamente costoso computacionalmente encontrar los factores primos de un número compuesto grande, el cual es el eje de seguridad del RSA. Los dos números primos p y q deben ser números primos muy grandes, suficientemente grandes para que sus representaciones binarias de cada uno sean alrededor de 500 bits o más. Esto conducirá

a un producto binario de alrededor 1000 bits, a medida que las computadoras se vuelven cada vez más rápidas, se necesitará que las cantidades de bits para $n = p \times q$ también tengan que crecer.

Sobre los productos comerciales de seguridad del RSA, nosotros debemos verificar qué fuentes de entropía utilizan, siempre es una buena pregunta que debemos hacernos. Los productos del RSA Security usan una variedad de módulos criptográficos, que incluyen, entre otros a, RSA BSAFE ® Crypto-C Micro Edition (Crypto-C ME)[10] que tiene su fuente de entropía probada contra las pruebas de NIST, y RSA BSAFE ® Crypto-J (Crypto-J)[11], que se basa en la máquina virtual de Java en la que se ejecuta la entropía. RSA BSAFE ®, es una biblioteca de criptografía validada por el FIPS 140-2[12], disponible en C y Java, ofrecida por RSA Security.

Referencias

T. BAIGNÈRES, P. JUNOD, Y. LU, J. MONNERAT, and S. VAUDENAY. A Classical Introduction To Cryptography Exercise Book. New York: Springer, 2006. p.181-186. ISBN 978-0-387-27934-3.

N.L. BIGGS. Codes: An Introduction to Information Communication and Cryptography. Verlag London: Springer, 2008. p.207-220. ISBN 978-1-84800-272-2. doi: 10.1007/978-1-84800-273-9.

N. J. DARAS and M. T. RASSIAS. Computation, Cryptography, and Network Security. New York: Springer International Publishing, 2015. p.1-9. ISBN 978-3-319-18274-2.

N. DASWANI, C. KERN, and A. KESAVAN. Foundations of Security What Every Programmer Needs to Know. New York: Apress, 2007. p.221-238. ISBN 978-1-59059-784-2.

H. DELFS and H. L. KNEB. Introduction to Crytography Principles and Applications. New York: Springer International Publishing, 2nd edition, 2007. p.33-80. ISBN 978-3-540-49243-6.

L. DONG and K. CHEN. Cryptographic Protocol, Security Analysis Based on Trusted Freshness. New York: Springer, 2012. p.41-78. ISBN 978-3-642-24072-0.

J.F. DOOLEY. History of Cryptography and Cryptanalysis, Codes, Ciphers, and Their Algorithms. New York: Springer, 2018. p.185-202. ISBN 978-3-319-90442-9.

S.S. EPP. Discrete Mathematics with Applications. EEUU: Cengage Learning, 5th edition, 2019. p.145-226. ISBN 978-1337694193.

J.L. GÓMEZ. Introduction to Cryptography with Maple. Verlag Berlin Heidelberg: Springer, 2013. p.419-461. ISBN 978-3-642-32165-8. doi: 10.1007/978-3-642-32166-5.

10. J. HOFFSTEIN, J. PIPHER, and J. H. SILVERMAN. An Introduction to Mathematical Cryptography. New York: Springer International Publishing, 2008. p.113-187. ISBN 978-0-387-77993-5.

R. JOHNSONBAUGH. Discretre Mathematics. Chicago: Pearson Education, eigtht edition, 2018. p.214-254. ISBN 978-0-321-96468-7.

M. JOYE and M. TUNSTALL. Fault Analysis in Cryptography. New York: Springer International Publishing, 2012. p.1-15. ISBN 978-3-642-29655-0.

N. KOBLITZ. A Course in Number Theory and Cryptography. New York: Springer, second edition, 1994. p.83-124. ISBN 978-1-4612-6442-2. doi: 10.1007/978-1-4419-8592-7.

N. KOBLITZ. Algebraic Aspects of Cryptography. Verlag Berlin: Springer, 3th edition, 2004. p.18-52. ISBN 978-3-642-08332-7. doi: 10.1007/978-3-662-03642-6.

C. KOSCIELNY, M. KURKOWSKI, and M. SREBRNY. Modern Cryptography Primer Theoretical Foundations and Practical Applications. New York: Springer, 2013. p.36-76. ISBN 978-3-642-41385-8.

E. KRANAKIS. Primality and Cryptography. New York: Wiley-Teubner Series in Computer Science, 1986. p.138-153. ISBN 78-3-322-96648-3. doi: 10.1007/978-3-322-96647-6.

J.F. KUROSE and K. W. ROSS. Computer Networking A Top-Down Approach. New Jersey: Addison-Wesley, 6th edition, 2013. p.671-754 ISBN 978-0-13-285620-1.

RSA LABS. The rsa homonym. RSA Point of View, Blog Post. [En línea] February 2020, [Consultado el: 10 Julio de 2020] Disponible en: https://www.rsa.com/en-us/blog/2020-02/the-rsa-homonym.

A. LENSTRA, J. P. HUGHES, M. AUGIER, J.W. BOS, T. KLEINJUNG, and C. WACHTER. Ron was wrong, whit is right. Technical report, IACR, 2012. URL https://eprint.iacr.org/2012/064

S. NISHA AND M. FARIK. RSA public key cryptography algorithm a review. International Journal Of Scientific Technology Research, 6(7):187-189, July 2017. ISSN 2277-8616.

C. PAAR AND J. PELZL. Understanding Cryptography, A Textbook for Students and Practitioners. Berlin: Springer, 2010. p.173-204. ISBN 978-3-642-04100-6.

G. J. PACE. Mathematics of Discrete Structures for Computer Science. New York: Springer, 2012. p.211-256. ISBN 978-3-642-29839-4.

R. PARTHAJIT. On the abundance of large primes with small b-smooth values for p-1: An aspect of integer factorization. International Journal on Computer Science and Engineering (IJCSE), 10(1):8, Jan 2018.

R. L. RIVEST, A. SHAMIR, and L. ADLEMAN. A method for obtaining digital signatures and public-key cryptosystem. Communications of the ACM, 21(2):120-126, February 1978.

S. S. RUBINSTEIN. Cryptography. Palo Alto, CA, USA: Springer, 2010. p.113-127. ISBN 978-3-319-94817-1.

A. SALOMAA. Public Key Cryptography. Berlin, Heidelberg: Springer, second edition, 1996. p.125-157. ISBN 978-3-642-08254-27. URL https://doi.org/10.1007/978-3-662-03269-5.

SARANYA, VINOTHINI, and VASUMATHI. A study on rsa algorithm for cryptography. (IJCSIT) International Journal of Computer Science and Information Technologies, 5(4):5708-5709, 2014. ISSN 0975-9646.

N. P. SMART. Cryptography Made Simple: Information Security and Cryptography. Switzerland: Springer International Publishing, 2016. p.27-50. ISBN 978-3-319-21935-6.

W. STALLINGS. Cryptography and Network Security: Principles and Practice. EEUU: Prentice Hall, sixth edition, 2014. p.253-285. ISBN 978-0133354690.

A. SWEIGART. Hacking Secret Ciphers with Python. United States: Released under Creative Commons BY-NCSA, 2013. p.378-420. ISBN 978-1482614374.

H. C. A TILBORG and S. JAJODIA. Encyclopedia of Cryptography and Security. New York: Springer, second edition, 2011. p.795-797. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5.

S. VAUDENAY. A Classical Introduction To Cryptography, Applications for Communications Security. New York: Springer, 2006. p.229-250. ISBN 978-0-387-25464-7.

Conflicto de interés

No existe conflicto de interés de este trabajo con ninguna organización académica y/o comercial.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 98-130

Contribuciones de los autores

El trabajo se realizó en todo su contexto por mí, **Hubner Janampa Patilla** al 100% no habiendo participación de ningún otro investigador y/o organismo.

Financiación

No se obtuvo financiamiento por parte de ninguna institución académica y/o comercial para realizar este trabajo de investigación.

[1] Secure Sockets Layer (SSL): protocolo criptográfico que proporciona comunicación segura en la red.

[2] Cellular Digital Packet Data (CDPD): es una tecnología de transmisión de datos en terminales TDMA.

[3] Pretty Good Privacy (PGP): protocolo que cifra el contenido y accede a él mediante una clave pública.

[4]Sea G un grupo cíclico de orden t, $|\langle g\rangle| = |G| = t, Y \in G$, un elemento del grupo G. Una raíz *k-ésimo* del logaritmo discreto de Y en la base g, es un entero que satisfaga x: $0 \leq x < t$, y satisfaga $Y = g^{x^k}$, para un x si existe.

[5]Si $k = 50$, y consideramos un primo de 512 bits de longitud, la factorización tomará $O(2^{\frac{512}{50}}) = O(2^{10})$ que no es seguro para la velocidad actual de las computadoras. Por otro lado, si $k = 2$, usando los mismos cálculos, obtenemos, $O(2^{\frac{512}{2}}) = O(2^{256})$ que sí es seguro en el contexto actual.

[6]En criptografía, PKCS # 1 es el primero de una familia de estándares llamados estándares de criptografía de clave pública (Public Key Cryptography Standards - PKCS), publicados por RSA Laboratories. Proporciona las definiciones básicas y las recomendaciones para implementar el algoritmo RSA para la criptografía de clave pública.

[7] Advanced Encryption Standard (AES), es un esquema de cifrado por bloques.

[8]Triple DES se le llama al algoritmo que hace triple cifrado del DES.

[9] $\mathbb{Z}^*_{\emptyset(n)} = \{[a] | 1 \leq a \leq \emptyset(n) - 1\, , mcd(a, \emptyset(n) = 1\}$

[10] RSA BSAFE ® Crypto-C ME, es un kit de herramientas de desarrollo de software (SDK) para crear tecnologías de seguridad criptográfica en aplicaciones, dispositivos y sistemas C y C++.

[11] RSA BSAFE ® Crypto-J, es un módulo criptográfico escrito en Java y validado por FIPS.

[12] FIPS 140-2 es el acrónimo de Federal Information Processing Standard, es un estándar de seguridad de ordenadores del gobierno de los Estados Unidos para la acreditación de módulos criptográficos.