TRABAJO DE INVESTIGACION E IMPLEMENTACIÓN DE ALGORITMOS QUE PERMITAN GENERAR NÚMEROS ALEATORIOS GRANDES Y ALGORITMOS QUE PRUEBEN LA PRIMALIDAD DE UN NÚMERO GRANDE

Fecha de presentación: 18 de Junio 2021

Instrucciones: Deberán subir al "Aula Virtual" un trabajo por grupo (responsable) y deben colocar el link al github en la parte de comentarios.

Exposición. Duración: máximo 15 minutos. Como referencia, ver el formato de la presentación de las diapositivas en Anexo 1)

---

1. Introducción

Criptografía es un elemento clave para preservar la confidencialidad de las comunicaciones. Ella es usada para codificar los datos, de tal forma que, no pueda ser fácilmente leída por nadie excepto por el receptor.

Hay muchos protocolos que usan criptografía. SSH y SSL son algunos de los más conocidos. PGP es una aplicación que utiliza criptografía de clave pública. Estos protocolos y sus respectivos algoritmos tienen éxito mientras más grandes sean sus claves, que generalmente se obtienen a partir de la generación de números primos grandes[1]. Además, sus cálculos implican la obtención de la inversa, multiplicación, factorización, etc., de números grandes.

Para un número grande, mayor a mil dígitos, no se conoce un algoritmo de factorización eficiente. Un intento demoró casi tres años para factorizar un número de 232 dígitos (RSA-768 http://en.wikipedia.org/wiki/RSA_numbers#RSA-768). No todos los números de una determinada longitud son igualmente difíciles de factorizar (https://en.wikipedia.org/wiki/RSA_Factoring_Challenge). Según algunos autores, los casos más difíciles son aquellos en que los factores son dos números primos escogidos al azar y del mismo tamaño. Esta dificultad de factorizar dos números grandes es la fortaleza del algoritmo de criptografía RSA.

En esta tarea de programación, se materializará los principales pilares fundamentales para implementar el algoritmo RSA. La generación de números aleatorios y primos.

2. Detalles

La generación de números aleatorios es empleada en diversas áreas de ciencia de la computación. Tales como criptografía, simulación, juegos, etc. El objetivo de este trabajo es analizar los diferentes generadores de números aleatorios y optar por el más seguro cuando se trata de su uso en criptografía.

Un algoritmo de generación de número aleatorios debe de tener las siguientes propiedades:

a. Periodicidad (repetitibilidad)
b. Predicibilidad

c. Irreproducibles
d. Independencia de valores
e. Complejidad
f. Propiedades estadísticas
g. Número de bits
h. Semilla

Puesto que el algoritmo RSA utiliza grandes exponentes, los tipos de enteros estándares (int -16 ó 32 bits, long int – 32 bits e incluso long long int - 64 bits) no son suficientes.

Entonces, puedes usar librerías como NTL (http://shoup.net/ntl/), librería para Teoría de Números, GMP (http://directory.fsf.org/wiki/GNU) –GNU Multiple Precision Arithmetic Library, entre otras. (***Nota: utilizar estas librerías solo para que sus funciones utilicen números grandes, no para otros fines***)

Además, el RSA requiere encontrar números primos muy grandes aleatorios rápidamente. Por lo que también, deben investigar e implementar métodos de test de primalidad para números grandes. Hay una gran cantidad de métodos cómo: La criba de Eratóstenes, Test de primalidad de Fermat, Test de Solovay-Strassen, Test de Miller-Rabin, Primality testing using the factorization n-1, Algoritmos AKS, etc. "Public-Key Parameters"

Para la generación de números aleatorios grandes, deben obtener la semilla usando el hardware como:

- The surrounding light and scene
- The surrounding sound or noise
- The TCP information for the network
- The RTT of a specific network, dependant on the congestion of the network
- System time
- Scheduling delay over core or multicores
- Entre otros

3. Informe

Pueden unir, en el mismo artículo la generación de números aleatorios y test de primalidad. O si desean, los puede separar. No se olviden de las referencias. En cada algoritmo de estudio propuesto debe explicar: en qué consiste, el seudo-algoritmo, seguimiento, implementación en c++.

a. Implemente cada uno de los algoritmos. Muestre los resultados paso a paso.
b. Analice cada uno de los algoritmos y determine las ventajas y desventajas de cada uno de ellos, ¿determine cuál de todos es el más eficiente?, Es decir quién converge más rápido. Cuál demora menos tiempo. Quién soporta trabajar con la mayor cantidad de bits.
c. Comparen los algoritmos en función del número de bits (tamaño del número) con respecto al tiempo de procesamiento, convergencia de la respuesta (número de vueltas en el loop, si las hubiese) y ocupación en memoria.

4. Implementación

Para los algoritmos que considere los más prometedores en la generación de aleatorios y primos. Realice lo siguiente.

Cree una función que se llame RANDBIGINTEGER y que pueda ejecutarse desde la línea de comandos o main(). Tiene un argumento de entrada, número de bits del número aleatorio (tamaño). Imprimir el número aleatorio.

Ejemplo:

- ZZ RANDBIGINTEGER(int Nro. Bits) y

  RANDBIGINTEGER tendrá un solo argumento, un entero positivo que representa el número de bits. La función debe generar un número de acuerdo a la cantidad de bits. No pueden usar las funciones predefinidas ya sea por el lenguaje de programación o por las librerías para soportar número grandes.
  Ejemplo:
  randomgen(1024)

Cree una función para el test de primalidad llamada PRIMECHECK que recibirá un número positivo grande y retornará verdadero o falso. Ejemplo:

- Bool PRIMECHECK (ZZ Aleatorio).
  Ejemplo:
  PRIMECHECK (32401) True
  PRIMECHECK (3244568) False

Luego realice lo siguiente

Instrucción 1:

RANDBIGINTEGER(1024)

142405175064861448442669284843420489603593930617313976674095914073492903976984848373315014340583589674334422581561784146805278310143147937016874549483037286357105260324082207009125626858996989027805604841776344359158053673248019204338406280932000275573354237039522117150476778214733739382939035838341675795443

Instruccion 2

PRIMECHECK (142405175064861448442669284843420489603593930617313976674095914073492903976984848373315014340583589674334422581561784146805278310143147937016874549483037286357105260324082207009125626858996989027805604841776344359158053673248019204338406280932000275573354237039522117150476778214733739382939035838341675795443)

True

Realice este proceso (Instrucción 1 e Instrucción 2) para 20 números aleatorios generados en un lapso de tiempo 5 minutos, (deben capturar en la pantalla el tiempo). El objetivo es determinar cuán aleatoria es su función de generación de primos.

Deberá probar el mismo proceso para números de 16 bits y para 1024.

Proporcionar resultados engañosos, falta de algún requisito o copia de trabajo dará lugar a una calificación de cero para toda la asignación

5. Implementación en el RSA

Una vez, estudiados los diferentes algoritmos y después de su análisis, deberá elegir las funciones que consideren las más adecuadas para la implementación robusta del RSA.
En el RSA deben de usar esas funciones para la generación de claves', cifrado y descrifrado.

En la generación de claves deben obtenerse, a partir de dos números primos p y q, las claves pública y privada.

En la siguiente tabla deben testar algunos casos para probar la función generar claves. Deben asegurar en seleccionar tres números (p,q,e) que tengan como mínimo 10 dígitos de longitud (para el caso del documento, en clase se evaluará para número de 1024 bits).

p	q	n	e	d
1019	1021	1040399	7	890023
1093	1097	1199021	5	478733
433	499	216067	5	172109
1061	1063			

Para el cifrado deberían usar la clave pública (n,e) y por lo menos 2 bloques en "c" para codificar y retornar el texto cifrado en "m" (separe los bloques en 'c' y'm' – para efectos de calificación).

n	e	c	m
1040399	7	99	579196
1199021	5	70	871579
216067	5	89	23901
1127843	7	98	
1461617	7	113	
105481	5	105	
193997	5	85	

Para el descifrado, usen la clave privada (n,d) el texto cifrado "m" y obtengan el mensaje original

Referencias

Generación de números aleatorios

- Handbook of Applied Cryptography, Menezes, Oorschot, Vanstone. CRC Press, New York, fifth edition (2001). http://www.cacr.math.uwaterloo.ca/hac/
  Capítulo 4: Public key parameters. Random search for probable primes. Página 145. http://www.cacr.math.uwaterloo.ca/hac/about/chap4.pdf
  Capítulo 5: Pseudorandom Bits and Sequences
  http://cacr.uwaterloo.ca/hac/about/chap5.pdf
- Sanguinetti, B., Martin, A., Zbinden, H., & Gisin, N. (2014). Quantum random number generation on a mobile phone. *Physical Review X*, *4*(3), 031056.
  https://www.researchgate.net/publication/262050051_Quantum_Random_Number_Generation_on_a_Mobile_Phone
- MARTON, Kinga; SUCIU, Alin; IGNAT, Iosif. (2010) Randomness in digital cryptography: A survey. *Romanian Journal of Information Science and Technology*, , vol. 13, no 3, p. 219-240.
  http://romjist.ro/content/pdf/kmarton.pdf

- STIPČEVIĆ, Mario; KOÇ, Çetin Kaya. True random number generators. En *Open Problems in Mathematics and Computational Science*. Springer, Cham, 2014. p. 275-315.
  http://cetinkayakoc.net/docs/b08.pdf

Generación de Primos

- Breve Reseña sobre la Hipótesis de Riemann, Primalidad y el Algoritmo AKS
  http://www.criptored.upm.es/guiateoria/gt_m117j.htm

- La criba refinada de Eratóstenes . Scientia et Technica Año XII, No 31, Agosto de 2006 UTP. ISSN 0122-1701

- Probabilidad, Números Primos y Factorización de Enteros. Implementaciones en Java y VBA para Excel. Revista digital Matemática, Educación e Internet
  https://tecdigital.tec.ac.cr/revistamatematica/HERRAmInternet/v8n2-DIC-007/Probabilidad_Primos_Factorizacion.pdf

- Los enigmáticos números primos
  https://www.yumpu.com/es/document/read/14281137/los-enigmaticos-numeros-primos-cinvestav

ANEXO 1

Contenido de Diapositivas

- **Introducción**
  Enumerar todos los algoritmos de análisis (tanto para la generación de primos como para el test de primalidad)
- **Explicar cómo generó la semilla**
  Explícitamente como uso el hardware de su computador
- **El mejor Algoritmo para Generación de aleatorios**
  . Explicar el algoritmo e Implementación
  . Mostrar la comparación con el resto de algoritmos que justifique que sea el mejor.
- **El mejor Algoritmo para Generación de primos y/o test de primalidad**
  . Explicar el algoritmo e Implementación
  . Mostrar la comparación con el resto de algoritmos que justifique que sea el mejor.
- **Implementación**
  Muestre los resultados que obtuvo en la parte 4 del informe (Implementación). Tanto para números de 16 y 1024 bits ()
  RANDBIGINTEGER(16) - PRIMECHECK (Nro Aleatorio) => True/False
  RANDBIGINTEGER(1024) - PRIMECHECK (Nro Aleatorio) => True/False

- **Conclusiones**
- **Referencias**