Generación de aleatoriedad para aplicaciones criptográficas

Trabajo Terminal No. _ _ _ _-_ _ _

*Alumnos: *Hernández Morales Brenda Mariana*
Directores: Díaz Santiago Sandra, Mancillas López Cuauhtemoc
**e-mail: bhernandezm1701@alumno.ipn.mx*

Resumen – La generación de números aleatorios para aplicaciones criptográficas, requiere de la búsqueda de mecanismos eficaces tanto en hardware como en software, pues los ya existentes emplean demasiados recursos y tiempo de tal forma que no son capaces de satisfacer la demanda de estos valores por las aplicaciones criptográficas modernas. Por ello, es de suma importancia buscar optimizaciones de los algoritmos diseñados para tal propósito en cuanto a calidad y eficiencia. Así, el objetivo del presente trabajo es analizar, implementar y comparar aquellos cuya salida sean las sucesiones de números aleatorios que satisfagan los estándares definidos por el NIST.

Palabras clave – Criptografía, pseudoaleatoriedad, entropía, hardware reconfigurable.

1. Introducción

Uno de los aspectos importantes para la seguridad de los sistemas criptográficos es la generación impredecible y rápida de bits aleatorios. La aleatoriedad en bits es referida como la obtención de un resultado 0 ó 1 con la misma probabilidad de aparición de cualquiera de los dos, tomados como eventos independientes a lo largo de la secuencia de bits de salida del sistema que la genere. Esta salida obtenida debe ser -idealmente- totalmente impredecible o bien casi imposible de determinar [1]. Ésta puede darse a través de dos tipos de mecanismos para su generación: Aleatorio Determinístico (DRBG) y Aleatorio No Determinístico (NRBG). Algunas de sus aplicaciones específicas se encuentran en la generación de llaves, vectores de inicialización en los diferentes modos de operación de los cifradores de bloque o la generación de números primos al azar en el caso del sistema RSA.

La generación puramente aleatoria de estos bits se hace mediante un *True Random Number Generator (TRNG)* [2] cuya aleatoriedad es posible obtenerla a través de una fuente no determinística, es decir, que para una misma entrada se pueda obtener una salida diferente en cada evento. Para ello hacemos uso de un recurso de tipo físico en el entorno en el que se implemente este proceso, conocido como como fuente de entropía, sea empleando un mecanismo directamente natural o bien uno construido. La entropía está definida como el reflejo de la incertidumbre asociada a la predicción del resultado de un experimento [3] y en el caso de los generadores de este tipo, se requiere tener una entropía ideal, es decir, que cada uno de los bits de su cadena de salida sea totalmente impredecible.

Las fuentes físicas de entropía no naturales, que puede ser con el empleo de hardware dedicado para tal propósito o bien mediante la recolección de datos del propio sistema donde se ejecute este generador, producen secuencias de bits que no siempre cumplen con la suficiente entropía para poder ser consideradas aleatorias, por lo que el uso de una fuente física es imprescindible para obtener secuencias con aleatoriedad real.

Sin embargo, generar números realmente aleatorios es un mecanismo que resulta ineficiente debido al tiempo y recursos que emplea, ya que las solicitudes a estos mecanismos son de alta demanda en tiempo y cantidad, y generar valores de este tipo implica tener la suficiente entropía para que cada una de estas peticiones genere un valor realmente aleatorio, lo cual es difícil de lograr en un tiempo reducido. Por ello, se han desarrollado mecanismos pseudoaleatorios - también llamados no determinísticos - que funcionan con base en la generación de una semilla obtenida a través de un mecanismo aleatorio [4] y son conocidos como *Pseudo-random Number Generators (PRNGs)*. A partir de este valor inicial obtenido, se deriva un proceso donde cualquiera de las salidas del sistema puede reproducirse a través de su semilla original, la cual es almacenada durante todo el procedimiento y debe permanecer siempre en secreto. Ya que el mecanismo pseudoaleatorio se deriva de uno aleatorio ambos cuentan con una fuente de entropía, sin embargo, este proceso determinístico no se ve obligado a tener una medida de entropía ideal.

La generación de aleatoriedad en bits está regida por estándares establecidos por el *National Institute of Standards and Technology (NIST)*, los cuales, a través de quince pruebas [1] aplicadas a las propiedades de los bits aleatorios de salida del sistema cuyos resultados están apegados a distribuciones de probabilidad para cada tipo de entrada específica y cuya salida fue medida con anterioridad, permiten determinar si un mecanismo diseñado para este fin es lo suficientemente seguro y está listo para ser implementado como función de los protocolos criptográficos existentes y validados por el mismo instituto.

Diversos sistemas para el cumplimento de este propósito han sido desarrollados y actualmente el NIST ha validado varios de ellos de acuerdo al porcentaje de cumplimiento de cada una de las pruebas pertinentes.

En la actualidad los sistemas operativos cuentan con un Generador de Números Aleatorios (RNG) propio, el cual tiene su base en el proceso de obtención de pseudoaleatoriedad de bits y trabajan recolectando entropía a través de fuentes físicas y no físicas del sistema permitiendo devolver números suficientemente pseudoaleatorios al tener una cantidad considerable de entropía recolectada después de determinado tiempo. A continuación se presenta una tabla descriptiva y comparativa para cada uno de estos mecanismos pertenecientes a los principales sistemas operativos disponibles en el mercado [5].

Sistema Operativo	Modo de obtención de aleatoriedad	Ventajas	Desventajas
Microsoft Windows [6]	Basa su generación el cifrador de bloque AES-CTR, almacenando siempre una semilla para alimentar los diversos generadores de bits. Existe un PRNG *raíz* que se inicializa junto con el sistema y del cual se realimentan los demás generadores y es alimentado con una nueva semilla de forma periódica. Así mismo, cuenta con un contador de 64 bits que contiene el número de veces que se ha realimentado a este generador raíz.	Diferentes tipos de fuentes de entropía. Buena calidad de los bytes pseudoaleatorios al realimentar el PRNG para cada proceso.	El proceso de realimentación de los diferentes PRNG del sistema es bastante costoso.
GNU/Linux [7]	El *kernel* de Linux mantiene dos archivos, */dev/random* y */dev/urandom,* que funcionan como colectores de entropía y cuya principal fuente son diferentes valores que guarda el sistema ante las decisiones del usuario durante su uso. La principal diferencia entre ambos procesos que devuelven bytes pseudoaleatorios es la calidad de los números retornados durante determinado tiempo.	*/dev/random* espera a tener suficiente entropía recolectada para poder devolver un número aleatorio.	*/dev/random* bloquea las salidas ante una alta demanda de peticiones de números, por lo cual retarda el proceso.
		/dev/urandom no bloquea las peticiones por más frecuentes que sean durante algún proceso de alta demanda.	*/dev/urandom* no espera a tener la suficiente entropía recolectada para devolver números pseudoaleatorios de mayor calidad.
OSX [8] [9]	Emplea el algoritmo *Yarrow*, que funciona similar al proceso de obtención de bytes en Linux a diferencia de que su límite son 160 bits de salida.	Fue creado con un proceso en contra de ataques. Los archivos donde se almacenan las semillas para la realimentación son constantemente actualizados.	El atacante que conozca el mecanismo de generación puede tener total acceso al sistema cuya seguridad depende de las salidas de este proceso.

Tabla 1. Descripción y comparación de los PRNGs para los principales sistemas operativos actuales.

2. Objetivo

Analizar e implementar mecanismos que permitan la generación de bits pseudoaleatorios para aplicaciones criptográficas, que satisfagan los requerimientos especificados por los estándares, mediante la combinación de algoritmos de software y hardware.

Objetivos específicos

- Analizar los algoritmos generadores de aleatoriedad en hardware que satisfacen las pruebas de aleatoriedad establecidas por el NIST.
- Analizar los algoritmos generadores de aleatoriedad en software que satisfacen las pruebas de aleatoriedad establecidas por el NIST.
- Implementar algoritmos generadores de aleatoriedad para software y hardware.
- Verificar la eficiencia de los generadores de aleatoriedad de software y hardware tanto en tiempo como en calidad de los valores obtenidos.
- Comparar los resultados obtenidos en cuanto a calidad de los números aleatorios de salida, así como de la eficiencia de cada uno de los generadores.

3. Justificación

En la actualidad, existen varios mecanismos que nos permiten generar aleatoriedad criptográfica y que están implementados en todo tipo de dispositivos: desde las computadoras que son utilizadas a diario, hasta en aplicaciones más pequeñas como para los dispositivos de *IoT (Internet of Things)*. Por tal motivo los generadores de números aleatorios *(RNG)* juegan un papel muy importante en cada una de estas aplicaciones que implementan protocolos de criptografía cuyas llaves o valores iniciales (como los vectores de inicialización o números primos) deben ser generados de forma totalmente aleatoria para combatir cualquier tipo de ataque que pueda vulnerar el sistema si estos valores resultan predecibles.

Los mecanismos que generan aleatoriedad pura son realmente costosos y su funcionamiento no satisface a las peticiones continuas que se hacen a la aplicación encargada de proporcionar estos valores, así como tampoco cumple con una implementación adecuada para todo tipo de dispositivos, pues los recursos resultan ser limitados y por lo tanto se tiene que optimizar el diseño prescindiendo de algunas características importantes para un buen funcionamiento de los mismos.

Entonces, la implementación de los *True Random Number Generators (TRNGs)* resulta poco eficiente y es por ello que sólo se utilizan para generar semillas que alimentan a los *Pseudo-random Number Generators (PRNG)*. Todas estas limitantes proveen un reto bastante grande para los desarrolladores tanto de software como de hardware en este ámbito, pues se debe buscar la manera de optimizar estos mecanismos para dos propósitos principales:

1. Proporcionar bits saludables o fuertes que cumplan con las pruebas de aleatoriedad establecidas por el estándar.
2. Incrementar la velocidad de procesamiento para la generación y salida de estos bits de tal forma que puedan ser usados en aplicaciones de alta demanda.

Un mal generador de números pseudoaleatorios o aleatorios puede devolver una salida ya sea con bits predecibles o con bits reusados de otra generación aleatoria, ambas bastante inseguras para ser implementadas junto con cualquier protocolo criptográfico y que pueden tener consecuencias catastróficas para cualquier sistema que lo utilice.

A través de los años se han detectado varias fallas en los generadores aleatorios [10, 11, 12], donde una de las más conocidas fue la del *PRNG* del sistema operativo Linux [13] en el año 2006, donde cualquier atacante no autenticado con apenas conocimiento de cómo vulnerar un sistema con un ataque de fuerza bruta, podía encontrar los valores aleatorios pertenecientes al sistema, ya que los datos que el generador devolvía eran recurrentes, lo que los volvía totalmente predecibles [14].

Analizar los métodos desarrollados actualmente para estos mecanismos, nos permite observar en dónde se pueden implementar mejoras para la generación pseudoaleatoria de bits, retomando la importancia de las pruebas aplicadas los bits de salida obtenidos para cada uno de los componentes de la arquitectura del generador.

Así, el presente trabajo tiene como fin buscar y analizar los mecanismos para la generación de números pseudoaleatorios partiendo de los estándares y métodos ya existentes, de tal manera que se pueda observar la eficacia del desempeño de un generador de este tipo tanto de forma física (hardware) como lógica (software), al ser este mecanismo parte fundamental de la seguridad en criptografía.

4. Productos o Resultados esperados

A continuación se describen los productos que se entregarán al concluir el presente trabajo terminal:

- Implementación en software de un algoritmo generador de aleatoriedad a partir de las instrucciones de AES-CTR para llamada a instrucciones RDSEED y RDRAND del procesador.
- Implementación en hardware de un algoritmo generador de aleatoriedad mediante el uso de una FPGA con procesador ARM.
- Co-diseño software-hardware de un generador de números aleatorios para aplicaciones criptográficas.

Los siguientes diagramas representan de manera general la arquitectura de la implementación que tendrá el generador de aleatoriedad en hardware así como el funcionamiento de las instrucciones para el generador en software.

En la figura 1, se describe la arquitectura y flujo de la implementación en hardware, donde el componente principal es la FPGA (Field-Programmable Gate Array), cuya salida serán las secuencias de bits aleatorios a los que les serán aplicadas las pruebas del NIST. Así mismo, la FPGA tendrá comunicación con una computadora la cual nos proporcionará los medios para programarla.

Por su parte, la figura 2 nos permite visualizar el funcionamiento de las instrucciones RDRAND y RDSEED del generador de aleatoriedad digital de los procesadores Intel, el cual permite entender de qué forma trabajan las instrucciones una vez que se invocan desde código en lenguaje C.

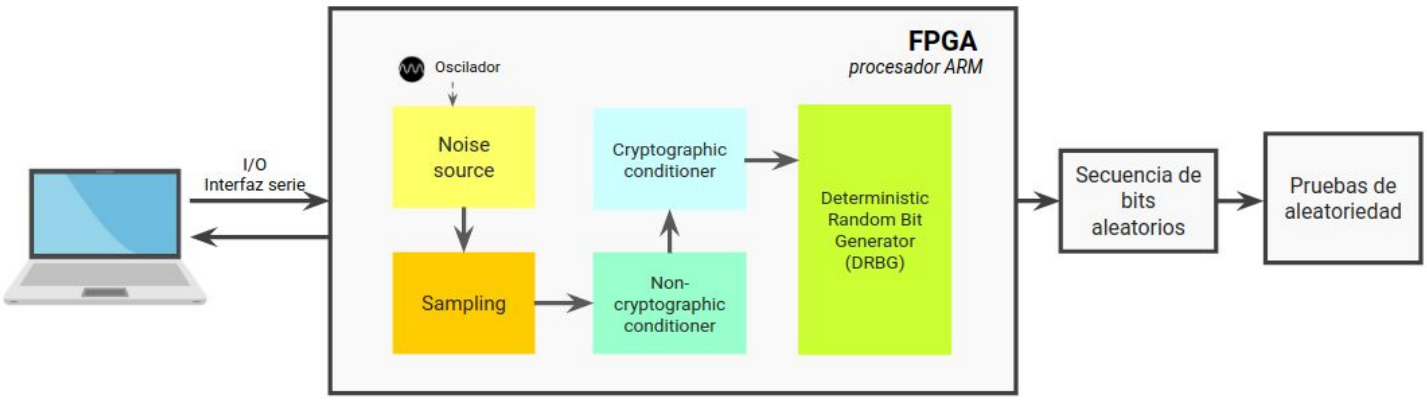

Figura 1. Arquitectura del generador de aleatoriedad en hardware.

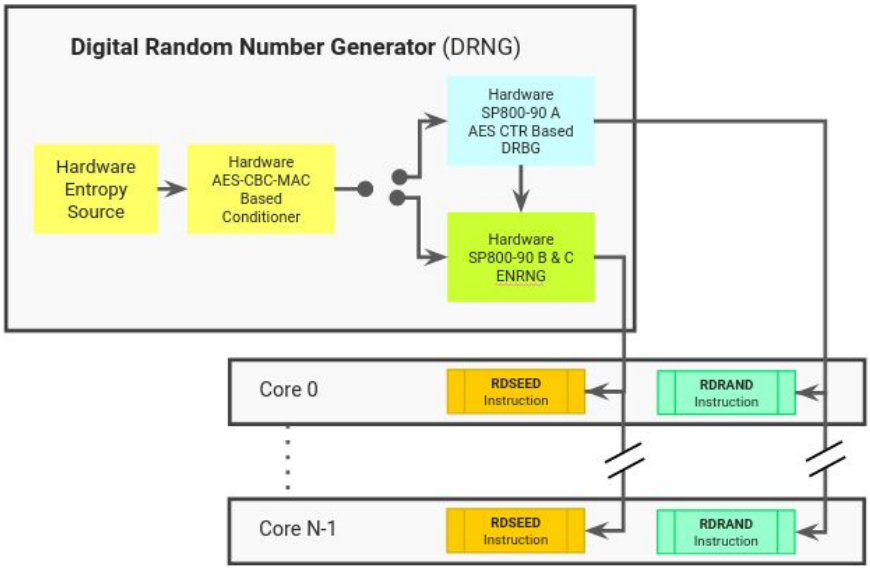

Figura 2. Arquitectura del generador de aleatoriedad digital con instrucciones RDRAND y RDSEED para un procesador Intel [2].

5. Metodología

Para el desarrollo del presente trabajo se pretende seguir las etapas propuestas por el método científico, al ser una labor en primera instancia de investigación de los mecanismos ya existentes para la generación de números aleatorios criptográficos para hardware y software, de tal forma que posteriormente se puedan implementar los mecanismos seleccionados y a partir del análisis de cada uno de estos mecanismos, se comparen los valores de salida de cada uno de ellos en cuanto a calidad y eficiencia.

El método científico aplicado a este trabajo comprende de:

1. Observación: realización del estado del arte acerca de los los mecanismos de aleatoriedad.
2. Inducción: identificación y extracción de las características principales de los mecanismos de aleatoriedad en software y hardware.
3. Hipótesis: generación de aleatoriedad de calidad con los mecanismos identificados.
4. Experimentación: implementación de los mecanismos de aleatoriedad seleccionados.
5. Análisis: aplicación de las pruebas de aleatoriedad para obtener la calidad de los bits de salida de cada una de las implementaciones.
6. Conclusión: determinación del mecanismo más adecuado para generar aleatoriedad.

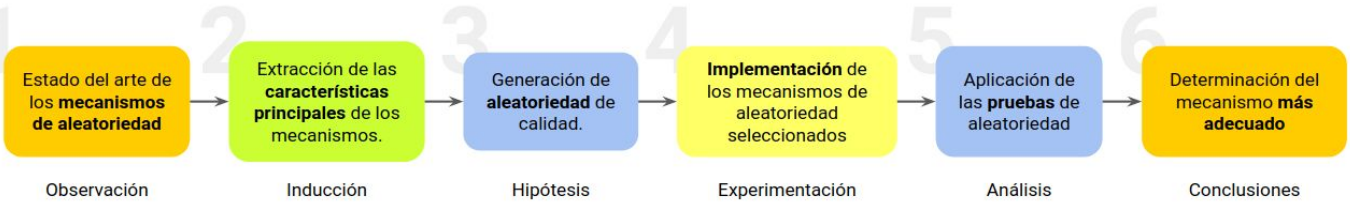

Figura 3. Secuencia del Método Científico [15] aplicada al desarrollo del trabajo.

6. Cronograma

Nombre del alumno(a): *Hernández Morales Brenda Mariana* TT No.: _ _ _ _ - _ _ _
Título del TT: ***Generación de aleatoriedad para aplicaciones criptográficas***

Actividad	FEB	MAR	ABR	MAY	JUN	AGO	SEPT	OCT	NOV	DIC
Investigación y análisis de algoritmos para generación de aleatoriedad criptográfica en software.										
Investigación y análisis de algoritmos para generación de aleatoriedad criptográfica en hardware.										
Realizar el co-diseño de la implementación software-hardware para cada uno de los generadores.										
Evaluación de TT I.										
Implementación de un generador de aleatoriedad criptográfica en software y otra en hardware.										
Verificación de la eficiencia y calidad de los generadores utilizando las pruebas del NIST.										
Comparación de la calidad de ambos generadores.										
Generación del reporte técnico.										
Evaluación de TT II.										

7. Referencias

[1] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Stefan D. Leigh, M Levenson, M Vangel, Nathanael A. Heckert, D L. Banks, (2010) A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-22 Rev 1a. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[2] Mechalas, J. P. (2018, October 17). Intel® Digital Random Number Generator (DRNG) Software Implementation... Intel® Development Topics & Technologies. https://software.intel.com/content/www/us/en/develop/articles/intel-digital-random-number-generator-drng-software-implementation-guide.html, consultada en (2020, Octubre 13).

[3] Meltem Sönmez Turan (NIST), Elaine Barker (NIST), John Kelsey (NIST), Kerry McKay (NIST), Mary Baish (NSA), Michael Boyle (NSA), (2018) Recommendation for the Entropy Sources Used for Random Bit Generation. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-90B. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf

[4] Elaine Barker (NIST), John Kelsey (NIST), (2016) Recommendation for Random Bit Generator (RBG) Constructions. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-90C (Draft). https://csrc.nist.gov/csrc/media/publications/sp/800-90c/draft/documents/draft-sp800-90c.pdf

[5] Menchalas, J. P. (2015, July 17). The DRNG Library and Manual. Intel® Development Topics & Technologies. https://software.intel.com/content/www/us/en/develop/articles/the-drng-library-and-manual.html

[6] Team, S. E. (2019, Noviembre 25). Going in-depth on the Windows 10 random number generation infrastructure. Microsoft Security. https://www.microsoft.com/security/blog/2019/11/25/going-in-depth-on-the-windows-10-random-number-generation-infrastructure/, consultada en (2020, Agosto 20).

[7] Random number generation - ArchWiki. (2020). ArchLinux Wiki. https://wiki.archlinux.org/index.php/Random_number_generation, consultada en (2020, Agosto 20).

[8] Apple Inc. (2013). random.c. Open Source Apple: Random.c. https://opensource.apple.com/source/xnu/xnu-4570.41.2/osfmk/prng/random.c.auto.html, consultada en (2020, Octubre 27)

[9] Wikipedia contributors. (2020a, August 10). Yarrow algorithm. Wikipedia. https://en.wikipedia.org/wiki/Yarrow_algorithm, consultada en (2020, Agosto 20).

[10] Leo Dorrendorf, Zvi Gutterman, and Benny Pinkas. (2007) Cryptanalysis of the Random Number Generator of the Windows Operating System. Cryptology ePrint Archive, Report 2007/419.https://eprint.iacr.org/2007/419

[11] Zvi Gutterman and Dahlia Malkhi. “Hold Your Sessions: An Attack on Java Session-Id Generation”.In:Topics in Cryptology - CT-RSA 2005, The Cryptographers’ Track at the RSA Conference 2005, SanFrancisco, CA, USA, February 14-18, 2005, Proceedings. Ed. by Alfred Menezes. Vol. 3376. LectureNotes in Computer Science. Springer, 2005, pp. 44–57.doi:10.1007/978-3-540-30574-3_5.url:https://doi.org/10.1007/978-3-540-30574-3%5C_5

[12] Robert Woolley, Mark Murray, Maxim Dounin, and Ruslan Ermilov. (2008) FreeBSD-SA-08.11.arc4random Security Advisory The FreeBSD Project arc4random predictable sequence vulnerability. https://www.freebsd.org/security/advisories/FreeBSD-SA-08:11.arc4random.asc, 2008.

[13] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. (2006) Analysis of the Linux Random Number Generator. Cryptology ePrint Archive, Report 2006/086.https://eprint.iacr.org/2006/086

[14] CERT Coordination Center. (2008, May 15). CERT/CC Vulnerability Note VU#925211. Software Engineering Institute CERT Coordination Center. https://www.kb.cert.org/vuls/id/925211, Consultada en (2020, Octubre 23)

[15] Bacon, F. (1970). Método científico. Recuperado de http://definicion. de/metodo/# ixzz3Q9wdBKEf, Consultada en (2020, Octubre 29)

8. Alumnos y Directores

Hernández Morales Brenda Mariana.- Alumno de la carrera de Ingeniería en Sistemas Computacionales en la Escuela Superior de Cómputo, Especialidad Sistemas, *No. de Boleta:* 2018630062: , *Tel.* 5540035919. , *Email:* bhernandezm1701@alumno.ipn.mx, marbrehi@gmail.com

Firma:_____

Díaz Santiago Sandra.- Doctorado en Ciencias en Computación en CINVESTAV-IPN, en 2014, Maestría en Ciencias (Matemáticas) en la UAM-Iztapalapa, 2005. Licenciatura en Computación en UAM-Iztapalapa, en 1998. Profesor en ESCOM (Departamento de Ciencias e Ingeniería de la Computación), desde 2004, Áreas de Interés: Criptografía, Pseudoaleatoriedad, Seguridad Demostrable, Ext 52022, email: sdiazs@gmail.com, sdiazsa@ipn.mx.

Firma:_____

Mancillas López Cuauhtemoc- Posdoctorado sobre criptografía aplicada en la Universidad de Lyon en St. Etienne, Francia, 2014-2018. Doctorado en Ciencias en Computación en CINVESTAV-IPN, 2013. Maestría en Ingeniería Eléctrica en CINVESTAV-IPN, 2007. Ingeniería en Comunicaciones y Electrónica en ESIME-IPN, 2004. Investigador Titular en el Departamento de Computación del CINVESTAV-IPN. Áreas de interés: criptografía simétrica, aritmética computacional, seguridad demostrable, hardware reconfigurable. Tel: 5557473800 Ext 4240, email: cuauhtemoc.mancillas@cinvestav.mx.

Firma:_____

Protocolo modificado al 06.Nov.2020

Generación de aleatoriedad para aplicaciones criptográficas

*Alumnos: *Hernández Morales Brenda Mariana*
Directores: Díaz Santiago Sandra, Mancillas López Cuauhtemoc
**e-mail: bhernandezm1701@alumno.ipn.mx*

El anterior envío del protocolo el día **05.Nov.2020 a las 08:58 hrs., contiene errores** en el cronograma únicamente**.**