Cómo mejorar la seguridad de un sistema criptográfico con pocos recursos y creatividad

Gastón N. Charkiewicz[1]*

Resumen
The purpose of this research is to present a way to improve Stream Cypher Cryptographic Systems using few resources. The approach is done by constructing True Random Number Generator prototipes and by analyzing their application to the mentioned Cryptographic Systems.

Keywords
random — pseudorandom — number — open source harware — arduino — stream ciphers — cryptography

[1] *Tecnicatura Universitaria en Programación Informática, Universidad Nacional de Quilmes, Argentina*
*gnchark@gmail.com

Índice

1. Introducción

En este proyecto se buscará mejorar el nivel de seguridad de un sistema criptográfico de Cifrado de Flujo utilizando pocos recursos. Para ello, se construirán prototipos de Generadores de Números Aleatorios (o True Random Number Generators) en hardware libre, utilizando recursos fáciles de obtener y de bajo costo monetario. Más tarde se analizará su utilización en el sistema mencionado previamente.

En la primera parte se introducirán los conceptos principales que serán la base del presente informe. Para ello se iniciará presentando el concepto de la aleatoriedad en la Computación y se mostrarán los distintos tipos. En las siguientes dos secciones, se tratará la generación de números verdaderamente aleatorios y su validación mediante pruebas específicas respectivamente.

En la sección Arduino se presentará la plataforma de hardware libre y se especificará por qué será utilizada como parte de este proyecto. Finalizando la primera parte, se introducir á el Sistema de cifrado de flujo.

En la etapa de desarrollo se presentará la implementación de tres prototipos para la generación de aleatoriedad. Los detalles del primer prototipo construido se analizarán en “Prototipo 0”. También se tratarán los problemas que surgieron con el mismo en la sección “Problemática”. Luego se explicará y analizará segundo prototipo, el cual permite realizar una validación específica en Pruebas Empíricas de sus resultados generados. Con este prototipo ya será posible la obtención de números verdaderamente aleatorios. Finalmente, se presentará un tercer prototipo, capaz de generar un flujo de bits puramente aleatorios.

A continuación se presentarán las conclusiones. Allí se disertará acerca de la validez de los prototipos, y se explicará su aplicación al desarrollo de sistemas de cifrado de flujo. Allí mismo se mostrarán las posibles ventajas.

En “Trabajos Futuros” se enumera el posible trabajo a realizar más adelante en este mismo proyecto.

2. Contexto

Actualmente, la mayoría de la aleatoriedad presente en la computación no es “pura”. Esto hace que sistemas criptográficos como el de cifrado de flujo sean particularmente interesantes de analizar, puesto que su implementaci ón actual es a

base de este tipo de aleatoriedad “no pura”, llamada pseudo-aleatoriedad.

2.1 La aleatoriedad en la Computación

En la computación actual, existen dos tipos de generadores de números aleatorios: los pseudoaleatorios (también llamados Pseudo Random Number Generators) y los “verdaderamente” aleatorios (o True Random Number Generators). Decir que “un número es aleatorio o no” es algo errado, puesto que los números en sí no conllevan esa responsabilidad: es del generador de los mismos. Los números simplemente son números. De todas formas, oír hablar de “números aleatorios” y de “números pseudoaleatorios” es normal. Habiendo hecho esta aclaración previa, se referirá a los mismos como “números aleatorios” y “números pseudoaleatorios” por simpleza.

Los generadores de números pseudoaleatorios utilizan un algoritmo, más específicamente una función matemática que se evalúa con un número (llamado semilla o *seed*) difícil de predecir, para generar un valor. La propiedad más importante de una función de generación de números pseudoaleatorios es: dada una cantidad no arbitraria de resultados de la función aplicada a distintas semillas, en los resultados no debe notarse una tendencia muy marcada de variación uniforme de los mismos. Es decir, no deben observarse patrones de repetición en los resultados de forma predecible.

Es muy importante especificar que la generación de números pseudoaleatorios es un proceso determinístico, e incluso cuando las probabilidades de predecir los resultados sea remota, nunca va a ser nula. **Siempre existirá la posibilidad de predecir el resultado de un generador de números pseudoaleatorios**.

En cambio, la generación de números aleatorios se lleva a cabo por un proceso **no determinístico**, haciendo que la predicción de los mismos sea **imposible**.

En las Figuras 1 y 2 se muestran dos bitmaps generados con valores aleatorios. El primero utiliza valores aleatorios, y el segundo valores pseudoaleatorios.

Las computadoras actuales utilizan pseudo aleatoriedad. Su estructura les impide generar números aleatorios por su propia cuenta. Von Neumann, -el creador de la arquitectura de cómputos que lleva su nombre y que más tarde sería la de la mayoría de las computadoras actuales-, se encontró con esta misma problemática. Citando a James Gleick en su libro La Información, *“Von Neumann se dio cuenta de que una computadora mecánica, con sus algoritmos deterministas y su limitada capacidad de almacenamiento, no podría generar nunca números verdaderamente aleatorios. Tendría que conformarse con números pseudoaleatorios: números generados de forma determinista que se comportaran como si fueran aleatorios. Eran lo bastante aleatorios para su uso con fines prácticos.”*[1]

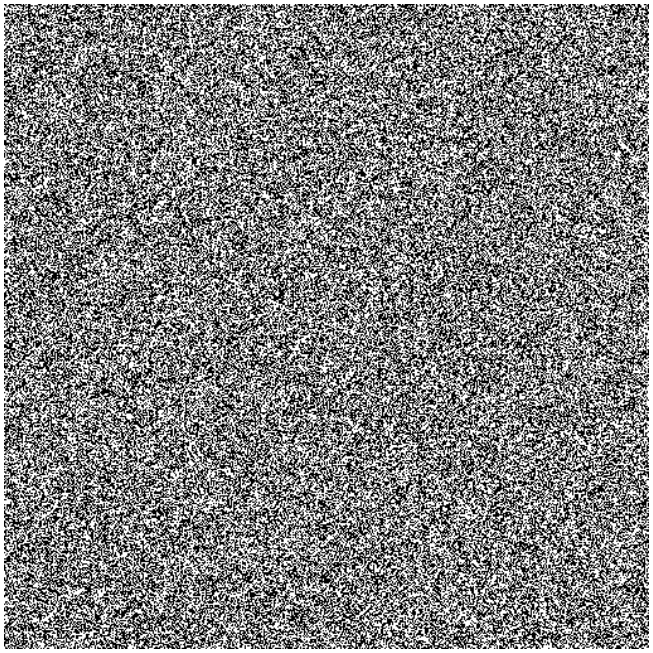

Figura 1. Bitmap generado con Random.org (TRNG sensando ruido atmosférico)

Por esto mismo, las computadoras deben valerse de hardware específico para poder generar números aleatorios.

Sin embargo, ha pasado mucho tiempo desde la especificación de la arquitectura de Von Neumannn, y se ha avanzado bastante sobre la problemática de generar números puramente aleatorios.

2.2 Generación de números verdaderamente aleatorios

Hay dos formas de obtener números aleatorios. Se detallan a continuación.

2.2.1 Fenómenos físicos con propiedades cuánticas aleatorias

Sensar este tipo de fenónemos para obtener números puramente aleatorios es el “estándar de oro”, gracias a una particularidad de la mecánica cuántica: los resultados de sus eventos no pueden ser predecibles. En este caso, hay dos fuentes de números aleatorios posibles: mecánica cuántica a nivel atómico o subatómico, y ruido térmico. Algunos tipos de ruido térmico terminan siendo sucesos de mecánica cuántica.

2.2.2 Fenómenos físicos sin propiedades cuánticas aleatorias

Hay algunos fenómenos físicos que, sin necesidad de ser analizados con mecánica cuántica, pueden ser fuente de números puramente aleatorios. Algunos pueden ser:

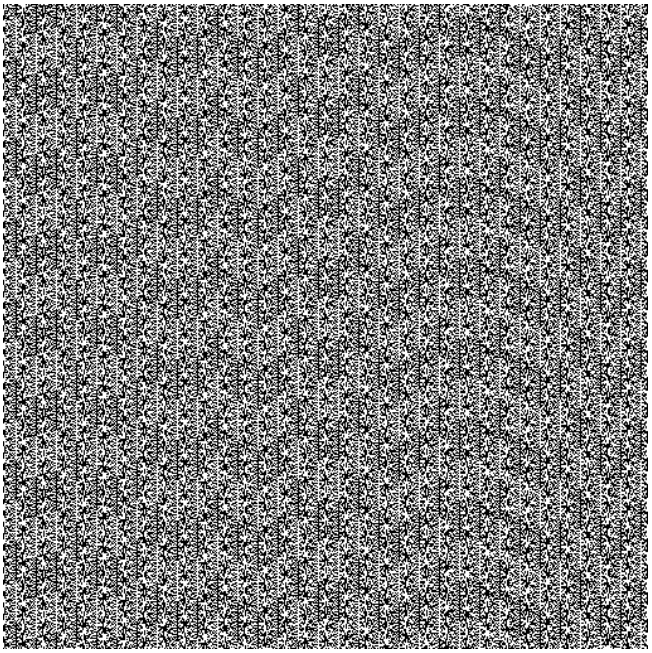

Figura 2. Bitmap generado con la función rand() de PHP en Windows

- Ruido térmico de una resistencia, amplificada para proporcionar un voltaje al azar.
- Ruido de avalancha generado por un diodo avalancha, o el ruido del quiebre de un diodo Zener polarizado inversamente.
- Ruido atmosférico, detectado con un receptor de radio conectado a una computadora.
- Derivaciones de Reloj (o Clock Drift), la variación de velocidad de un reloj comparado con otro. Dados dos relojes, en un momento no arbitrario la velocidad de alguno de ellos variará, haciendo que ambos tengan mediciones distintas. La relación entre ambos es sensada y utilizada como fuente de números aleatorios. Este método de generación es inseguro, puesto que tiende a ser el más expuesto a ataques.

A continuación, se explicará la validación de números aleatorios.

2.3 Probando la validez de nuestro generador de números aleatorios

Una vez diseñado un generador de números aleatorios, es necesaria su validación: es necesario probar que los resultados generados son propiamente aleatorios. Las demostraciones no son simples. La forma más habitual de probar un generador de números aleatorios es utilizar herramientas automatizadas que verifiquen ciertas propiedades en diversas muestras tomadas con él. Estas muestras deben ser muy grandes para poder verificar estas propiedades con mayor exactitud. Se pueden observar ejemplos de forma visual de sistemas de generación de números aleatorios con propiedades opuestas en las Figuras 3 y 4.

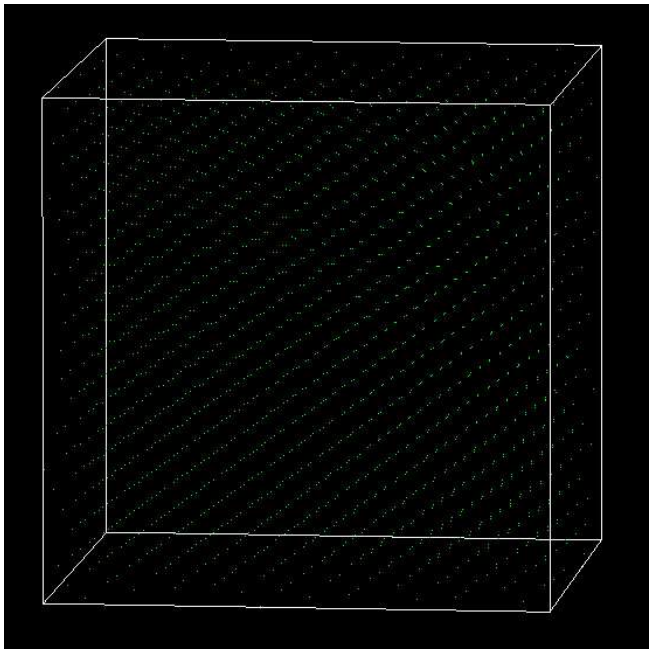

Figura 3. Si el generador de números aleatorios es malo, se podrían obtener muestras en patrones uniformes

2.4 Arduino

De la página de Arduino: *"Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos. Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basasdo en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, Processing, MaxMSP)."*[2]

Arduino es hardware libre. La cantidad de usuarios de Arduino es cada vez mayor, y no está limitada a perfiles con conocimientos técnicos. Por ejemplo, es utilizado mucho con fines artísticos. Arduino es fácil de utilizar. Además, es amigable con materiales electrónicos habituales, y se pueden conseguir piezas específicas (como acelerómetros o sensores de presión) en Internet o negocios especializados.

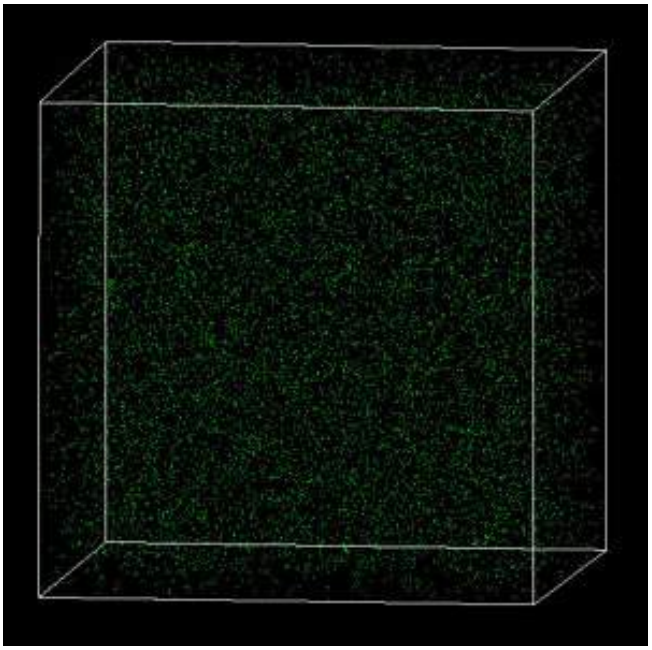

Figura 4. Si el generador es bueno, se obtendrán muestras verdaderamente aleatorias

2.5 Cifrado de flujo (criptografía de palabra clave)

Uno de los mecanismos habituales de cifrado, es el cifrado de flujo. Se utiliza cuando es necesario cifrar flujos de datos de tamaño no arbitrario (en caso de que lo fuera, se podría utilizar el Cifrado en Bloques). Por este motivo, es muy utilizado en telecomunicaciones, proceso donde luego de digitalizar la voz, es necesario enviar ese flujo de bits de forma cifrada por una red.

En la Figura 5 se puede apreciar el funcionamiento de este tipo de cifrado.

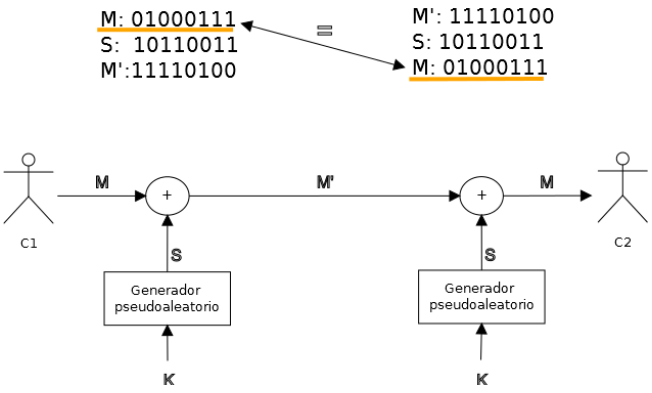

Figura 5. Esquema de cifrado de flujo habitual

En este ejemplo, cada bit del mensaje (ese flujo de bits constante) que C1 desea enviarle a C2 es combinado, utilizando la función lógica XOR, con el bit correspondiente del flujo aleatorio S, para generar de esta forma el bit correspondiente al flujo de salida. Para que C2 pueda descifrar el mensaje, bastará con aplicar la misma función lógica a cada bit del mensaje recibido, junto con el bit correspondiente del flujo aleatorio S (el generador es compartido). Este proceso es constante.

Al flujo de bits que se utiliza para "oscurecer" el mensaje M, se le denomina flujo de clave. Se genera gracias a un valor aleatorio K, y a un generador pseudoaleatorio, una función matemática, cuya particularidad es que los resultados ante valores de entrada no arbitrarios serán una muestra uniforme, y sin patrones uniformes entre sí de forma general. Es fundamental citar que, si el flujo de clave es seguro, el flujo de datos cifrados también lo será.

Ya tocando aspectos técnicos, la generación de este flujo de clave no debería exigir demasiado esfuerzo, puesto que de suceder esto, podría retrasar el flujo de datos.

El aspecto de seguridad más importante del Cifrado de flujo, es el valor K. Este valor debe ser lo suficientemente aleatorio como para no ser predecible, puesto que, en caso de serlo, el mensaje cifrado sería fácilmente descifrable.

Principio de Kerckhoffs

La seguridad del sistema debe recaer en la seguridad de la clave, debiéndose suponer conocidos el resto de los parámetros del sistema criptográfico.

3. Desarrollo de los prototipos

Se buscó construir prototipos que generasen aleatoriedad, valiéndose de materiales ya disponibles o fáciles de conseguir. Luego se preparó un ambiente donde las pruebas pudieran llevarse a cabo con la mayor facilidad posible, utilizando tecnologías enumeradas más adelante. A continuación se enumeran los prototipos construidos, y la experiencia con cada uno de ellos.

3.1 Prototipo 0

Fue el primer prototipo a desarrollarse. El diseño de este generador de números aleatorios está basado en el de Rob Seward, quién diseñó su propio True Random Number Generator y lo compartió en su página web[3].

Los materiales utilizados por Rob Seward fueron los siguientes:

- 1 Arduino
- 3 Transistores 2N3904
- 2 Resistencias de 4,7k
- 1 Resistencia de 10k
- 1 Resistencia de 1.5M
- 1 Capacitor de 0.1μf

- 1 Capacitor de 10μf
- 1 Protoboard
- 1 Transformador de 12V

Se buscó lograr el mismo circuito utilizando materiales iguales o casi equivalentes. Los materiales utilizados fueron:

- 1 Arduino (específicamente, un Arduino Duemilanove con un microcontrolador Atmel ATMEGA 168)
- 3 Transistores 2N3904
- Cada resistencia de 4.7k se reemplazó por 2 resistencias de 10k en paralelo(equivalentes a 5k)
- 1 Resistencia de 10k
- 1 Resistencia de 1.5M
- 1 Capacitor de 0.1μf
- 1 Capacitor de 10μf
- 1 Protoboard
- 1 Transformador de 12V

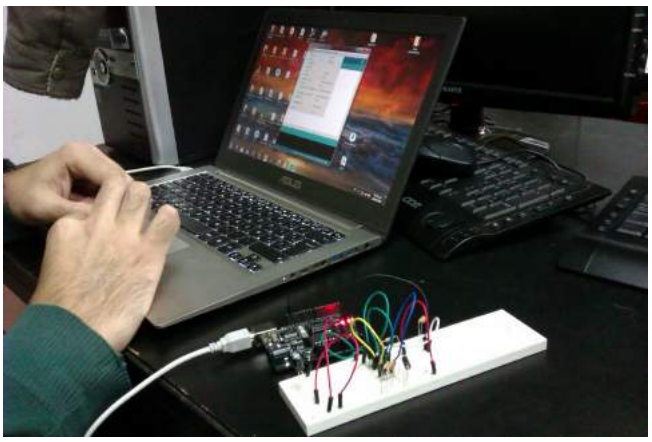

Figura 6. Primer prototipo funcionando

Los dos transistores con sus bases conectadas generan una avalancha de ruido en una unión p-n polarizada inversamente[4]. Este ruido es aplificado por el tercer transistor, y se envía a un pin del Arduino luego de pasar por un divisor de voltaje. Utilizando un osciloscopio, se puede observar una señal muy borrosa e irregular. Una vez en el Arduino, esa aleatoriedad es convertida a un flujo de ceros y unos.

El diagrama del circuito se puede observar en la Figura 9.

El código en el Arduino se encuentra enúmerado en “Apéndice: Código Arduino del Prototipo 0”,en este proyecto. Funciona de la siguiente forma: por los primeros 10 segundos, se sensa la señal del circuito en la entrada analógica utilizada del Arduino. Se determina la media. Luego, para cada siguiente lectura de dicha entrada analógica, se determina si está por encima o por debajo de la media. Si está por encima, se genera

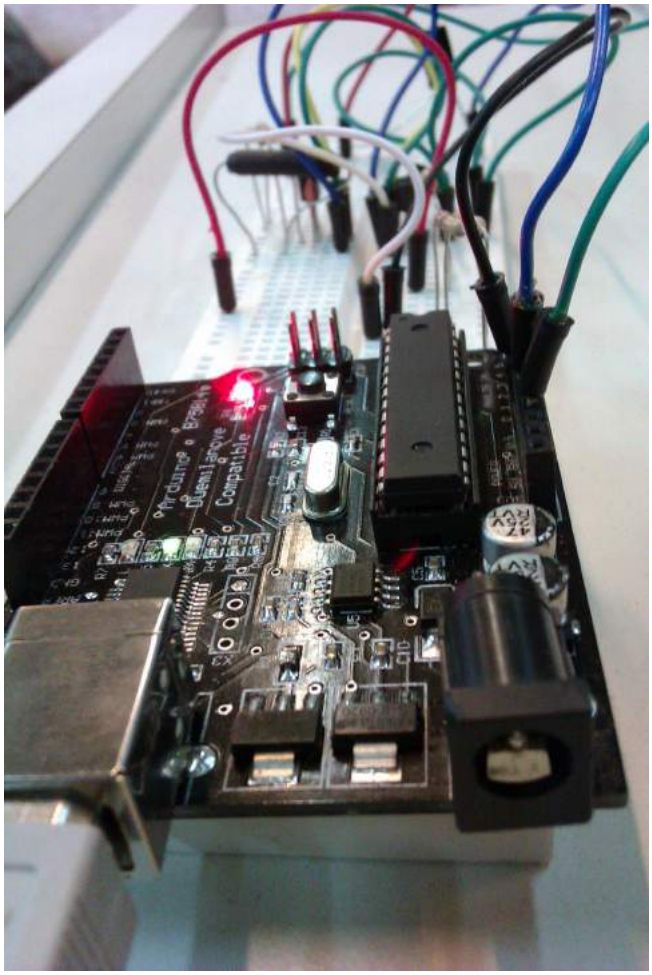

Figura 8. Hardware externo del primer prototipo

un 1 como salida en el puerto serie; en caso de que esté por debajo, se genera un 0. Es necesario reducir el voltaje de la polarización en la salida, para que la misma tenga el formato deseado. Para eso, a la misma se le aplica previamente filtrado XOR y filtrado Von Neumann (esto sucede en el Arduino, el código lo contempla).

Se pueden ver imágenes del Prototipo 0 en las Figuras 6, 7 y 8.

3.1.1 Obteniendo las muestras de la forma correcta

Este programa por si solo, generaba un flujo de bits aleatorio, y lo enviaba a través del puerto serie. El flujo se podía apreciar utilizando la herramienta “Monitor Serial” de la IDE de Arduino[5]. Sin embargo, esta herramienta no era cómoda para obtener una cantidad significativa de muestras, y además Arduino cuenta con la limitación de no poseer herramientas para la manipulación de archivos de texto plano, el formato de salida deseado para las muestras. Por esto mismo, se diseñó un un script muy simple hecho en Python[6], utilizando pySerial[7] para la manipulación del puerto Serie.

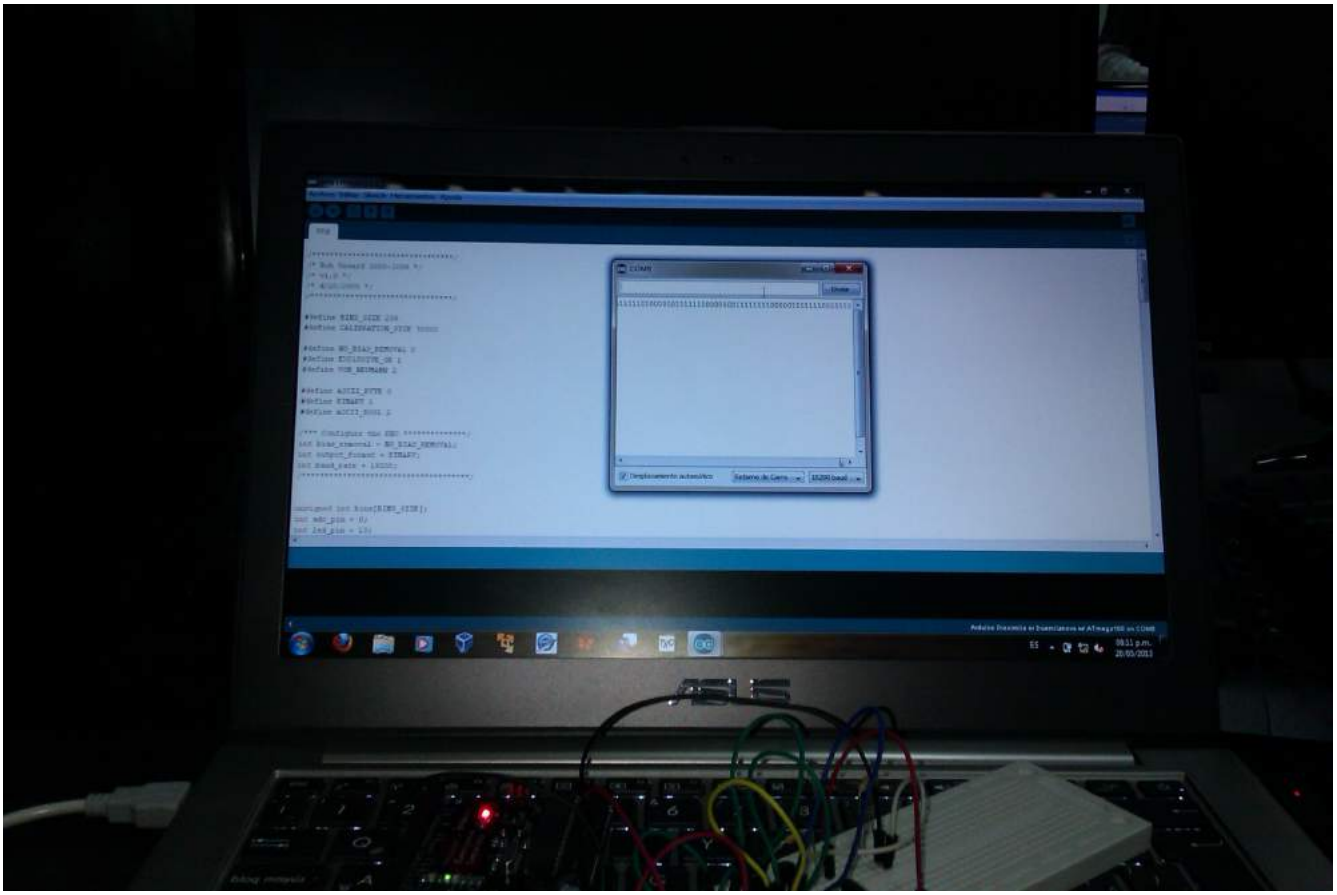

Figura 7. Resultados del primer prototipo funcionando

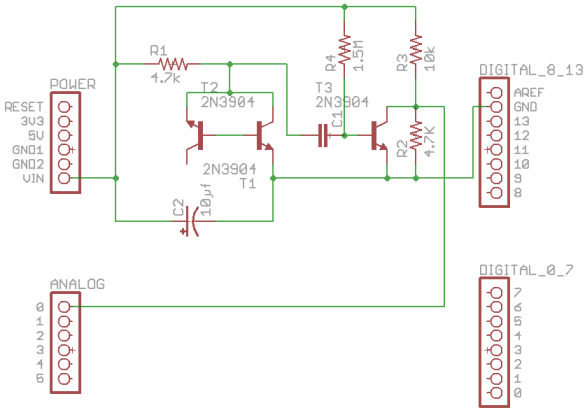

Figura 9. Circuito externo del primer prototipo

```
import serial

# inicializar el Serial

arduino = serial.Serial
          ("/dev/ttyUSB0",19200)

# abrir el archivo de texto

results = open('results.txt','w')

while True:
    results.write(arduino.readline())
```

Se preparó una computadora con el prototipo y el script. La misma tenía un sistema Operativo Ubuntu 12.04 de 64bits[8] (la de las imágenes correspondía a las primeras pruebas realizadas, funcionaba con Windows 7 64bits). Se la dejó tomando muestras por unas 25 horas. Con esto, se logró sensar un flujo de alrededor de 33 millones bits en un archivo de texto plano.

Sin embargo, durante la toma de muestras surgieron imprevistos.

3.1.2 Problemática

En el momento de recolectar las muestras, se notó algo sumamente negativo: uno de los capacitores del circuito había explotado (ver Figura 10).

Esto inmediatamente hizo que las muestras recolectadas se

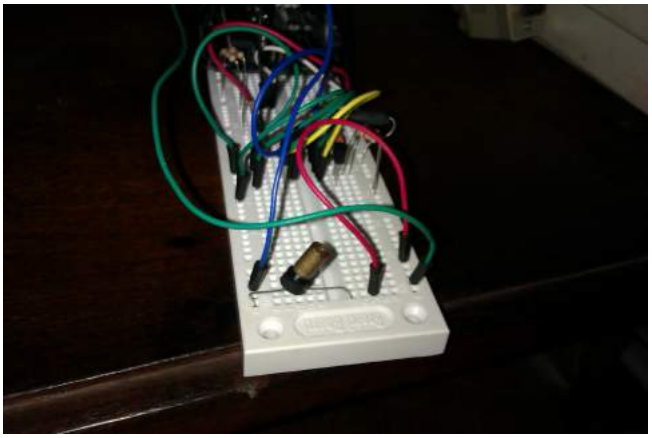

Figura 10. Capacitor explotado del Prototipo 0

descarten, puesto que fueron generadas con un circuito no íntegro. Se procedió a la construcción de un nuevo generador.

3.2 Prototipo 1

Gracias al apoyo de Rob Seward, quién proveyó de documentación acerca de su prototipo y trabajos futuros sobre el mismo, es que se llegó al diseño de un prototipo con un funcionamiento completamente distinto. El nuevo prototipo tuvo la particularidad de no poseer hardware externo al Arduino, pues se valía exclusivamente de este último para la generación de muestras aleatorias(ver Figura 12). Las muestras aleatorias se obtienen gracias al jitter del watchdog en el microcontrolador ATMEGA del Arduino. Sin entrar en detalles, el jitter[9] es una especie de ruido no deseado, y junto con el watchdog[10] sirven para determinar cuando el Arduino se bloquea. Para detalles más específicos de su funcionamiento, se puede consultar el manual y el código fuente de de la herramienta utilizada, llamada Entropy[11].

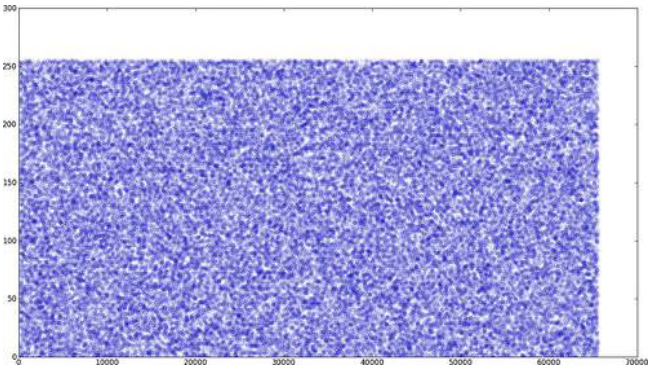

Figura 11. Pruebas de Entropy en un Arduino Duemilanove. Las mismas muestran que no hay uniformidad en patrones, y una correcta distribución

Para poder utilizar esta herramienta que genera muestras sin hardware externo al Arduino, basta con instalarla y utilizar la

Figura 12. El prototipo 1 no requería software externo al Arduino para generar muestras

library que provee en un entorno Arduino. El código en este último es muy simple de entender, y es el siguiente:

```
#include <Entropy.h>

void setup()

{

//Se inicializa el puerto serial
//con el baudrate 115200

 Serial.begin(115200);

//Se prepara Entropy

 Entropy.Initialize();

}

void loop()

{

//Se imprime un valor random en serial

 Serial.println(Entropy.random());
}
```

Para poder contar con esos valores que se envían por el puerto serial, se utilizó el mismo script Python; la única modificación fue cambiarle el baudrate, de esta forma:

```
import serial

# inicializar el Serial
#se cambia el baudrate a 115200

arduino = serial.Serial("/dev/ttyUSB0",19200)
```

```
# abrir el archivo de texto

results = open('results.txt','w')

while True:
    results.write(arduino.readline())
```

A fines de facilitar la manipulación remota del equipamiento, se configuró un servidor SSH[12] para poder acceder al sistema encargado de generar las muestras, y poder monitorearlo de forma no presencial.

3.2.1 Pruebas empíricas sobre el prototipo

La mayor cantidad de pruebas ya estaba realizada previamente por los desarrolladores del prototipo.

Las pruebas (que se encuentran disponibles)[13] arrojaron la siguiente conclusión: las muestras pueden ser consideradas aleatorias (ver Figura 11). Además, en lo respectivo a Arduino, este prototipo es uno de los que mejores resultados genera.

3.3 Prototipo 2

La finalidad del tercer prototipo es generar directamente un flujo de bits aleatorio (en la sección siguiente se explicará porqué se buscó este tipo de salida).

Para ello, también se utilizó Entropy. Se modificó el código Arduino para que generase un flujo continuo de bits aleatorios. El mismo fue el siguiente:

```
#include <Entropy.h>

void setup()

{

//Se inicializa el puerto serial
//con el baudrate 115200

Serial.begin(115200);

//Se prepara Entropy

Entropy.Initialize();

}

void loop()

{

//Se imprime un bit random en serial

Serial.println(Entropy.random(2));
}
```

Para sensar las muestras, se utilizó el mismo script Python. De esta forma, se logró generar un flujo de bits aleatorio.

4. Conclusiones

Finalmente, evaluemos las repercusiones del trabajo realizado aplicado a un sistema criptográfico de flujo.

En un sistema de Cifrado de Flujo, es fundamental tener un flujo de clave seguro. Como este último se constituye de un valor aleatorio y de un generador, es importante que este proceso sea constante y que no represente un gran esfuerzo, para de esta forma no retrasar la generación del flujo de datos cifrados.

La utilización del Prototipo 1 para la generación del valor de entrada al Generador Pseudoaleatorio (llamado k en el diagrama de la Figura 5) aumentaría seriamente el grado de robustez del sistema.

En caso de que un atacante a dicho sistema quisiese dar con este valor k, tendría las siguientes posibilidades:

- Acceder al hardware de la misma forma en que los comunicandos lo hacen, y tomar el mismo valor que ellos determinaron como k.
- Acceder al puerto serie de la computadora que utiliza el prototipo. Restaría determinar cuál de los valores generados es el valor k.

Evidentemente estos ataques tienen una complejidad inherente. La única alternativa a los mismos, es un ataque de fuerza bruta. De todas formas, gracias a Entropy y Arduino se podría generar claves alfanuméricas y de mayor tamaño, disminuyendo la probabilidad de que las mismas sean descifrables.

El prototipo 2 podría utilizarse para suplantar el Generador Pseudoaleatorio. En este caso, se contaría con un Generador Aleatorio.

En este caso, la única forma posible de dar con los valores generados, sería obteniendo acceso al Generador. Aun considerando los sistemas de cómputos más actuales y poderosos, un ataque de fuerza bruta a un mensaje cifrado en este sistema no es posible.

Todos los valores generados por los prototipos son puramente aleatorios. Al no ser generados por un proceso determinista, no hay forma en que estos puedan ser predichos. Como se ha visto en los ejemplos, sólo se podría dar con los mismos accediendo al mismo sistema que utilice los prototipos (como por ejemplo a través de malware específico), y aun así eligiendo el valor utilizado por los comunicandos.

Un detalle no menor, es el tipo de hardware utilizado: Arduino es hardware libre. Esto implica que la manipulación del mismo

no está restricta. Además, su constitución es completamente visible, y en caso de pruebas muy específicas, esto también es una ventaja. A diferencia del hardware privativo, donde no se tiene acceso completo a su referencia y la existente es acotada, en este caso podremos realizar las pruebas que queramos, conociendo el funcionamiento más íntimo del prototipo, o incluso modificándolo, cosa que con otro tipo de hardware tampoco es posible.

Para finalizar, la utilización de este prototipo no es restrictiva a sistemas de cifrado de flujo: la generación de números aleatorios tiene aplicación en juegos de azar, muestras estadísticas, simulación, diseño completamente aleatorio[14], o global conciousness[15].

5. Trabajos futuros

A continuación se enumeran algunos de los aspectos a trabajar en el futuro:

- Desarrollar una batería de testeo para números aleatorios propia. Este era uno de los aspectos primarios al iniciar el proyecto, pero no se implementó por falta de tiempo. Debe desarrollarse en Haskell[16], puesto que provee un ambiente donde no hay efectos de lado, y además está predispuesto para procesamientos de números y magnitudes. Se podría utilizar el módulo SerialPort[17] para obtener los resultados en el puerto serie.
- Retomar el desarrollo del primer prototipo, construyendo también una jaula de Faraday[18] para que los resultados del mismo no sean modificables desde el entorno.
- Evaluar los trabajos realizados de criptografía cuántica, y lograr establecer una relación con la generación de números aleatorios mediante procesos de esa misma índole.

Agradecimientos

Quiero agradecer a las siguientes personas por hacer este proyecto posible. A Daniel Ciolek, por orientarme al comenzar a probar los prototipos. Al empezar no lograba determinar si la prueba formal de los mismos era posible, o si debía probarlos directamente de forma empírica. Si bien al final no hizo falta probar los prototipos (ya estaban probados por los que los diseñaron), podría haberlo hecho sin problemas, porque ya contaba con las herramientas necesarias y sabía cómo hacerlo. A Gonzalo Sánchez y a Julián Skalic, por ayudarme con la verificación y construcción de los prototipos cuando se presentaron complicaciones.

Referencias

[1] James Gleick. *LA INFORMACION: HISTORIA Y REALIDAD*. CRÍTICA, 2012.

[2] Página web de arduino (en inglés). `http://www.arduino.cc/`, jun 2013.

[3] Prototipo rng2 de rob seward (en inglés). `http://robseward.com/misc/RNG2/`, June 2013.

[4] p-n junction (en inglés). `http://www.powerguru.org/p-n-junction/`, 2013.

[5] Descarga de arduino ide (en inglés), June 2013.

[6] Lenguaje de programación python (en inglés). `http://www.python.org/`, June 2013.

[7] Página de pyserial (en inglés). `http://pyserial.sourceforge.net/`, 2013.

[8] Ubuntu, página principal (en inglés). `http://www.ubuntu.com/`, 2013.

[9] Julian Dunn. Jitter. specification and assessment in digital audio equipment. *AES 93rd Convention, October 1992*, 1992.

[10] Introduction to watchdog timers (en inglés). `http://www.embedded.com/electronics-blogs/beginner-s-corner/4023849/Introduction-to-Watchdog-Timers`, 2013.

[11] Página en googlecode deentropy (en inglés). `http://code.google.com/p/avr-hardware-random-number-generation/wiki/WikiAVRentropy`, 2013.

[12] openssh, página principal (en inglés). `http://www.openssh.org/`, 2013.

[13] Entropy tests results (en inglés). `http://code.google.com/p/avr-hardware-random-number-generation/wiki/WikiAVRentropy#Test_Results`, 2013.

[14] Diseño completamente aleatorio. `http://www.uru.edu/fondoeditorial/libros/pdf/manualdestatistix/cap2.pdf`, 2013.

[15] Proyecto global consciousness de princeton (en inglés). `http://noosphere.princeton.edu/`, 2013.

[16] Haskell, página principal (en inglés). `http://www.haskell.org/haskellwiki/Haskell`, 2013.

[17] Haskell: Serial port (en inglés). `http://hackage.haskell.org/package/serialport`, 2013.

[18] Faraday cage (en inglés). `http://www.juliantrubin.com/bigten/faradaycageexperiments.html`, 2013.

6. Apéndice: Código Arduino del Prototipo 0

A continuación se detalla el código Arduino utilizado en el primer prototipo diseñado. El mismo se encuentra disponible en la página de Rob Seward[3].

```
/*******************************/
/*  Rob Seward 2008-2009       */
/*  v1.0                       */
/*  4/20/2009                  */
/*******************************/

#define BINS_SIZE 256
#define CALIBRATION_SIZE 50000
#define NO_BIAS_REMOVAL 0
#define EXCLUSIVE_OR 1
#define VON_NEUMANN 2
#define ASCII_BYTE 0
#define BINARY 1
#define ASCII_BOOL 2
/***  Configure the RNG **************/
int bias_removal = NO_BIAS_REMOVAL;
int output_format = BINARY;
int baud_rate = 19200;
/*********************************/

unsigned int bins[BINS_SIZE];
int adc_pin = 0;
int led_pin = 13;
boolean initializing = true;
unsigned int calibration_counter = 0;

void setup(){
 pinMode(led_pin, OUTPUT);
 Serial.begin(baud_rate);
 for (int i=0; i < BINS_SIZE; i++){
   bins[i] = 0;
 }
}
void loop(){
 byte threshold;
 int adc_value = analogRead(adc_pin);
 byte adc_byte = adc_value >> 2;
 if(calibration_counter >= CALIBRATION_SIZE){
   threshold = findThreshold();
   initializing = false;
 }
 if(initializing){
   calibrate(adc_byte);
   calibration_counter++;
 } else{
   processInput(adc_byte, threshold);
 }
}
void processInput(byte adc_byte,
                  byte threshold){
 boolean input_bool;
 input_bool = (adc_byte < threshold) ? 1 : 0;
 switch(bias_removal){
   case VON_NEUMANN:
     vonNeumann(input_bool);
```

```
      break;
    case EXCLUSIVE_OR:
      exclusiveOr(input_bool);
      break;
    case NO_BIAS_REMOVAL:
      buildByte(input_bool);
      break;
  }
}
void exclusiveOr(byte input){
 static boolean flip_flop = 0;
 flip_flop = !flip_flop;
 buildByte(flip_flop ^ input);
}
void vonNeumann(byte input){
 static int count = 1;
 static boolean previous = 0;
 static boolean flip_flop = 0;

 flip_flop = !flip_flop;
 if(flip_flop){
   if(input == 1 && previous == 0){
     buildByte(0);
   }
   else if (input == 0 && previous == 1){
     buildByte(1);
   }
 }
 previous = input;
}
void buildByte(boolean input){
 static int byte_counter = 0;
 static byte out = 0;
 if (input == 1){
   out = (out << 1) | 0x01;
 }
 else{
   out = (out << 1);
 }
 byte_counter++;
 byte_counter %= 8;
 if(byte_counter == 0){
   if (output_format == ASCII_BYTE)
   Serial.println(out, DEC);
   if (output_format == BINARY)
   Serial.print(out, BIN);
   out = 0;
 }
 if (output_format == ASCII_BOOL)
 Serial.print(input, DEC);
}

void calibrate(byte adc_byte){
 bins[adc_byte]++;
 printStatus();
}
unsigned int findThreshold(){
```

```
 unsigned long half;
 unsigned long total = 0;
 int i;
 for(i=0; i < BINS_SIZE; i++){
   total += bins[i];
 }
 half = total >> 1;
 total = 0;
 for(i=0; i < BINS_SIZE; i++){
   total += bins[i];
   if(total > half){
     break;
   }
 }
 return i;
}

//Blinks an LED after each
//10th of the calibration completes

void printStatus(){
 unsigned int increment =
        CALIBRATION_SIZE / 10;

 static unsigned int num_increments = 0;
 //progress units so far

 unsigned int threshold;

 threshold = (num_increments + 1) * increment;

 if(calibration_counter > threshold){

   num_increments++;

   Serial.print("*");

   blinkLed();
 }
}
void blinkLed(){
 digitalWrite(led_pin, HIGH);
 delay(30);
 digitalWrite(led_pin, LOW);
}
```