UNIVERSITAT ROVIRA I VIRGILI (URV) Y UNIVERSITAT OBERTA DE CATALUNYA (UOC)

MÁSTER UNIVERSITARIO EN INGENIERÍA COMPUTACIONAL Y MATEMÁTICA

TRABAJO FINAL DE MÁSTER

ÁREA: CRIPTOGRAFÍA

Generación de números aleatorios con SPADs en condiciones de oscuridad

Autor: Javier Piay Pombo

Tutor: Oriol Farràs Ventura

Fecha de Entrega: julio 2023

El /a Dr./Dra. (nombre) Oriol Farràs Ventura....., certifica que el/la estudiante (nombre) Javier Piay Pombo..... ha elaborado el trabajo bajo su tutoría y autoriza la presentación de esta memoria para la su evaluación.

Firma del director/a:

FARRAS VENTURA ORIOL - 79271456R Digitally signed by FARRAS VENTURA ORIOL - 79271456R Date: 2023.07.04 19:42:04 +02'00'

Ficha del TFM

Título:	Generación de números aleatorios con SPADs en condiciones de oscuridad.
Nombre del autor:	Javier Piay Pombo
Nombre del tutor/a:	Oriol Farràs Ventura
Fecha (mm/aaaa)**:**	07/2023
Titulación:	Máster en Ingeniería Computacional y Matemática
Área:	Criptografía
Idioma:	Castellano
Palabras clave:	Criptografía, TRNG, SPAD

La generación de números aleatorios es demasiado importante como para dejarlo al azar.
- Robert R. Coveyou

La utilización de métodos aritméticos para generar números aleatorios es un pecado.
- John von Neumann

La información es el valor inverso opuesto de la probabilidad.
- Claude Shannon

La probabilidad de un suceso es la razón que tenemos para creer que ha tenido lugar, o lo tendrá.
- Siméon-Denis Poisson

A mi tutor de TFM, el profesor Oriol Farràs Ventura, en agradecimiento por la oportunidad ofrecida para realizar este trabajo, así como por su dedicación y disposición para realizar las tareas de orientación y revisión del mismo.

Resumen

Este trabajo tiene por objeto analizar la posibilidad de generar números aleatorios a nivel hardware utilizando dispositivos SPAD en condiciones de oscuridad. El alcance del trabajo es principalmente teórico y la motivación del mismo es explorar las distintas técnicas disponibles que permiten extraer secuencias aleatorias a partir de SPADs. El trabajo se estructura en dos partes bien diferenciadas. En la primera parte se introducen los conceptos de aleatoriedad y entropía y se describen diferentes técnicas de extracción de la misma. En esta primera parte también se describen y clasifican los diferentes tipos de generadores de números aleatorios.
La segunda parte del trabajo está dedicada al estudio estadístico y de diferentes estrategias de digitalización de la aleatoriedad generada por un SPAD en condiciones de oscuridad. Como parte de este estudio se presentan dos herramientas software desarrolladas en el marco de este trabajo para permitir el modelado, la representación, la visualización y la digitalización de series sintéticas de pulsos de avalancha originados en el SPAD.

Abstract

The aim of this work is to assess the possibility of generating random numbers at hardware level using SPADs working under dark conditions. The scope of this work is mainly theoretical, and its motivation is to explore the different available techniques that allow the extraction of random sequences based on SPADs. This work is structured in two well differentiated parts. In the first part, concepts related to randomness and entropy are introduced and different techniques of entropy extraction are described. Different types and classification of random number generators are also described in this first part. The second part of this work is devoted to the study of the statistics and different digitalization strategies of the randomness generated by a SPAD under dark conditions. As part of this study, two software tools that have been developed within the framework of this work to allow the modeling, representation and digitalization of synthetic series of avalanche pulses originated in the SPAD are presented.

Índice

Parte I

1. Introducción

En el área de las ciencias de la computación se puede definir la aleatoriedad de un suceso como una función del observador que caracteriza su capacidad para predecir un valor asociado a dicho suceso.

Hasta tal punto esto es así que, en base al principio de causalidad, se podría llegar a afirmar que la aleatoriedad de un suceso no es una propiedad intrínseca del sistema generador de dicho suceso, sino un reflejo del desconocimiento, por parte del observador, de los principios y mecanismos que rigen el comportamiento de dicho sistema.

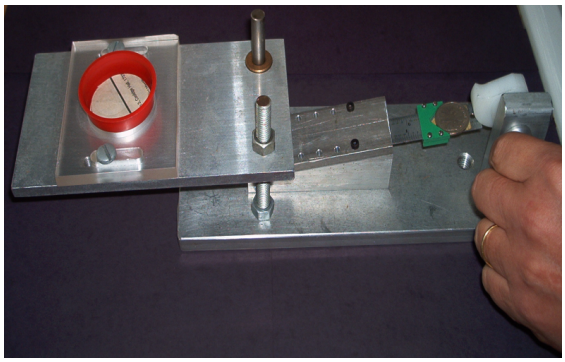


Figura 1.1. Máquina de lanzamiento de moneda utilizada para analizar el comportamiento dinámico del fenómeno e intentar reproducir siempre el mismo resultado: que la moneda caiga mostrando el mismo lado que mostraba antes de lanzarla. **[1]**

La aleatoriedad, entendida entonces como el desconocimiento del observador que dificulta o imposibilita su capacidad de predicción, juega un papel esencial en el campo de la criptografía, ya que ésta se basa en asegurar el desconocimiento por parte de un observador malintencionado (adversario) para limitar, cuando no eliminar, su capacidad para predecir, inferir o generar

resultados en los procesos de ocultación y revelación de la información, aún en el caso de que dichos mecanismos de ocultación y revelación sean conocidos por el observador.

La generación de números aleatorios es una parte fundamental de los algoritmos y primitivas utilizados en los sistemas criptográficos modernos, cuya seguridad depende en gran medida de la calidad o grado de impredecibilidad de dichos números. La utilización de números aleatorios también resulta de gran interés y necesidad en muchas aplicaciones no criptográficas, como juegos, muestreo de datos o simulaciones, con requisitos de impredecibilidad menos exigentes que en el caso anterior.

Ejemplos de casos de uso de números aleatorios en criptografía:

- Claves de sesión y mensaje en sistemas de cifrado.
- Valores iniciales para algoritmos (de generación de números primos, por ejemplo).
- Modificación de contraseñas.
- Vectores de inicialización en cifrado por bloques.
- Números de un único uso en protocolos de comunicación.
- Generación de parámetros y claves en sistema de firma digital.

2. Generadores de números aleatorios

En función del modo de generación empleado, los generadores de números aleatorios se pueden clasificar en realmente aleatorios y pseudo-aleatorios.

Los generadores **realmente aleatorios** son aquellos sistemas no deterministas donde, en teoría, no sería de aplicación el principio de causalidad, es decir, en sistemas en los que el comportamiento es totalmente impredecible en cuanto a los resultados producidos o, dicho de otra manera, en sistemas donde las salidas no se pueden aproximar en función de las entradas y/o el estado del propio sistema.

En la práctica, el nivel de exigencia en cuanto a la definición de generadores realmente aleatorios es más relajado, y el requisito de invalidez o inexistencia del principio de causalidad suele sustituirse por el desconocimiento, en mayor o menor medida, de dicho principio.

Estos generadores se designan comúnmente por sus siglas en inglés **TRNG** y requieren de sistemas físicos o hardware dedicados a las funciones de generación de valores impredecibles, por lo que también se denominan **HRNG**. Los principales inconvenientes de este tipo de generadores son la necesidad de postprocesado de la aleatoriedad (tal y como se verá más adelante) y que su tasa y/o el volumen de generación pueden no ser suficientes para satisfacer los requisitos de la aplicación en cuestión.

Una solución alternativa consiste en utilizar sistemas mixtos de generación de números aleatorios, donde a la parte no determinista implementada vía hardware se le añade otra parte determinista implementada vía software, cuya finalidad es el procesado de una secuencia corta inicial de números impredecibles (denominada semilla) y su posterior expansión, mediante algoritmos deterministas, hasta alcanzar la tasa y/o el volumen necesario para la aplicación en cuestión. Estos generadores se denominan pseudo-aleatorios y se designan comúnmente por sus siglas en inglés **PRNG**.

La validez o idoneidad de los generadores pseudo-aleatorios se puede extender incluso, bajo determinadas condiciones, a aplicaciones criptográficas, dando lugar a los denominados generadores pseudo-aleatorios criptográficamente seguros, designados comúnmente por sus

siglas en inglés **CSPRNG**. En estos casos, los números generados no sólo deben parecer aleatorios (al igual que los PRNG), satisfaciendo pruebas de aleatoriedad estadística (distribución uniforme y sin correlaciones), sino que también deben resultar impredecibles, en el sentido de que ningún algoritmo probabilístico de tiempo polinomial (PPT) debería ser capaz de distinguir entre la secuencia generada por un CSPRNG y la generada por un TRNG.

Llegados a este punto, conviene señalar que, desde el punto de vista del principio de funcionamiento, la principal diferencia entre los generadores realmente aleatorios y los generadores pseudo-aleatorios es que, en lo primeros, no hay expansión de la secuencia aleatoria, es decir, no se generan números aleatorios a partir de los generados previamente (o de la semilla inicial), sino que se procesa la propia secuencia generada por la fuente para intentar extraer la máxima aleatoriedad disponible (o entropía, más propiamente dicho), eliminado las posibles imperfecciones, como sesgos y correlaciones, inherentes al proceso de generación.

Los tres tipos de generadores de números aleatorios descritos anteriormente hacen uso de fuentes de aleatoriedad no deterministas, ya sea para la generación de la propia secuencia de números aleatorios o para la generación de la semilla.

En el caso de los PRNG, se suelen utilizar mecanismos de aleatoriedad sencillos, que no ocurren de forma “natural”, como los basados en el uso del teclado y el ratón, valores de registros y contadores internos, tráfico de red, acceso a memoria, etc. Todos estos mecanismos tienen en común que, aun estando basados en hardware, no requieren un hardware específico para la generación de la semilla.

Los generadores CSPRNG y TRNG, como se ha visto, tienen requisitos de aleatoriedad más exigentes que los PRNG, de modo que se basan en fenómenos físicos, tanto clásicos como cuánticos, donde dicha aleatoriedad sucede de forma natural y, si cabe, más impredecible, como los basados en ruido eléctrico producido en resistencias y diodos Zener, fluctuaciones en circuitos oscilantes, metaestabilidad o desintegración radiactiva, entre otros. A diferencia de los PRNG, estos generadores precisan hardware dedicado a la generación, acondicionamiento y digitalización de las señales aleatorias.

Figura 2.1. Dispositivo TRNG basado en el efecto avalancha en semiconductores, con una tasa de generación de 400 Kbits/s. **[2]**

Los fenómenos físicos clásicos, por complejos que sean o parezcan, son fundamentalmente deterministas, de modo que su evolución a partir de unas condiciones iniciales conocidas sería totalmente predecible. Se podría afirmar, entonces, que la evolución de dichos fenómenos no es aleatoria sino (muy) compleja y, por consiguiente, impredecible en la medida que la insuficiente capacidad computacional así lo permita. Asimismo, dicha complejidad dificulta modelar el comportamiento del generador de números aleatorios correspondiente, y podría resultar imposible verificar su correcto funcionamiento, o incluso asegurar que la aleatoriedad generada no está siendo manipulada de forma malintencionada.

Los problemas señalados anteriormente no tienen lugar en los generadores TRNG basados en fenómenos físicos cuánticos, o generadores **cuánticos** de números aleatorios, designados comúnmente por sus siglas en inglés **QRNG**, ya que dichos fenómenos son fundamentalmente aleatorios, la aleatoriedad se puede generar mediante procesos relativamente sencillos, el generador se puede modelar y su funcionamiento se puede llegar a monitorizar, validar, verificar e incluso certificar. Los QRNG, aun siendo conocidos desde hace ya un par de décadas, han ganado popularidad en los últimos años, debido a su irrupción comercial en forma de chips integrables en otros dispositivos y en un momento de creciente preocupación por cuestiones relativas a la seguridad y la privacidad en las comunicaciones.

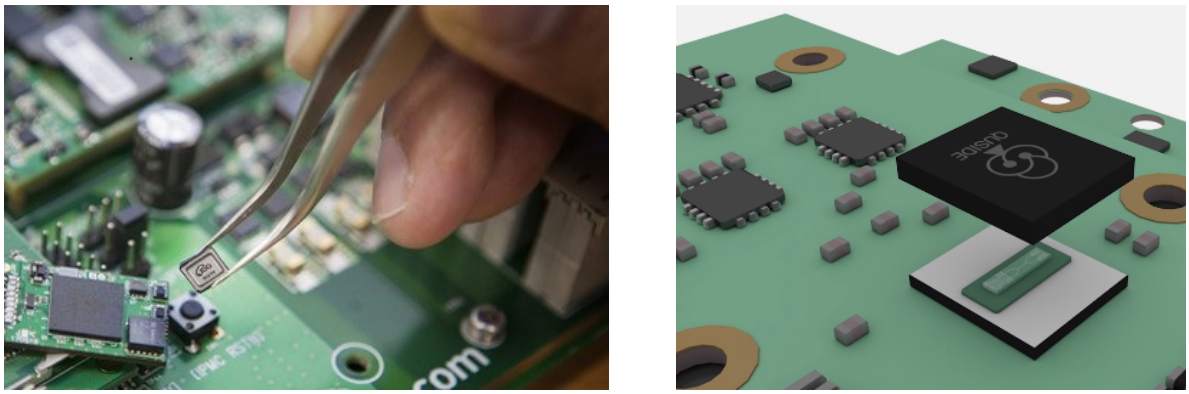


Figura 2.2. Chips QRNG para integración en dispositivos electrónicos. Izquierda **[3]**: tasa de generación de 20 Mb/s. Derecha **[4]**: tasa de generación de 400 Mb/s.

Los generadores QRNG se pueden clasificar en dos grupos principales:

Dependientes del dispositivo: son los más comunes y se basan en el principio de superposición cuántica, en virtud del cual, dos estados físicos perfectamente distinguibles (desde el punto de vista clásico) pueden coexistir con idéntica probabilidad en el llamado estado cuántico superpuesto. La medida de este estado es lo que finalmente determinará en cuál de los dos estados clásicos se encuentra el sistema, cuyo valor se puede considerar intrínsecamente aleatorio. Si bien los procesos de preparación y medida del estado cuántico superpuesto son relativamente sencillos, la operación del generador depende del dispositivo, en la medida que dicha operación se basa en la confianza en que procesos anteriores se implementen de forma correcta.

Independientes del dispositivo: este tipo de generadores no se basa en la confianza en la correcta realización física del dispositivo, sino en la posibilidad de certificar la naturaleza cuántica de su funcionamiento, mediante ensayos (como el de Bell) que permitan asegurar que dicho funcionamiento sólo es posible si tienen lugar fenómenos de naturaleza “no clásica”, como el entrelazamiento.

Atendiendo al fenómeno cuántico en el que basan su funcionamiento, los generadores QRNG se pueden clasificar en los siguientes tipos:

- **Desintegración radiactiva:** fueron los primeros en desarrollarse y sus principales inconvenientes son la necesidad de una fuente radiactiva y su baja tasa de generación, del orden de cientos de kilobits por segundo.

- **Ópticos (o fotónicos):** son el tipo más habitual y en mayor desarrollo. Su tasa de generación es del orden de los megabits o incluso decenas de gigabits por segundo. Estos generadores se clasifican, a su vez, en dos grupos:

    - **De variable discreta:** su funcionamiento se basa en esquemas de distribución espacial o temporal de fotones individuales. Entre los primeros cabe señalar los basados en **divisor de haz de fotones** y, entre los segundos, los basados en el **tiempo de llegada** o el **conteo de fotones**.

    - **De variable continua:** su funcionamiento se basa en el muestreo de sistemas con dinámicas observables a nivel macroscópico. En este grupo cabe señalar los basados en **fluctuaciones del vacío**, en **ruido de fase**, o en **emisión espontánea** amplificada, entre otros.

- **Electrónicos:** su fuente de aleatoriedad no está tan claramente definida como en los tipos anteriores, pero con una implementación adecuada se puede conseguir un régimen de funcionamiento cuántico. En este tipo se incluyen, entre otros, los basados en el **efecto túnel** producido en diodos Zener o en uniones p-n en transistores MOS.

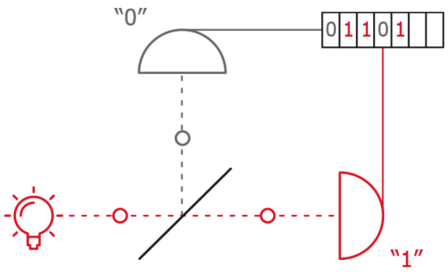


Figura 2.3. Esquema de funcionamiento de un QRNG óptico basado en la división de un haz de fotones por medio de un espejo semitransparente. La transmisión o reflexión de los fotones es un fenómeno intrínsecamente aleatorio que no puede verse influenciado por factores externos. **[5]**

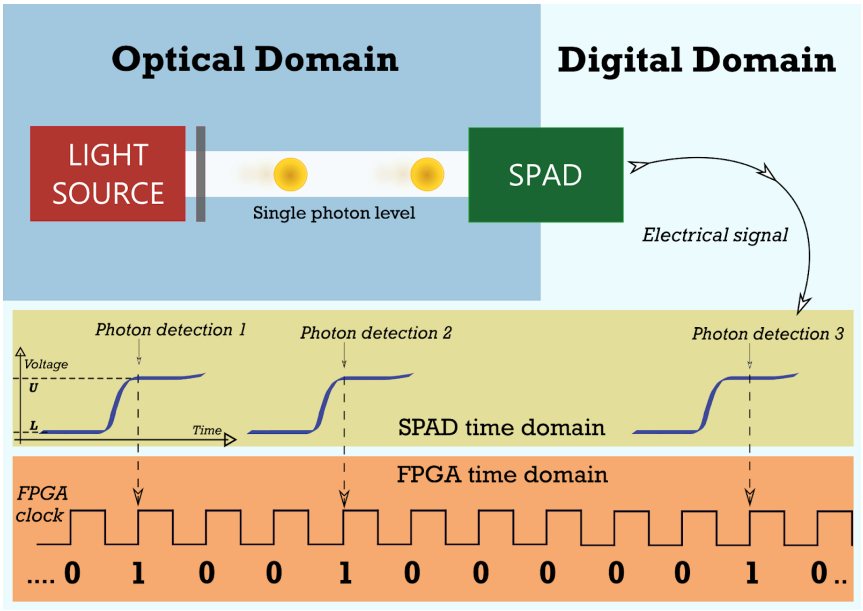


Figura 2.4. Esquema de funcionamiento de un QRNG óptico basado en el tiempo de llegada de fotones individuales detectados por un diodo de avalancha (SPAD). El fotodetector envía las señales eléctricas a un dispositivo FPGA. El dispositivo muestrea las señales utilizando su reloj interno y genera un '1' o un '0' en función de que se detecte, o no, la llegada de un nuevo fotón. **[6]**

3. Aleatoriedad y entropía

Idealmente, las fuentes de aleatoriedad deberían producir secuencias de números indistinguibles de una distribución normal, de modo que dichos números deberían ser independientes entre sí y equiprobables, dando lugar a lo que se conoce como fuentes de aleatoriedad fuerte. Sin embargo, la realización práctica y el funcionamiento de estas fuentes contienen imperfecciones que se manifiestan en forma de sesgos y correlaciones indeseadas en las secuencias generadas, resultando así menos impredecibles de lo deseable y dando lugar a lo que se conoce como fuentes de aleatoriedad débil.

NOTA: en lo que sigue, sin pérdida de generalidad, nos referiremos a fuentes generadoras de bits aleatorios, cuyos únicos valores posibles son '0' y '1'.

Una forma de medir el grado de impredecibilidad de un proceso estocástico es utilizando el concepto de **entropía Shannon**, quien en su día definió la entropía de una variable aleatoria discreta X como una medida matemática de la cantidad de información obtenida de la observación de X, de modo que dicha medida siempre es relativa al observador y su conocimiento previo a la observación.

La entropía Shannon se expresa en bits y se puede definir formalmente mediante la **Ecuación 3.1**, en la que X representa una variable aleatoria discreta, P_i la probabilidad de que X adopte uno de sus valores posibles, e i recorre todas los valores posibles de X.

Ecuación 3.1:

$$H(X) = \sum_{i} -P_i \times log_2 P_i$$

A continuación se muestran un par de ejemplos para ayudar a comprender el concepto de entropía y su relación con la cantidad de información.

Ejemplo 3.1:

Supongamos, en primer lugar, un experimento en el que se genera una secuencia con los resultados de lanzar 100 monedas al aire de modo que todos los resultados posibles son equiprobables. La secuencia resultante se puede representar como $\{0,1\}^{100}$ y el número de secuencias posibles es 2^{100}, de modo que la probabilidad de aparición de cada una de las secuencias posibles es igual a $1/2^{100}$ o 2^{-100}, resultando el siguiente valor de la entropía Shannon para la secuencia en cuestión:

$$H(X) = -\sum_{1}^{2^{100}} (2^{-100} \times log_2 2^{-100}) = -\sum_{1}^{2^{100}} (2^{-100} \times -100) = 100\ bits$$

Este valor indica que se necesitan exactamente 100 bits para representar la secuencia generada, de modo que la entropía por bit, o tasa de entropía, es exactamente 1.

Cuando la entropía de una secuencia es igual al número de bits de la misma, la secuencia se denomina de **entropía completa** y se considera una secuencia de aleatoriedad ideal que se puede utilizar con cualquier propósito criptográfico.

NOTA: en la práctica, el requisito para definir una secuencia aleatoria como de entropía completa es algo menos exigente, de modo que es suficiente con que la entropía por bit sea al menos igual a $1 - \varepsilon$, siendo ε un valor arbitrariamente pequeño, denominado grado de incertidumbre. En la serie de publicaciones **NIST SP 800-90**, sobre recomendaciones para la construcción y validación de generadores de bits aleatorios, se adopta 2^{-64} como valor límite superior para esta incertidumbre.

Ejemplo 3.2

Supongamos que se generan claves criptográficas de 256 bits de modo que la clave con todo ceros aparece el 50% de las veces y el resto de las claves lo hacen de forma equiprobable. En este caso se identifican dos únicos valores de probabilidad, uno para la clave con todo ceros, que llamaremos $P1$, y otro para el resto de claves, que llamaremos $P2$. Resulta evidente que $P1$ debe ser igual a $1/2$ (ya que la clave aparece una de cada dos veces) y $P2$ se puede calcular

como la probabilidad de que la clave resultante sea distinta a la de todo ceros y, además, tenga alguno de los valores posibles, de modo que:

$P2 = 1/2 \times 1/(2^{256} - 1)$ (ya que hay $2^{256} - 1$ claves que no son todo ceros)

En este caso, el valor de la entropía Shannon correspondiente a cualquiera de las claves de 256 bits tiene el siguiente valor:

$$H(X) = -0.5\ log_2 0.5\ - (2^{256} - 1)\ 0.5 \times 1/(2^{256} - 1)\ log_2(0.5 \times 1/(2^{256} - 1))\ = 128\ bits$$

Este resultado se podría interpretar como que las claves generadas sólo tienen 128 bits realmente aleatorios que, aun siendo la mitad de los 256 bits originales, podrían resultar suficientes para ciertas aplicaciones. Pero lo cierto es que la clave con todo ceros se repite cada dos veces y un adversario podría adivinarla al primer intento.

Por este motivo, y a efectos de evaluación de la calidad de la aleatoriedad, es preferible hacer uso de la probabilidad máxima con la que un adversario podría predecir la secuencia generada, dando lugar al concepto de **entropía mínima**, expresada también en bits y definida mediante la **Ecuación 3.2**.

Ecuación 3.2:

$$H_\infty(X) = -log_2 max(P[X = v])$$

Así pues, la entropía mínima correspondiente al caso anterior sería $-log_2 0.5 = 1\ bit$

De modo que la entropía de la fuente generadora de claves pasaría de 128 bits a tan sólo 1 bit de aleatoriedad, lo que supondría una reducción drástica de la misma y cuyo valor final (mucho más conservador) debería utilizarse a efectos de caracterización de dicha fuente.

De la **Ecuación 3.2** se deduce que una fuente generadora de aleatoriedad tiene una entropía mínima de valor k si y sólo si para cualquier distribución X en $\{0,1\}^n$ y cualquier $v\ \epsilon\ X$ se cumple que $P[X = v] \leq 2^{-k}$, tal y como se demuestra a continuación:

Demostración 3.1:

$$H_\infty(X) = -log_2 P_{max} = k \Rightarrow P_{max} = 2^{-k} \Rightarrow P \leq 2^{-k}$$

O dicho de otra forma, una secuencia de n bits tiene una entropía mínima de valor k si una distribución en $log_2 n$ bits contine k bits aleatorios. La fuente de aleatoriedad correspondiente se denomina de tipo (n, k), siendo el cociente k/n la tasa de entropía mínima de la fuente.

4. Extracción de entropía

La extracción de entropía tiene por objeto convertir una secuencia de aleatoriedad débil en otra de aleatoriedad fuerte, tratando de mantener la aleatoriedad introducida originalmente por la fuente generadora, pero reduciendo al máximo, o incluso eliminando, las posibles imperfecciones introducidas en el proceso de generación. Este proceso de extracción también se conoce por el nombre de postprocesado, acondicionamiento o blanqueo de los datos generados (o crudos).

4.1 Extractores deterministas

NOTA: se entiende por clase de una fuente de aleatoriedad, los conjuntos de dicha fuente en los que los bits generados tienen una estructura particular y común a todos los conjuntos.

Sea C una clase de fuente de aleatoriedad en $\{0,1\}^n$. Un extractor determinista de tipo ε para la clase C se puede definir formalmente como una función $E : \{0,1\}^n \rightarrow \{0,1\}^m$ tal que para todo $X \in C$ se cumple que la distribución $E(X)$ resulta indistinguible de una distribución uniforme U_m en $\{0,1\}^m$ o, dicho de otro modo, la distancia estadística entre ambas distribuciones es inferior a ε (arbitrariamente pequeño), tal como se representa en la **Ecuación 4.1**.

Ecuación 4.1:

$$1/2 \sum_{y=0}^{2^m-1} |P[E(X) = y] - P[U_m = y]| \leq \varepsilon$$

O, de forma equivalente, en la **Ecuación 4.2**.

Ecuación 4.2:

$$\sum_{y \; tal \; que \; P[E(X)=y] \geq P[U_m=y]} P[E(X) = y] - P[U_m = y] \leq \varepsilon$$

Para poder extraer m bits de aleatoriedad es necesario que $H_\infty(X) \geq m$, tal y como se demuestra a continuación.

Supongamos que el extractor extrae m bits de aleatoriedad. Si la entropía mínima de la fuente no cumple la condición anterior, esto significa que $H_\infty(X) \leq m - 1$, de modo que existe al menos un elemento $x \, \epsilon \, X$ tal que $P[X = x] \; = 2^{-(m-1)}$ y el elemento $y = E(x)$ de salida correspondiente del extractor también tendría esa probabilidad, por lo que la salida del extractor ya no podría ser uniformemente distribuida, de lo que se deduce que la condición anterior debe cumplirse.

El cociente n/m se denomina razón, proporción o **relación de extracción**.

Se muestran, a continuación, algunos ejemplos de extractores deterministas que son de aplicación a ciertas clases de fuentes de aleatoriedad. Estos extractores también se denominan **ad-hoc** (por ser de aplicación específica) y **no-criptográficos** (debido a que su principio de funcionamiento no se basa en primitivas criptográficas).

4.1.1 Extractor tipo von-Neumann

En la **Figura 4.1** se muestra el esquema de funcionamiento de un extractor de tipo **von-Neumann**, aplicable a fuentes de aleatoriedad en las que los bits generados son independientes e idénticamente distribuidos (IID), pero no de forma uniforme, es decir: $P[X_i = 1] = p \neq 1/2$.

En este tipo de extractor, las secuencias se procesan en pares de bits consecutivos. Si estos pares están formados por bits diferentes (‘10’ o ‘01’), el bit que se produce es igual al primer bit del par. En caso contrario, no se produce ningún bit. La probabilidad del par ‘10’ es igual a $p \; (1 - p)$ y la del par ‘01’ es igual a $(1 - p) \; p$ (igual que la anterior), de modo que la probabilidad de producir un ‘0’ es igual a la de producir un ‘1’ y el extractor consigue eliminar el sesgo presente en la secuencia original ($p \neq 1/2$), aunque con una tasa de extracción variable.

Este extractor es, por tanto, determinista de tipo 0, ya que la secuencia resultante está uniformemente distribuida ($\varepsilon = 0$). En promedio, el número de bits producidos es igual al número de pares ‘10’ o ‘01’. Como la probabilidad de encontrar alguno de estos pares es $p \; (1 - p) + (1 - p) \; p = 2p \; (1 - p)$, la longitud de la secuencia producida es $2p \; (1 - p) \; n/2$.

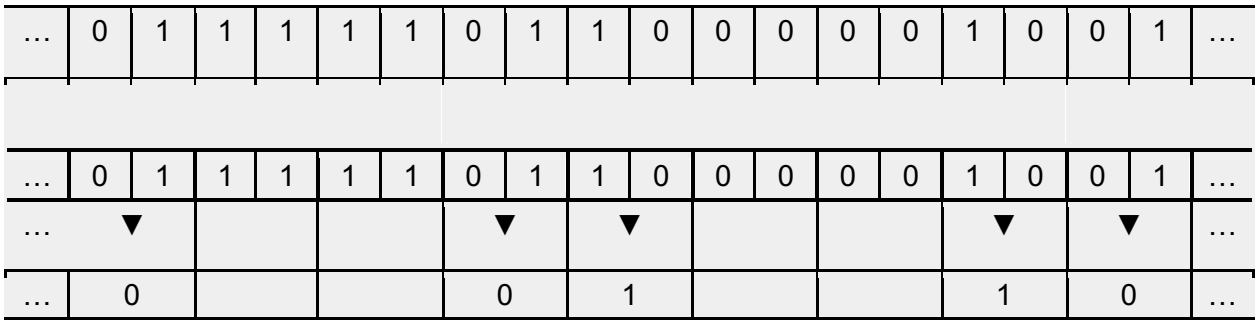


Figura 4.1. Esquema de funcionamiento de un extractor de tipo von-Neumann.

El principal inconveniente de los extractores deterministas es que sólo son de aplicación en las clases de fuentes para las que han sido definidos, de modo que su utilización requiere el conocimiento previo de las mismas y la idoneidad de las secuencias generadas. Supongamos, por ejemplo, el caso de un extractor de tipo **von-Neumann** aplicado a una fuente donde los pares '10' y '01' estén correlacionados de forma muy negativa, es decir, con muchos más pares de un tipo que de otro. En este caso, la aplicación del extractor daría lugar a la extracción de cadenas muy largas de '0' o de '1', produciendo el efecto contrario al inicialmente buscado por el extractor.

4.1.2 Extractor tipo XOR

Otro tipo de fuente de aleatoriedad es aquella en la que cada bit se genera de forma independiente del resto y con distinto sesgo, de modo que $P[X_i = 1] = p_i$ y $0 \leq p_i \leq 1 - \delta$. La extracción de entropía en este tipo de fuentes puede llevarse a cabo mediante un extractor determinista de tipo **XOR**, que sustituye los pares de bits no solapados por el resultado de aplicar este operador lógico a los bits de cada par. Este extractor siempre reduce, aunque no elimina, el sesgo de la secuencia original, y su tasa de extracción es constante (ya que siempre se genera un bit por cada par de bits procesados). Al igual que sucedía en el caso anterior, la aplicación de un extractor de tipo **XOR** en secuencias donde alguno de los valores esté correlacionado de forma muy negativa, carece de sentido (o efectividad).

4.1.3 Extractor tipo Blum

Otro tipo de fuente de aleatoriedad es aquella en la que el valor de cada nuevo bit generado depende de los valores de los bits generados con anterioridad, y de manera tal que la secuencia generada se puede modelar mediante una cadena de Markov con un número finito de estados, tal y como se describe a continuación.

En este modelo, la probabilidad de permanecer en un estado i es P_i y la probabilidad de cambiar de un estado i a otro estado j es$1 - P_i$. La permanencia en el mismo estado implica que el valor del nuevo bit generado sea igual al del bit generado previamente, mientras que el cambio de estado implica que el valor del nuevo bit sea distinto al anterior.

En esta clase de fuente se puede aplicar un extractor determinista de tipo Blum, donde un bit se extrae cuando un estado se visita un número veces que es múltiplo de tres, es decir, por 3ª vez, 6ª vez, etc. siendo el valor del bit extraído igual al resultado de aplicar el método von-Neumann a los valores de los dos últimos estados desde los que se visitó el estado en cuestión, tal y como se muestra en la **Figura 4.2**.

Supongamos, por ejemplo, que una cadena de Markov se representa como (A-B:P), siendo A el estado actual, B el estado siguiente y P la probabilidad de cambio (o permanencia) de A a B, y supongamos que la secuencia de bits mostrada en la **Figura 4.2** se modela mediante la siguiente cadena de Markov:

(0-0:0.4), (0-1:0.6), (1-0:0.1), (1-2:0.9), (2-1:0.3), (2-3:0.7), (3-3:0.2), (3-2:0.8)

El extractor de tipo **Blum** elimina tanto los sesgos como las correlaciones existentes en esta clase de fuentes, por lo que se trata de un extractor determinista de tipo 0 para esta clase en particular.

…	0	0	1	0	1	1	1	1	0	1	1	1	0	1	…
	S0	S0	S1	S0	S1	S2	S1	S2	S3	S2	S1	S2	S3	S2	
				▼			▼			▼				▼	
Par de valores de los dos últimos estados desde los que se accedió al estado en cuestión:															
				(0,1)			(0,1)			(1,0)				(1,0)	
				0			0			1				1	

Figura 4.2. Esquema de funcionamiento de un extractor de tipo Blum.

4.1.4 Extractor para fuentes Santha-Vazirani

Otro tipo de fuente de aleatoriedad es aquella en la que cada bit generado contiene cierta cantidad de aleatoriedad, aunque ésta está condicionada por los bits anteriores, de modo que para todo $1 \leq i \leq n$ y para todo $x_1, \ldots, x_{i-1} \epsilon \{0,1\}$, existe una constante $0 \leq \delta \leq 1/2$ tal que $\delta \leq P[X_i = 1 \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq 1 - \delta$. Este tipo de fuente se denomina **Santha-Vazirani** y se considera una generalización de la clase **von-Neumann**. Cuando la dependencia se extiende a todos los bits restantes, se habla de fuente **Santha-Vazirani fuerte**.
Pues bien, se ha comprobado que no existe ningún extractor determinista que se pueda aplicar a las fuentes **Santha-Vazirani**.

4.1.5 Extractores criptográficos

Surge así la pregunta de si es posible encontrar un extractor de entropía determinista que sea de aplicación general, es decir, válido (o efectivo) para cualquier fuente de aleatoriedad de la que sólo se conozca su entropía mínima (ignorando las posibles estructuras o relaciones entre bits).

La respuesta a la pregunta anterior, a día de hoy, sigue siendo negativa, y esta imposibilidad es lo que históricamente ha motivado la aparición de los llamados extractores de entropía con semilla que se tratarán más adelante.

No obstante la imposibilidad de existencia de extractores deterministas que sean de aplicación general, en la práctica abundan los casos de aplicación de funciones y algoritmos criptográficos a tal efecto, dando lugar a los denominados extractores deterministas **criptográficos**.

En el documento **NIST SP 800-90B [13]** se admite la utilización, a efectos de acondicionamiento de entropía, de funciones hash seguras, como las de las familias **SHA-1**, **SHA-2** y **SHA-3** descritas en los estándares **FIPS 180** y **FIPS 202**, así como la utilización de las funciones auxiliares basadas en hash y cifrado por bloques, descritas en el documento **NIST SP 800-90A [12]**, que forman parte de los generadores pseudo-aleatorios correspondientes. La razón esgrimida para admitir estas funciones criptográficas como extractores de entropía es que dichas funciones pueden distribuir uniformemente la entropía de entrada en la salida.

Dado que la aplicación de las técnicas de extracción de entropía mencionadas anteriormente produce secuencias de bits de longitud fija (128, 256, etc.), cabría preguntarse si existe algún requisito en cuanto a la longitud de las secuencias de entrada correspondientes, al objeto de poder preservar la aleatoriedad existente en las mismas a la salida del extractor. Pues bien, se entiende que a la salida del extractor se dispone de una secuencia de bits con entropía completa si la longitud de dicha secuencia es, como mucho, la mitad de la entropía mínima de la secuencia a la entrada al extractor.

Ejemplo 4.1:
Supongamos que se quieren producir 256 bits de entropía completa utilizando **SHA-256** con una fuente con entropía mínima igual a 0.25 bits/bit. Como la entropía mínima de entrada debe ser 512 bits (256 x 2 = 512), se deben procesar 2048 bits (512 / 0.25 = 2084) procedentes de la fuente para producir los 256 bits con aleatoriedad completa a la salida del extractor.

NOTA: la relación anterior entre la entropía mínima a la entrada y la aleatoriedad completa a la salida de un extractor aparece en el segundo borrador del documento **NIST SP 800-90B [13]**, pero no así en su versión final. De hecho, en el documento **NIST SP 800-90C [14]** sobre la construcción de generadores de bits aleatorios se considera que un bloque de salida de una función hash (o cifrador de bloque) tiene entropía completa si la entropía mínima a la entrada del algoritmo es, al menos, 64 bits mayor que el número de bits que se quieran extraer a la salida. Más concretamente, en el documento **NIST IR 8427 [15]** sobre la definición de entropía completa

en la serie **NIST SP 800-90** se justifica que cuando la entropía mínima de la secuencia de entrada es 64 bits superior a la longitud de la secuencia de salida, la entropía mínima por bit de dicha secuencia de salida es al menos $1 - 2^{-32}$ bits y se considera completa.

Ejemplo 4.2:
Supongamos que se quieren producir 128 bits de entropía completa aplicando el criterio **NIST SP 800-90C [14]** y utilizando una relación de extracción 2:1. La entropía mínima a la entrada debería ser 128 + 64 = 192 bits. Como la relación de extracción es 2:1, el tamaño de la secuencia a la entrada debería ser 128 x 2 = 256 bits, de modo que la entropía mínima a la entrada debería ser, como mínimo, 192 /256 = 0.75 bits/bit.

El documento **NIST SP 800-90B [13]** incluye también un método matemático para calcular la entropía a la salida del extractor, cuya representación gráfica aplicada al caso de **SHA-256** se muestra en la **Figura 4.3**. El documento establece que la longitud de la salida del extractor debe coincidir con la de la salida de la función hash utilizada. Si se aplica este diagrama al caso descrito en el **Ejemplo 4.1**, se observa que sería necesario una relación de extracción igual a 8, de modo que habría que procesar secuencias de 256 x 8 = 2048 bits a la entrada del extractor, lo que coincide exactamente con el cálculo numérico realizado en el ejemplo.

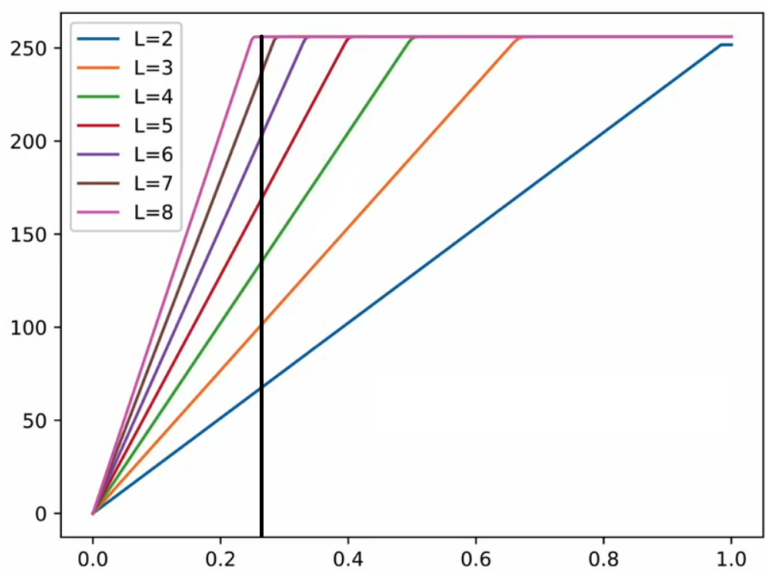


Figura 4.3. Entropía a la salida (eje Y) de un extractor tipo SHA-256 en función de la tasa de entropía a la entrada (eje X) y la relación de extracción (L). La línea vertical corresponde al caso del **Ejemplo 4.1** (tasa de entropía igual a 0.25 bits/bit). Aunque no se muestra en el diagrama, las gráficas tienden asintóticamente al valor de saturación (256) sin llegar a alcanzarlo nunca. **[7]**

4.1.5.1 El extractor Fortuna

La idea de utilizar hashing para acumular, mezclar y extraer entropía de una o varias fuentes de aleatoriedad ha ido evolucionando técnicamente con el paso del tiempo y es la más utilizada en sistemas operativos como Windows, macOS o Linux.

Un ejemplo de extractor de este tipo es el presente en el generador de números pseudo-aleatorios denominado **Fortuna**. En este generador hay 32 acumuladores de entropía entre los que se distribuye cíclicamente la aleatoriedad procedente de varias fuentes (hasta 256). Cada vez que se añade un byte de entropía a un acumulador, éste se concatena con el contenido previo del acumulador y se compacta el acumulador utilizando **SHA-256**, de modo que en cada acumulador siempre hay 32 bytes resultantes de todos los procesos de compactación (actual y previos). Se supone que cada byte que se añade al acumulador tiene una entropía mínima igual a 0.5 bits/bit, de modo que un acumulador se considera que tiene entropía completa cuando se han añadido un total de 64 bytes (en lugar de 32). Cuando el generador de números pseudo-aleatorios necesita nueva entropía, se incrementa un contador y se utiliza su valor para seleccionar los acumuladores que deben proporcionar dicha entropía, de manera que sólo se seleccionan aquellos acumuladores cuyo número (de 1 a 32) cumple que el valor del contador

es múltiplo de dicho número. El contenido de los acumuladores seleccionados se concatena con la última secuencia de entropía extraída y se compacta todo utilizando **SHA-256** para producir la nueva secuencia de entropía "fresca" requerida por **Fortuna**. Cabe señalar que los autores de **Fortuna** proponen utilizar doble hashing $(hasx(hasx(\dots))$, quizás como medida contra posibles ataques por extensión de longitud.

En el caso de Windows 10, la entropía procedente de las fuentes de aleatoriedad se compacta mediante **SHA-512** antes de distribuirse entre los distintos acumuladores de entropía existentes. Linux, por su parte, utiliza **SHA-1** (160 bits) para extraer la entropía de los distintos acumuladores de entropía existentes.

Por último, en este apartado de extractores deterministas cabe mencionar aquellos basados en algoritmos similares a los utilizados en la generación de códigos de redundancia cíclica (**CRC**), con salidas típicas de 16 o 32 bits. Con estos algoritmos, de implementación sencilla y ejecución rápida, se puede realizar un postprocesado o acondicionamiento de la fuente de entropía previo a la entrada a la función hash extractora, reduciendo así su carga computacional, o incluso eliminando la necesidad de su utilización.

4.2 Extractores con semilla

Estos extractores disponen de una entrada adicional de aleatoriedad, considerada fuerte o uniforme a efectos prácticos, llamada semilla, la cual se integra en el proceso general de extracción de entropía de la fuente principal de aleatoriedad débil.

Un extractor con semilla de tipo (k, ε) se puede definir formalmente como una función $E : \{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^m$ tal que para cualquier fuente X de tipo (n, k) en $\{0,1\}^n$ se cumple que la distribución $E(X, Y)$, siendo Y una fuente de aleatoriedad fuerte en $\{0,1\}^s$ e **independiente** de X, resulta indistinguible de una distribución uniforme U_m en $\{0,1\}^m$, en el mismo sentido y dando lugar a ecuaciones similares que en el caso descrito para extractores deterministas.

Además, si se cumple que la concatenación de la salida del extractor y la semilla produce también una distribución casi uniforme en $\{0,1\}^{s+m}$, el extractor se denomina **fuerte**.

El objetivo de un extractor con semilla es utilizar una semilla lo más pequeña posible, ya que la aleatoriedad fuerte es un recurso escaso y difícil de obtener, y maximizar el tamaño de la salida producida, teniendo en cuenta que $m \leq k + s$. La expresión $k + s - m$ se denomina pérdida de entropía del extractor (cuyo valor siempre será positivo).

El cociente n/m se denomina razón, proporción o **relación de extracción** (al igual que en el caso de extractores deterministas).

Los extractores con semilla, al igual que los deterministas, se encuadran en dos grupos principales: no criptográficos y criptográficos. En el primer grupo se encuentran extractores como los basados en funciones **hash universales tipo 2** (como el utilizado por IDQ a nivel de chip) o los extractores tipo **Trevisan** y **Toeplitz**, cuya complejidad y baja tasa de extracción pueden hacer inviable tanto su realización a nivel hardware como su aplicación práctica. En cuanto al segundo grupo, el documento **NIST SP 800-90B [13]** es, de nuevo, la mejor referencia para encontrar las distintas implementaciones de extractores con semilla basados en mecanismos criptográficos. En el mencionado documento se admiten tres algoritmos criptográficos como componentes acondicionadores (o extractores) de entropía:

- **HMAC**: códigos de autenticación de mensajes basados en las mismas funciones hash que las admitidas como extractores deterministas.

- **CMAC**: códigos de autenticación de mensajes basados en bloques de cifrado **AES** (estándar de cifrado avanzado).

- **CBC-MAC**: códigos de autenticación de mensajes basados en encadenamiento de bloques de cifrado **AES**.

NOTA: tanto Intel como AMD utilizan **CBC-MAC-AES-256** para el acondicionamiento de entropía en sus generadores de números pseudo-aleatorios, siendo 256 el número de bits de la clave utilizada para el encriptado mediante **AES**.

En el documento también se menciona que los extractores anteriores deben cumplir los siguientes requisitos en relación a la independencia de la semilla (o clave), tal y como se destacaba en la definición formal:

- Las claves utilizadas pueden ser fijas (con valor predeterminado), establecidas mediante alguna entrada adicional al dispositivo, o generadas mediante las fuentes de aleatoriedad utilizadas. Pero su valor se debe determinar antes de que el extractor produzca ninguna salida.

- Cualquier valor que se utilice para determinar la clave no se podrá usar como entrada del extractor. La entropía de entrada del extractor no podrá incluir ninguna entropía proporcionada a la clave.

El mismo documento establece que la longitud de la salida del extractor debe coincidir, en el caso de **HMAC**, con la de la salida de la función hash utilizada, y en los otros dos casos, con la longitud del bloque **AES**, que es igual a 128.

En la **Figura 4.4** se muestra un diagrama similar al de la **Figura 4.3**, pero correspondiente a un extractor **CBC-MAC-AES**, por lo que el valor máximo de entropía mínima a la salida es 128 bits.

En las **Figuras 4.5** y **4.6** se muestran las construcciones de los extractores **HMAC** y **CBC-MAC**, respectivamente. En estas construcciones, el mensaje se corresponde con los bloques o secuencias de bits de entropía débil generados por la fuente de aleatoriedad, y la clave se corresponde con una secuencia independiente de bits con entropía fuerte.

NOTA: en los chips de Intel, el extractor **CBC-MAC-AES** utiliza una única clave fija obtenida mediante el encriptado (con **AES)** del valor 0, utilizando para ello una clave con valor 1. De esta forma se asegura la independencia entre fuentes, requerida y referida anteriormente. También conviene señalar que en el documento **NIST SP 800-90B [13]** se hace mención explícita a la no aprobación de este método para fines distintos a los de acondicionamiento o extracción de entropía.

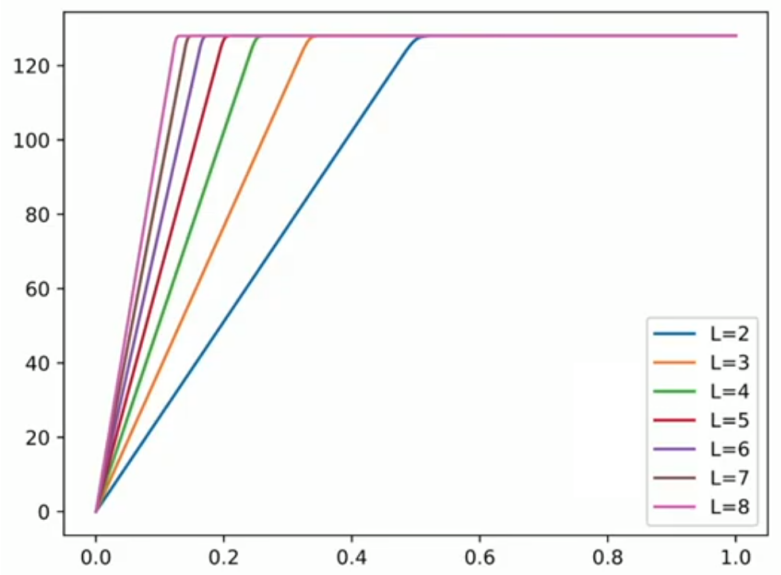


Figura 4.4. Entropía a la salida (eje Y) de un extractor tipo CBC-MAC / AES en función de la tasa de entropía débil a la entrada (eje X) y la relación de extracción (L). **[7]**

Llegados a este punto, conviene hacer hincapié en que un extractor de entropía no debe confundirse con un generador de números pseudo-aleatorios. La misión principal del primero es convertir una fuente de aleatoriedad débil en otra de aleatoriedad fuerte, mientras que el último persigue la ampliación o extensión de la aleatoriedad fuerte ya existente en la semilla.

No obstante lo dicho anteriormente, conviene señalar que los extractores de entropía son una parte fundamental de los generadores de números pseudo-aleatorios, en la medida que la misión de los primeros es, precisamente, generar la semilla a utilizar por los últimos.

También cabe mencionar que un extractor de entropía puede servir perfectamente como generador de números aleatorios, siempre y cuando dicho extractor satisfaga los mismos requisitos de seguridad y tasa de producción que los exigidos a la función de generación.

Por último, conviene recordar que un extractor de entropía no puede realizar su función si no hay entropía extraíble a su entrada. Hasta el punto esto es así, que esta condición resulta, de hecho, un requisito imprescindible, cuyo cumplimiento debe monitorizarse y evaluarse por medio de los correspondientes tests o pruebas de buen funcionamiento de la fuente de entropía.

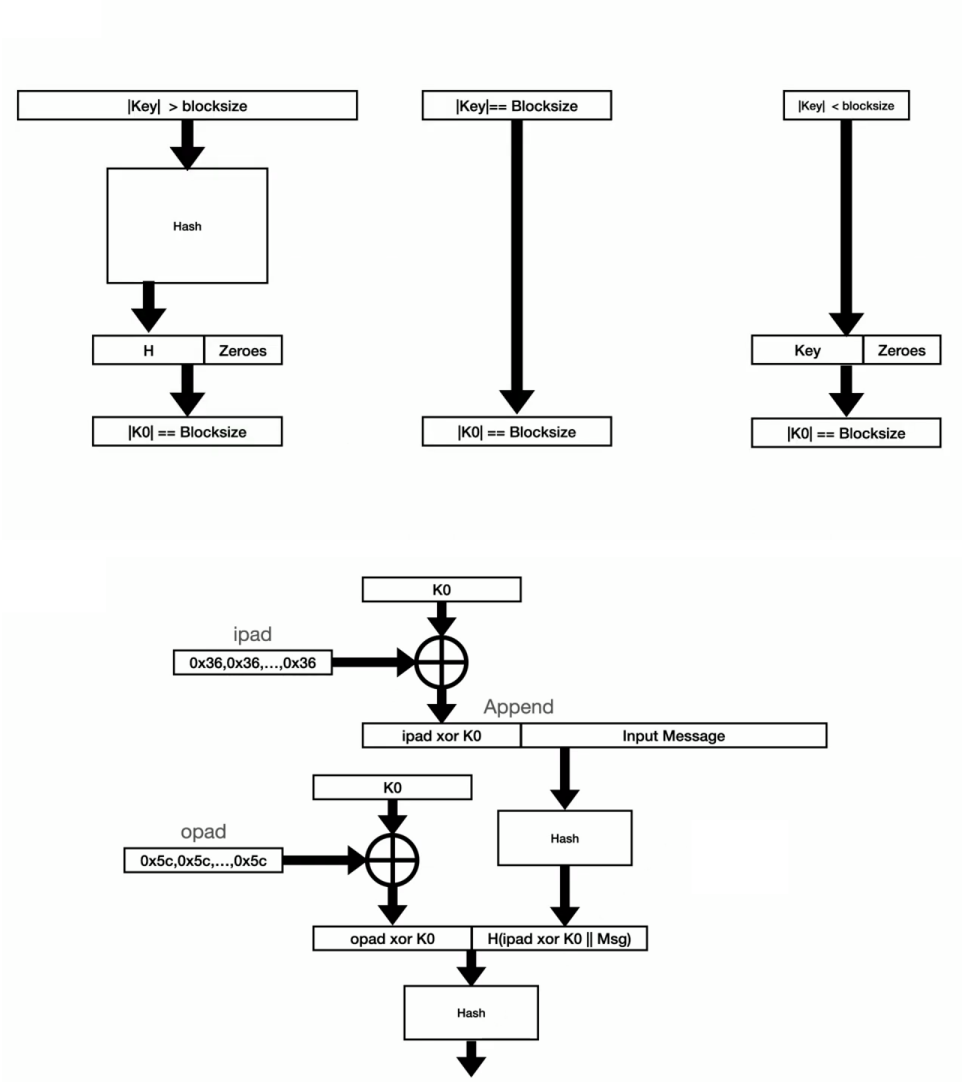


Figura 4.5. Construcción de HMAC **[8]**. Arriba: establecimiento de la clave. Abajo: operación del algoritmo para generar la salida: $HMAC(M) = Hash((K0 \oplus opad) \,||\, Hash((K0 \oplus ipad) \,||\, M))$

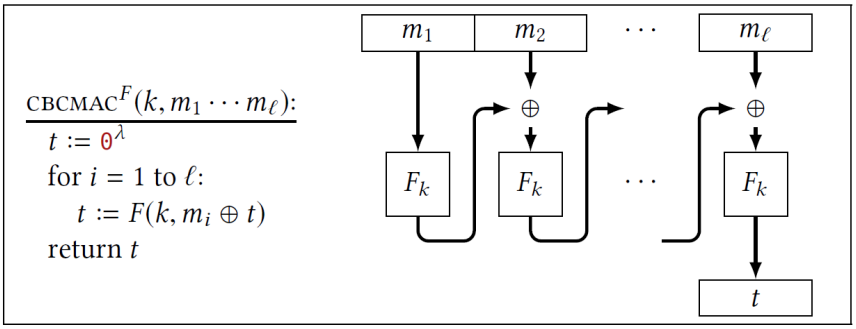


Figura 4.6. Construcción de CBC-MAC. F_k se refiere, en este caso, al cifrado AES mediante clave k. **[9]**

Parte II

5. TRNGs basados en dispositivos SPAD

Existen numerosos mecanismos generadores de aleatoriedad basada en fenómenos físicos, tal y como se describe en el capítulo 2. De entre todos ellos, resultan de especial interés para este trabajo aquellos cuya implementación, control e integración a nivel de dispositivo o chip electrónico, sea factible, sencilla y asequible, evitando la utilización de sistemas y dispositivos costosos, complejos y voluminosos.

En la segunda parte de este trabajo se aborda el estudio de la generación hardware de bits aleatorios basada en el funcionamiento, en condiciones de oscuridad, de los diodos de avalancha para la detección de fotones individuales (**SPAD**).

5.1 El dispositivo SPAD

Un SPAD es un dispositivo electrónico basado en una unión semiconductora polarizada de forma inversa por encima de su tensión de ruptura, en lo que se denomina zona o modo de operación **Geiger**. Su funcionalidad original es la detección de fotones individuales emitidos por una fuente emisora acoplada al dispositivo.

Cuando un fotón con suficiente energía incide en la región sensible (o de multiplicación) de un SPAD, dicho fotón se absorbe generando un par electrón-hueco cuyos portadores de carga se separan rápidamente debido a la sobretensión de polarización inversa a la que está sometida el dispositivo. El electrón, acelerado por el fuerte campo eléctrico existente, choca con otros átomos y genera nuevos pares electrón-hueco por medio del mecanismo de ionización por impacto, originando así nuevos electrones en estado excitado. Este proceso se repite por efecto multiplicativo y da lugar a una **corriente de avalancha** autosostenida.

La corriente de avalancha, mientras se está produciendo, tiene un efecto de auto-extinción de la misma, ya que dicha corriente provoca una disminución de la sobretensión inversa a la que se ve sometido el dispositivo, lo que a su vez provoca la disminución en la aceleración de los portadores de carga y el correspondiente cese del mecanismo de ionización por impacto. Al proceso de extinción de la avalancha le sigue el proceso de recuperación o restablecimiento de las condiciones de sobretensión inversa en el SPAD, lo cual habilita al dispositivo para la generación de un nuevo pulso de corriente de avalancha una vez transcurrido el tiempo de duración de estos procesos, denominado **tiempo muerto** del SPAD.

Un pulso de corriente de avalancha suele ir seguido de **pulsos secundarios** dependientes de la avalancha anterior, cuya aparición puede evitarse si se retrasa el restablecimiento de las condiciones de sobretensión tras la aparición de una avalancha en el SPAD, introduciendo un **tiempo de espera** entre los procesos de extinción y recuperación del dispositivo, lo que se traduce en un mayor tiempo muerto del SPAD.

La generación de pares electrón-hueco que puede dar lugar a la aparición de corrientes de avalancha, además de un origen fotónico, puede tener también un origen térmico, con el SPAD funcionando en **condiciones de oscuridad** (sin fotones incidentes). Si bien estas avalanchas térmicas se podrían considerar como una fuente de ruido no deseable en el caso del SPAD funcionando como “detector” de la aleatoriedad externa generada por una fuente emisora de fotones acoplada al dispositivo, dicho ruido se puede considerar como una fuente de generación de **aleatoriedad interna** del SPAD, cuyo funcionamiento en estas condiciones de oscuridad resulta de especial interés y aplicación, ya que la realización física y el control de la operación de dicho generador se simplifica en gran medida al eliminar la necesidad de disponer de una fuente emisora de fotones y los elementos ópticos de acoplamiento correspondientes.

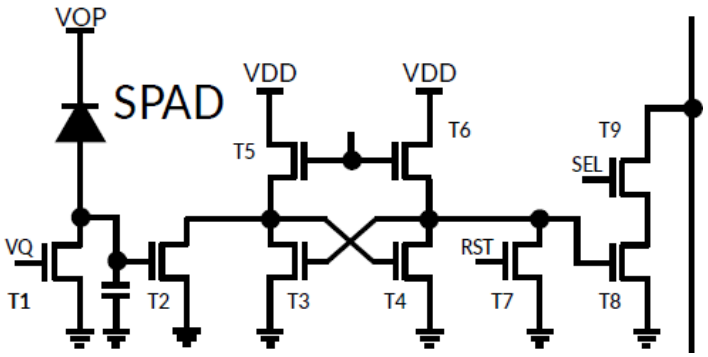


Figura 5.1. Ejemplo de esquema de integración de un SPAD y su circuitería auxiliar a nivel de chip electrónico. Función de los transistores: T1 para eliminación de la corriente de avalancha, T2 para escritura del bit en la memoria formada por T3, T4, T5 y T6, T7 para resetear la memoria, y T8 y T9 para lectura del bit almacenado en la memoria. **[10]**

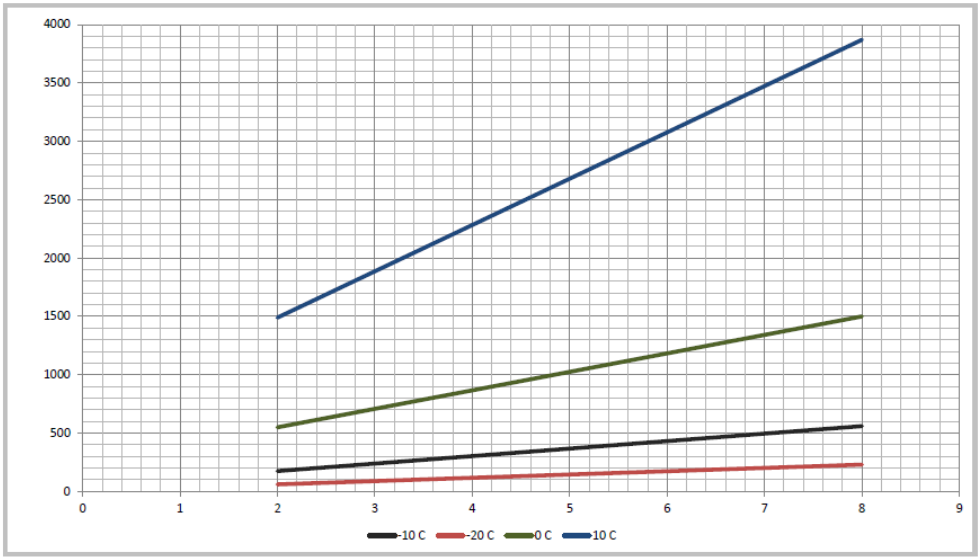


Figura 5.2. Ejemplo del número de avalanchas de origen térmico originadas cada segundo en un SPAD en condiciones de oscuridad, en función de la sobretensión de polarización inversa (en voltios) y para distintos valores de temperatura. Se observa que dicha cantidad varía de forma lineal con la sobretensión inversa y de forma exponencial con la temperatura. **[11]**

5.2 Estadística de la aleatoriedad

Como se ha visto anteriormente, un SPAD funcionando en condiciones de oscuridad y sometido a unos valores determinados de temperatura y sobretensión inversa se caracteriza por la generación de pulsos de corriente de avalancha que se pueden considerar aleatorios, independientes y con una tasa de generación determinada por dichas condiciones, además de por las características físicas del propio dispositivo.

El proceso de generación de avalanchas en oscuridad se puede modelar inicialmente considerando una variable aleatoria discreta X que sigue una distribución binomial expresada de la forma siguiente:

$$P_X[n] = \binom{N}{n} p^n (1-p)^{N-n}$$

siendo n el número total de pares electrón-hueco generados térmicamente, N el número total de electrones susceptibles de generar dichos pares, y p la probabilidad de que un electrón pase a la banda de conducción dando lugar a dicha generación.

En un SPAD en condiciones de oscuridad, el valor de p es muy pequeño y el valor de N es muy grande, de modo que en el límite, cuando N tiende a infinito y p tiende a cero, la distribución binomial anterior se puede aproximar a una distribución de tipo Poisson expresada de la forma siguiente:

$$P_X[n] = \frac{e^{-\lambda T}(\lambda T)^n}{n!}$$

siendo X la variable aleatoria discreta que representa la aparición de un pulso de avalancha, n el número de avalanchas registradas en un intervalo de tiempo T y λ la tasa promedio de aparición de los pulsos de avalancha en oscuridad.
Además, se cumple que $\sum_{n=0}^{\infty} P_X[n] = 1$, tal como se demuestra a continuación.

$$\sum_{n=0}^{\infty} P_X[n] = \sum_{n=0}^{\infty} \frac{e^{-\lambda T}(\lambda T)^n}{n!} = e^{-\lambda T} e^{\lambda T} = 1$$

Por otro lado, como las avalanchas se consideran sucesos independientes, la probabilidad de que una avalancha se produzca en un instante t posterior al instante s en que se produjo la avalancha previa equivale a la probabilidad de que no se haya producido ninguna avalancha durante el intervalo de tiempo $(s, s+t)$. Si llamamos Y a la variable aleatoria discreta que representa la aparición de una nueva avalancha, la probabilidad de que dicha avalancha se produzca transcurrido un intervalo t viene determinada por la siguiente expresión:

$$P[Y > t] = P_X[0] = \frac{e^{-\lambda t}(\lambda t)^0}{0!} = e^{-\lambda t}$$

Del mismo modo, la probabilidad de que la diferencia de tiempo entre la aparición de dos avalanchas consecutivas sea menor o igual que un valor t, o su función de distribución acumulada F, se puede calcular utilizando la probabilidad del suceso contrario calculada anteriormente:

$$F_Y(t) = P[Y \leq t] = 1 - P_X[0] = 1 - \frac{e^{-\lambda t}(\lambda t)^0}{0!} = 1 - e^{-\lambda t}$$

Como la función de distribución acumulada es la integral de la función de densidad de probabilidad correspondiente f, esta última se puede calcular derivando la primera, de modo que:

$$f_Y(t) = \frac{d}{dt}(1 - e^{-\lambda t}) = \lambda e^{-\lambda t}$$

Se observa, pues, que el tiempo transcurrido entre dos avalanchas consecutivas sigue una distribución de tipo **exponencial negativa**.

Para simular tiempos de aparición de avalanchas con aleatoriedad uniforme basta con igualar la función de distribución acumulada a una función de distribución uniforme U y obtener los valores de t correspondientes, tal y como se muestra a continuación.

$$U = 1 - e^{-\lambda t} \rightarrow U - 1 = e^{-\lambda t} \rightarrow ln(1-U) = -\lambda t \rightarrow t = -\frac{ln(1-U)}{\lambda}$$

En la **Tabla 1** del **Anexo 1** se muestran 100 sucesos de avalanchas correspondientes a una tasa media de aparición igual a 5 avalanchas por unidad de tiempo y organizados en 4 grupos de 25

sucesos. En cada grupo, en la primera columna de la izquierda se muestra un valor aleatorio uniforme entre 0 y 1, en la siguiente columna se muestra el tiempo transcurrido entre dos avalanchas consecutivas y en la siguiente columna se muestra el tiempo de aparición de cada avalancha.

En la **Figura 1** del **Anexo 1** se representa el histograma correspondiente a los tiempos transcurridos entre dos avalanchas consecutivas, donde se observa su variación exponencial negativa, y en la **Figura 2** del mismo anexo se representa el histograma correspondiente a los tiempos de aparición de las avalanchas, donde se observa la variación (casi) uniforme del número de avalanchas por unidad de tiempo (10 avalanchas cada 2 unidades de tiempo, lo que equivale a 5 cada unidad de tiempo).

5.3 Digitalización y entropía de la aleatoriedad

Con el fin de disponer y poder analizar series temporales de pulsos de avalancha originados en un SPAD, sin necesidad de tener que acceder y utilizar un equipo hardware en particular, dentro del ámbito de este trabajo se han desarrollo herramientas software para la generación sintética, representación y digitalización de dichas series, cuyos códigos se incluyen en los **Anexos 2** y **3**.

En la **Tabla 5.1** se muestran ejemplos de valores de tasas promedio de aparición de pulsos en un SPAD, las tasas promedio normalizadas (correspondientes a un pulso), los valores de la unidad de tiempo resultante y las duraciones de los intervalos de tiempo en el supuesto de que la unidad de tiempo se divida en 1000 intervalos como en el ejemplo anterior.

Tasa promedio real (pulsos / s):	200	500	1000	4000	20000
Tasa promedio normalizada:	1 pulso cada 2 5 ms	1 pulso cada 2 ms	1 pulso cada ms	1 pulso cada 0.25 ms	1 pulso cada 0.05 ms
Unidad de tiempo:	5 ms	2 ms	1 ms	0.25 ms	0.05 ms
Intervalos por unidad de tiempo:	1000				
Duración de cada intervalo:	5 us	2 us	1 us	0.25 us	0.05 us

Tabla 5.1. Duración de los intervalos de tiempo.

El objetivo de la digitalización de la aleatoriedad de los pulsos de avalancha es obtener una secuencia de bits equiprobables e independientes a partir de los sucesos aleatorios correspondientes a la aparición y detección de dichos pulsos (en nuestro caso, las series sintéticas generadas por las herramientas software desarrolladas a tal efecto).

En la literatura especializada se pueden encontrar diversas estrategias de digitalización que se pueden clasificar en dos grupos principales, según sea el criterio escogido como base para la definición de la variable aleatoria, tal y como se describe a continuación.

5.3.1 Estrategias basadas en el conteo de los pulsos

Un primer grupo de generadores de aleatoriedad mediante dispositivos SPAD se basa en la medida del número de pulsos de avalancha detectados en un cierto intervalo de tiempo. Estos generadores se diferencian principalmente por cómo definen la variable aleatoria asociada a dichos pulsos. Como ejemplo, a continuación se describen tres estrategias de este tipo y sus referencias correspondientes.

En **[17]** la variable aleatoria adopta el valor ‘0’ o ‘1’ en función de que el número de avalanchas detectadas en el intervalo de muestreo sea par o impar, respectivamente. Un esquema alternativo al anterior es el adoptado en **[18]**, donde la detección de pulsos individuales en tres SPAD conectados entre sí activa un contador digital de 2 bits que forman la secuencia aleatoria resultante. Finalmente, en **[19]** se describe otro método para el conteo de los pulsos de avalancha basado en la resolución espacial de los fotones incidentes en una matriz de SPAD conectados en paralelo. En este último caso, el pulso eléctrico resultante tiene un valor de tensión proporcional al número total de fotones incidentes en el dispositivo y la variable aleatoria adopta el valor ‘0’ o ‘1’ en función de que dicho número en un ciclo de muestreo sea mayor o menor, respectivamente, que en el ciclo anterior.

Estos esquemas de digitalización (y otros similares) no parecen adecuados y de aplicación práctica en el caso que nos ocupa, ya que la tasa de generación de pulsos de avalancha en un SPAD en condiciones de oscuridad es muy reducida y, además, el número de pulsos de avalancha necesarios para la generación de un bit de aleatoriedad suele ser muy superior a uno, que es justo lo contrario de lo que sería deseable en estos casos.

5.3.2 Estrategias basadas en el tiempo de llegada de los pulsos

El segundo grupo de generadores de aleatoriedad mediante dispositivos SPAD está formado por aquellos en los que la variable aleatoria está asociada al momento de aparición de los pulsos de avalancha. En este grupo se pueden distinguir cuatro categorías o tipos generales, en función de cómo se digitaliza la variable aleatoria correspondiente.

Tipo 1: digitalización por comparación de tiempos de aparición de pulsos consecutivos.

En este esquema, los bits de la variable aleatoria se generan en función de la relación existente entre los tiempos de llegada de tres pulsos de avalancha consecutivos, de modo que la variable aleatoria adopta el valor '0' si un pulso está más próximo al anterior que al siguiente, o el valor '1' en caso contrario.

Figura 5.3. Digitalización en función de la separación entre tres pulsos consecutivos. Bit = '0' si $t_{23} < t_{12}$ y Bit = '1' si $t_{23} > t_{12}$

La equiprobabilidad e independencia de los bits generados mediante esta estrategia está asegurada porque, tal y como se ha mostrado anteriormente, las separaciones entre pulsos de avalancha son independientes entre sí y siguen todas la misma distribución de tipo exponencial. Dicho de otra forma, el tiempo de espera entre un pulso y el siguiente no depende del tiempo transcurrido entre dicho pulso y el anterior, lo que suele expresarse diciendo que el tiempo de espera entre sucesos en un proceso de tipo Poisson no tiene memoria, ya que no se ve afectado ni determinado por lo que sucedió con anterioridad. Esta característica de falta de memoria se puede expresar mediante la siguiente ecuación:

$$P[X > t + x \mid X > t] = P[X > x] = e^{-\lambda x}$$

Cuya demostración se muestra a continuación:

Demostración 5.1:

$P[X > t + x \mid X > t] = \frac{P[(X>t+x)\ y\ (X>t)]}{P[X>t]} =$

(como la condición $X > t + x$ está incluida en la condición $X > t$)

$= \frac{P[X>t+x]}{P[X>t]} = \frac{e^{-\lambda(t+x)}}{e^{-\lambda t}} = e^{-\lambda x} = P[X > x]$

El principal inconveniente de este esquema de digitalización es que la tasa de aleatoriedad se limita a un bit por cada dos pulsos detectados (o 0.5 bits por pulso), si bien el método empleado en **[20]** permite aumentar esta tasa y aproximarse al máximo teórico de 1 bit por pulso en este tipo de digitalización.

La entropía mínima calculada mediante la **Ecuación 3.2** resulta igual al valor máximo de 1 bit por bit de codificación, ya que la probabilidad máxima de la variable aleatoria en este caso, como se ha visto antes, es igual a 1/2, de modo que:

$$H_\infty(X) = -log_2 P_{max} = -log_2 1/2 = 1$$

Tipo 2: digitalización por medida de tiempos con respecto a una señal de referencia.

En este tipo de digitalización se pueden distinguir, a su vez, dos soluciones diferentes, tal y como se describe a continuación.

Una primera solución consiste en utilizar una señal periódica externa de referencia con respecto a la cual se miden y comparan los tiempos de llegada de pulsos consecutivos **[21]**. Esta estrategia permite una mayor tasa de extracción de aleatoriedad que la anterior, alcanzando un valor de 1 bit por pulso.

La segunda solución se basa en establecer ventanas de muestreo divididas en un número fijo de intervalos a los cuales se les asigna una codificación binaria en función de su posición dentro de la ventana **[22] [23]**, tal y como se ilustra en la **Figura 5.4**.

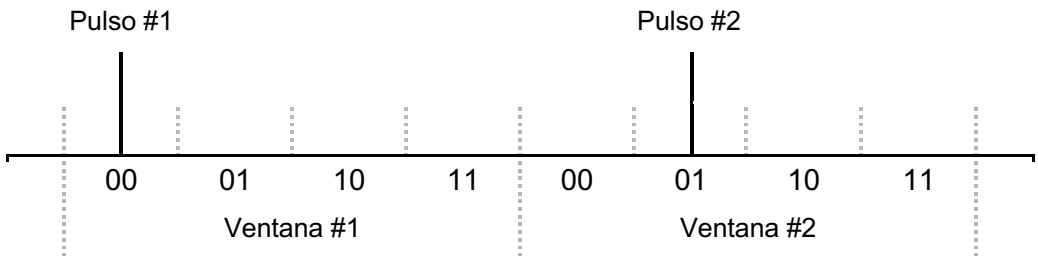


Figura 5.4. Digitalización por codificación del intervalo de la ventana de muestreo donde aparece el pulso.

Si el tamaño de las ventanas de muestreo se escoge de forma que la probabilidad de detectar más de un pulso en cada una de las ventanas sea muy reducida, la detección de un pulso se puede considerar que se distribuye uniformemente entre los intervalos en los que se divide la ventana de muestreo correspondiente (donde tiene lugar la detección), tal y como se demuestra a continuación.

Demostración 5.2:

Sean T el tamaño de la ventana de muestreo, N el número de intervalos en los que se divide dicha ventana y τ el tamaño de cada intervalo. La probabilidad de que la aparición del pulso suceda en un intervalo concreto de la ventana ($P[X \leq \tau]$) se puede calcular como la probabilidad de detectar un pulso en dicho intervalo ($P[N(\tau) = 1]$) condicionada por la aparición de dicho pulso dentro de la ventana de muestreo ($N(T) = 1$), es decir:

$$P[X \leq \tau] = P[N(\tau) = 1 \mid N(T) = 1] = \frac{P[N(\tau)=1 \; y \; N(T)=1]}{P[N(T)=1]} =$$

$$\frac{P[N(\tau)=1 \; y \; N(T)-N(\tau)=0]}{P[N(T)=1]} = \frac{P[N(\tau)=1]\,P[N(T)-N(\tau)=0]}{P[N(T)=1]} =$$

$$\frac{\lambda\tau e^{-\lambda\tau} e^{-\lambda(T-\tau)}}{\lambda T e^{-\lambda T}} = \frac{\tau}{T} = \frac{1}{T/\tau} = \frac{1}{N}$$

De modo que la probabilidad de detectar un pulso en un intervalo concreto de la ventana de muestreo no depende de la posición de dicho intervalo dentro de la ventana, siendo el valor de esta probabilidad igual a 1/N.

A continuación se describe la implementación de este tipo de digitalización realizada por el programa incluido en el **Anexo 2**.

Digitalización por codificación de los intervalos de detección de los pulsos:

Sean X: variable aleatoria.

t: intervalo de aparición del pulso.

v: ventana actual de muestreo.

v_prev: ventana previa de muestreo.

n: número de orden del intervalo de aparición del pulso $(de\ 0\ a\ bins-1)$.

$bins$: número de intervalos que forman la ventana de muestreo.

cod: número de bits de codificación.

$n = t\ MOD\ bins$

$v = t\ DIV\ bins$

$cod\ =\ log_2(bins)$

La variable aleatoria es la codificación binaria de n:

$X: n \rightarrow \{0,1\}^{cod}, si\ v \neq v\_prev\ (sólo\ se\ codifica\ el\ primer\ pulso\ detectado)$

En la **Tabla 5.2** se muestran los resultados de esta digitalización para una serie sintética de pulsos con una duración de 5000 unidades de tiempo, 10000 intervalos por unidad de tiempo y 8 intervalos por ventana de muestreo.

Unidad de tiempo:							ms
Tamaño de cada intervalo:							100 ns
Tamaño de la ventana de muestreo:							800 ns
Tasa promedio de aparición de pulsos (pulsos por unidad de tiempo):							1.02
Pulsos no codificados (%):							0.00
Tasa promedio de aleatoriedad (bits por unidad de tiempo):							3.08
Porcentaje de bits aleatorios con valor ‘1’ (%):							50.07
Probabilidad de aparición de un pulso en cada intervalo de la ventana de muestreo:							
0.120	0.125	0.129	0.127	0.126	0.124	0.127	0.122
Probabilidad teórica:							
1/8 = 0.125							

Tabla 5.2. Ejemplo de resultados de digitalización con 8 intervalos por ventana de muestreo.

En este caso, la entropía mínima de la variable aleatoria es el número de bits resultantes de la codificación de los N intervalos en los que se divide cada ventana de muestreo, tal y como resulta de la aplicación de la **Ecuación 3.2**.

$$H_\infty(X) = -log_2 P_{max} = -log_2 1/N = log_2 N$$

La tasa de aleatoriedad en este esquema de digitalización se puede aumentar de forma arbitraria reduciendo el tamaño y aumentando consiguientemente el número de los intervalos de la ventana de muestreo, en la medida que la resolución temporal del dispositivo así lo permita y se mantenga la condición de que la probabilidad de detectar más de un pulso en una ventana sea muy reducida.

Así, por ejemplo, en la **Tabla 5.3** se muestran los resultados de la digitalización de una serie sintética de pulsos con una duración de 5000 unidades de tiempo, 100000 intervalos por unidad de tiempo y 32 intervalos por ventana de muestreo.

Unidad de tiempo:							ms
Tamaño de cada intervalo:							10 ns
Tamaño de la ventana de muestreo:							320 ns
Tasa promedio de aparición de pulsos (pulsos por unidad de tiempo):							1.02
Pulsos no codificados (%):							0.02
Tasa promedio de aleatoriedad (bits por unidad de tiempo):							5.09
Porcentaje de bits aleatorios con valor ‘1’ (%):							50.27
Probabilidad de aparición de un pulso en cada intervalo de la ventana de muestreo:							
0.029	0.030	0.032	0.033	0.033	0.031	0.035	0.028
0.028	0.033	0.032	0.030	0.031	0.031	0.034	0.032
0.027	0.025	0.031	0.034	0.034	0.030	0.034	0.026
0.033	0.033	0.033	0.032	0.029	0.030	0.030	0.035
Probabilidad teórica:							
1/32 = 0.031							

Tabla 5.3. Ejemplo de resultados de digitalización con 32 intervalos por ventana de muestreo.

En este segundo caso, aunque la duración de la ventana de muestreo se reduce a 320 ns (frente a los 800 ns del caso anterior), 100 de los 5000 pulsos detectados (el 0.02%) no se codificaron por haberse detectado en ventanas de muestreo donde ya se había detectado un primer pulso. Esta estrategia de ignorar múltiples detecciones en una misma ventana de muestreo tiene un efecto positivo en la digitalización de la aleatoriedad, en la medida que ayuda a descartar la detección de los pulsos secundarios o posteriores que, de haberse codificado, habrían introducido correlaciones no deseadas en dicha digitalización.

En **[30]** se muestra el enlace a la aplicación web desarrollada en el ámbito de este trabajo, la cual permite simular y visualizar la estrategia de digitalización descrita anteriormente, cuyo código se incluye en el **Anexo 3**.

Tipo 3: digitalización basada en la detección por muestreo.

Este tipo de digitalización se ilustra en la **Figura 2.4** y se basa en la utilización de una señal externa para muestrear los pulsos de avalancha generados por el SPAD, de modo que la variable aleatoria adopta el valor '1' si en un instante de muestreo igual a $j\,\tau$ se detecta un pulso no habiéndose detectado otro pulso en el instante de muestreo anterior $(j-1)\,\tau$, o el valor de '0' en caso contrario, siendo τ el período de la señal de muestreo.

Por consiguiente, la tasa de aleatoriedad uniforme correspondiente a este esquema de digitalización podría ser, como mucho, igual a un bit por cada pulso detectado, para lo cual sería necesario que las probabilidades de detección y no detección de un pulso en el instante de muestreo fuesen iguales. Como la probabilidad de no aparición de pulsos en el intervalo τ es igual a $e^{-\lambda\tau}$, la probabilidad de aparición de algún pulso en dicho intervalo es igual a $1-e^{-\lambda\tau}$, y la condición de equiprobabilidad anterior se puede expresar como $e^{-\lambda\tau} = 1 - e^{-\lambda\tau} = 1/2$, de donde resulta que $e^{-\lambda\tau} = 1/2$, de modo que $-\lambda\tau = ln(1/2) = -ln\,2 \Rightarrow \tau = ln\,2/\lambda \Rightarrow 1/\tau = 1.44\lambda$. Con esta frecuencia de muestreo, la probabilidad de la variable aleatoria sería igual a 1/2, con un valor de entropía mínima resultante de la aplicación de la **Ecuación 3.2** igual a $H_\infty(X) = -log_2 P_{max} = -log_2 1/2 = 1$.

En **[6]** se propone aumentar la tasa de generación de aleatoriedad aumentando la frecuencia de muestreo. Este aumento conlleva la generación de muchos más ceros que unos en la secuencia

aleatoria, ya que el muestreo detecta no presencia de pulsos la mayor parte de las veces, al tiempo que incluye correlaciones en dicha secuencia derivadas de la detección de pulsos secundarios. Tras descartar algunos ciclos de muestreo para eliminar dichas correlaciones (reduciendo así la frecuencia efectiva del mismo), en la publicación se propone la utilización de un método de extracción de entropía **[24]** derivado del tipo von-Neumann descrito en el apartado 4.1.1 para generar una secuencia aleatoria lo más uniforme posible. Esta estrategia parece carecer de sentido en sí misma, ya que se propone aumentar la frecuencia de generación de la aleatoriedad cuando la frecuencia inicial (igual a la tasa promedio de aparición de pulsos) es la que proporciona ya la entropía máxima. Así pues, el valor de 10.4 bits por pulso (1.8×10^6 bits por segundo / 172×10^3 pulsos por segundo) obtenido en la publicación tras aumentar la frecuencia de muestreo unas 500 veces con respecto a la tasa promedio de aparición de pulsos y aplicar posteriormente el extractor **[24]** debe corresponder a un valor de entropía muy inferior al máximo (igual a 1).

NOTA: aunque en **[6]** y otras publicaciones referidas en este trabajo se consideran SPADs iluminados, con una tasa promedio de aparición de pulsos de avalancha de origen fotónico del orden de cientos de miles por segundo, tanto los cálculos realizados como los métodos descritos en dichas publicaciones se pueden considerar de aplicación también en el caso de SPADs funcionando en condiciones de oscuridad.

Tipo 4: digitalización basada en la precisión de la medida de tiempos.

Este último tipo de digitalización se basa, simplemente, en la medición del tiempo transcurrido entre dos pulsos de avalancha consecutivos. Dicha medición se reinicia cada vez que se detecta un pulso y ha transcurrido un tiempo muerto lo suficientemente largo como para evitar la detección de pulsos secundarios asociados al pulso anterior, y se extiende durante un intervalo de tiempo lo suficientemente largo como para considerar despreciable la probabilidad de un tiempo de espera superior a dicho intervalo **[25]**.

NOTA: conviene hacer hincapié en que este tipo de digitalización se basa en la precisión, no en la resolución, del equipo de medida utilizado para registrar el instante en el que se detecta un pulso de avalancha. La resolución se deriva de la frecuencia o el período de la medición, mientras que la precisión es la diferencia más pequeña que se puede registrar entre los valores de dos medidas cualesquiera.

Si llamamos δt a la precisión del equipo de medida del tiempo, $a\delta t$ al tiempo muerto (con un valor típico de 50 ns) , $b\delta t$ al tiempo durante el cual se extiende la medición y k al número del intervalo temporal en el que se registra la detección del pulso, la probabilidad $P[t_k]$ de que el tiempo de espera t se corresponda con un determinado valor de k se puede calcular como la probabilidad de que la llegada de un pulso se registre en el intervalo $[k\delta t, (k+1)\delta t]$ condicionada a la detección de dicho pulso en el intervalo $[a\delta t, b\delta t]$, tal y como se muestra a continuación.

$$P[t_k] = P[k\delta t \leq t \leq (k+1)\delta t \mid a\delta t \leq t \leq b\delta t] = \frac{P[k\delta t \leq t \leq (k+1)\delta t \ y \ a\delta t \leq t \leq b\delta t]}{P[a\delta t \leq t \leq b\delta t]}$$

Como $k = a, a+1, \dots, b-1$, la condición $k\delta t \leq t \leq (k+1)\delta t$ está incluida en la condición $a\delta t \leq t \leq b\delta$, de modo que:

$$P[t_k] = \frac{P[k\delta t \leq t \leq (k+1)\delta t]}{P[a\delta t \leq t \leq b\delta t]} = \frac{P[t \leq (k+1)\delta t] - P[t \leq k\delta t]}{P[a\delta t \leq t \leq b\delta t]} = \frac{(1 - e^{-\lambda(k+1)\delta t}) - (1 - e^{-\lambda k\delta t})}{e^{-\lambda a\delta t} - e^{-\lambda b\delta t}} = \frac{e^{-\lambda k\delta t} - e^{-\lambda(k+1)\delta t}}{e^{-\lambda a\delta t} - e^{-\lambda b\delta t}}$$

$$= \frac{e^{-\lambda k\delta t} - (e^{-\lambda k\delta t} e^{-\lambda\delta t})}{e^{-\lambda a\delta t} - e^{-\lambda b\delta t}} = \frac{e^{-\lambda k\delta t}(1 - e^{-\lambda\delta t})}{e^{-\lambda a\delta t} - e^{-\lambda b\delta t}} \simeq \frac{e^{-\lambda k\delta t}(1 - e^{-\lambda\delta t})}{e^{-\lambda a\delta t}}$$

Donde se ha considerado que $e^{-\lambda b\delta t}$ es muy inferior a 1, ya que la duración del intervalo $b\delta t$ se escoge de forma que la probabilidad de que no aparezca un pulso en dicho intervalo sea prácticamente nula. De modo que:

$$P[t_k] \simeq e^{-\lambda(k-a)\delta t}(1 - e^{-\lambda\delta t}) \simeq e^{-\lambda(k-a)\delta t}(1 - (1 - \lambda\delta t)) = \lambda\delta t \, e^{-\lambda(k-a)\delta t}$$

Donde en la última aproximación se ha sustituido la función exponencial por los dos primeros términos correspondientes a su expansión en serie de Taylor.

Esta probabilidad es máxima cuando k es igual a a, es decir, cuando se inicia el muestreo de la detección del pulso al finalizar la espera correspondiente al tiempo muerto, resultando el siguiente valor de entropía mínima obtenido mediante la **Ecuación 3.2**.

$$H_\infty(X) = -log_2 P_{max}[t_k] \simeq -log_2(e^{-\lambda(a-a)\delta t}(1 - e^{-\lambda\delta t})) = -log_2(1 - e^{-\lambda\delta t})$$

En la **Figura 5.5** se representa el valor de la entropía mínima resultante en función de la tasa promedio de aparición de pulsos en el SPAD en condiciones de oscuridad (λ) y la precisión del equipo de medida (δt). En esta figura se observa cómo la entropía mínima disminuye a medida que aumenta la tasa promedio de aparición de pulsos en el intervalo de medida y disminuye la precisión disponible en el equipo de medida.

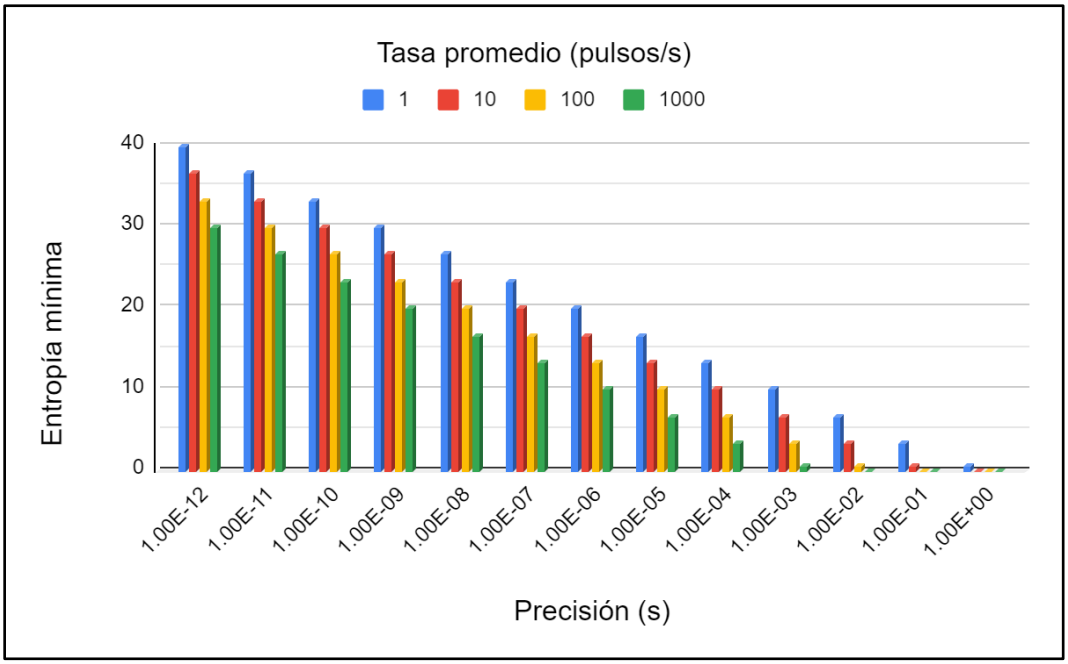


Figura 5.5. Valor de la entropía mínima en función de la precisión del equipo y la tasa promedio de pulsos.

Como ejemplo de cálculo, en esta figura se observa que para una tasa promedio de 1000 pulsos por segundo, el tiempo de llegada de los pulsos se podría codificar utilizando 6 bits si la precisión de la medida temporal fuese igual (o superior) a 10^{-5} segundos (10 microsegundos). $(H_\infty(X) = -log_2(1 - e^{-\lambda\delta t}) = -log_2(1 - e^{-0.01}) = 6.65\ bits)$. El número de bits utilizados en esta codificación podría aumentar, por ejemplo, hasta 20 si la precisión aumentase hasta 1 nanosegundo.

En este tipo de digitalización la distribución de los tiempos de espera registrados entre dos pulsos consecutivos no se puede considerar uniforme, sino exponencial (como debe ser), lo que hace necesaria una etapa de postprocesado o extracción de entropía para mejorar la calidad de aleatoriedad obtenida. No obstante, esta etapa de extracción de entropía puede evitarse si en lugar de registrar los tiempos de espera se registra la función de distribución acumulada de dichos

tiempos, ya que dicha distribución se comporta, a su vez, como una distribución uniforme, tal y como se demuestra en **[26]** y a continuación.

Demostración 5.3

Sea X una variable aleatoria continua y $Z = F(X)$ su función de distribución acumulada.

Para analizar cómo se distribuye Z, calculamos su función de distribución acumulada:

$F_Z(z) = P[Z \leq z] = P[F(X) \leq z]$

Como F es estrictamente creciente, la función inversa F^{-1}existe y está bien definida, de modo que:

$P[F(X) \leq z] = P[F^{-1}F(X) \leq F^{-1}(z)] = P[X \leq F^{-1}(z)] = F(F^{-1}(z)) = z$ para $0 \leq z \leq 1$

Como la función de densidad de probabilidad es la derivada de la función de probabilidad acumulada, $f_Z(z) = \frac{d}{dt}F_Z(z) = \frac{d}{dt}z = 1$, de modo que la función de distribución acumulada se distribuye uniformemente entre 0 y 1.

En **[25]** se hace uso de esta propiedad para obtener una distribución uniforme derivada de la transformación correspondiente de los tiempos de espera entre pulsos de avalancha consecutivos, y dicha propiedad se puede observar utilizando, como ejemplo, los valores de los tiempos mostrados en la **Tabla 1** del **Anexo 1**. En la **Tabla 2** de dicho anexo se incluyen estos tiempos y una nueva columna con los valores de la función de distribución acumulada correspondiente a cada tiempo de espera (calculados como $1 - e^{-\lambda t}$). En la **Figura 3** del mismo anexo se muestra el histograma de estos últimos valores, observándose su variación (casi) uniforme. Esta transformación no altera la entropía mínima obtenida inicialmente, de modo que los valores de las probabilidades acumuladas (comprendidos entre 0 y 1) se podrían codificar utilizando el número de bits correspondientes a la tasa promedio de aparición de pulsos y la precisión del equipo de medida del tiempo.

La función de distribución acumulada de un tiempo $t_k$se puede calcular como sigue:

$$F(t_k) = P[t \leq k\delta t] = 1 - P[t \geq k\delta t \mid t \geq a\delta t]$$

$$F(t_k) = 1 - \frac{P[t \geq (k+1)\delta t \ y \ t \geq a\delta t]}{P[\, t \geq a\delta t]} = 1 - \frac{P[t \geq (k+1)\delta t]}{P[\, t \geq a\delta t]}$$

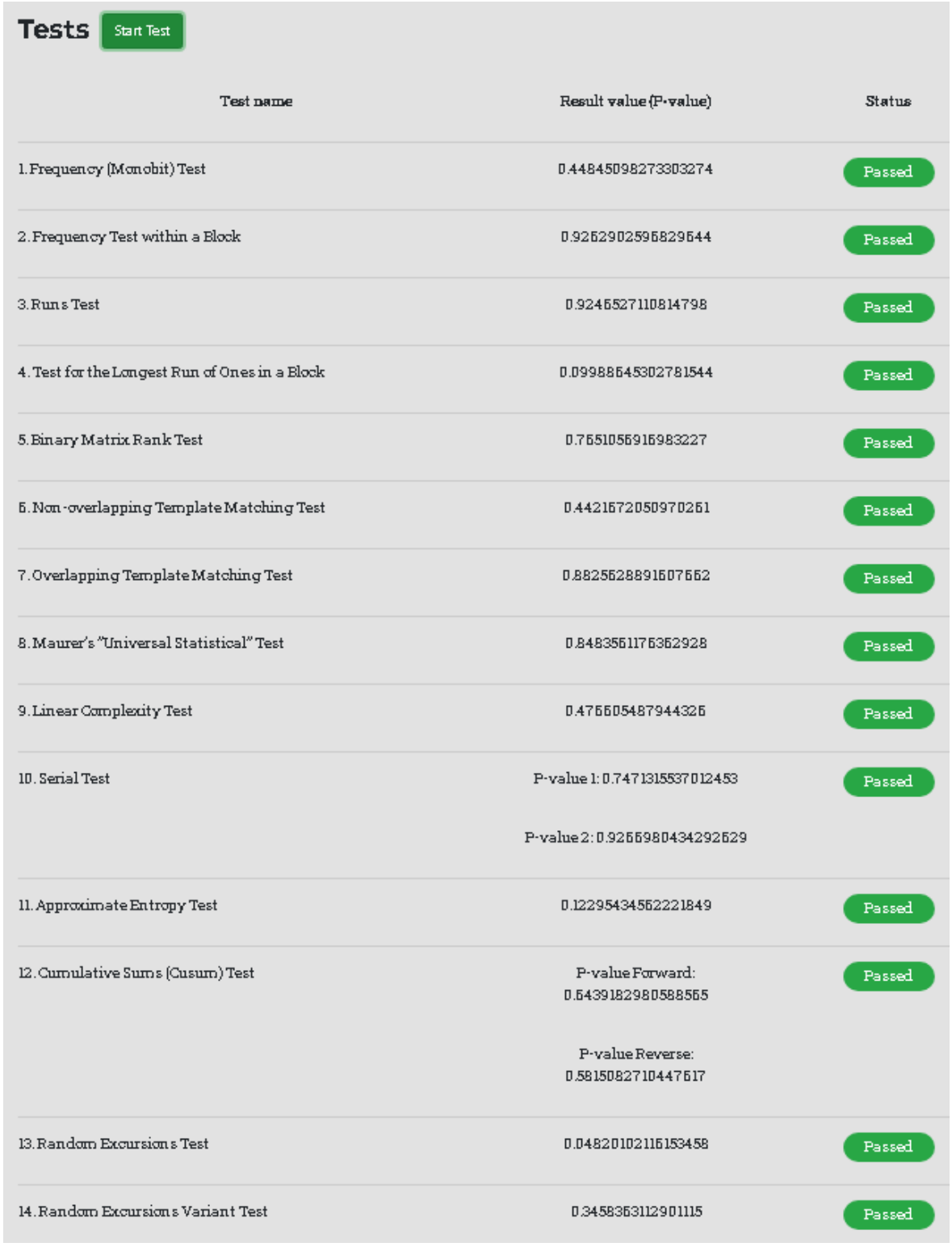
$$F(t_k) = 1 - \frac{e^{-\lambda(k+1)\delta t}}{e^{-\lambda a \delta t}} = 1 - e^{-\lambda(k+1-a)\delta t}$$

El valor de k es igual a la división entera de t y δt (siendo t el tiempo de espera entre dos pulsos consecutivos), mientras que el valor de a es igual a la división entera del tiempo muerto predeterminado y δt.

Siguiendo esta estrategia se ha generado una serie sintética de medio millón de pulsos con una tasa promedio de aparición de 1000 pulsos por segundo, una precisión en la medida del tiempo igual a 3 microsegundos y un tiempo muerto también igual a 3 microsegundos, resultando una tasa promedio de generación de aleatoriedad igual a 8 bits cada milisegundo, equivalente a 8 kilobits por segundo y a un total de medio megabyte de datos aleatorios.

Para evaluar de forma más exhaustiva la calidad de la aleatoriedad obtenida mediante estas estrategias de digitalización se ha utilizado la herramienta online **[29]**, la cual permite introducir secuencias de bits y someterlas a las pruebas estadísticas especificadas en el documento **NIST SP 800-22 [16]**:

Haciendo uso de esta herramienta, el medio megabyte de datos aleatorios obtenidos mediante la estrategia de digitalización anterior se ha sometido a estas pruebas estadísticas, obteniéndose un resultado satisfactorio en todas ellas, tal y como se muestra en la **Figura 5.6**.



Tests Start Test

Test name	Result value (P-value)	Status
1. Frequency (Monobit) Test	0.44845098273303274	Passed
2. Frequency Test within a Block	0.9252902596829644	Passed
3. Runs Test	0.9246527110814798	Passed
4. Test for the Longest Run of Ones in a Block	0.09988645302781544	Passed
5. Binary Matrix Rank Test	0.7651056916983227	Passed
6. Non-overlapping Template Matching Test	0.4421672050970261	Passed
7. Overlapping Template Matching Test	0.8825628891607662	Passed
8. Maurer's "Universal Statistical" Test	0.8483561176362928	Passed
9. Linear Complexity Test	0.476605487944326	Passed
10. Serial Test	P-value 1: 0.7471315537012453 P-value 2: 0.9266980434292629	Passed
11. Approximate Entropy Test	0.12295434562221849	Passed
12. Cumulative Sums (Cusum) Test	P-value Forward: 0.6439182980588565 P-value Reverse: 0.5815082710447617	Passed
13. Random Excursions Test	0.04820102116153458	Passed
14. Random Excursions Variant Test	0.3458363112901115	Passed

Figura 5.6. Resultados de las pruebas estadísticas. **[29]**

Conclusiones y trabajos futuros

El principio de funcionamiento de los dispositivos SPAD en condiciones de oscuridad puede permitir su utilización como generadores de números realmente aleatorios, siendo los pulsos de corriente de avalancha originados por mecanismos térmicos y de efecto túnel, en ausencia de fotones incidentes, la fuente de aleatoriedad. Este modo de funcionamiento permite la realización de generadores más simples, asequibles y compactos, con un alto nivel de integración a nivel hardware.

Dicha fuente de aleatoriedad se caracteriza por una baja tasa de generación, la cual puede oscilar entre unas pocas decenas y algunos miles de pulsos por segundo. Este parámetro de diseño viene determinado por los valores de tensión de polarización inversa (que es otro parámetro de diseño) y temperatura a los que se ve sometido el dispositivo, lo cual puede resultar de gran utilidad de cara a realizar las necesarias comprobaciones periódicas del correcto funcionamiento del generador (health tests).

Un parámetro crítico de diseño del generador es su tiempo muerto, entendido como el tiempo que debe transcurrir hasta que se restablezcan las condiciones propicias en el SPAD para que se pueda originar y detectar un nuevo pulso de avalancha, durante el cual el generador no está operativo y cuya duración puede extenderse para evitar la detección de posibles pulsos de avalancha secundarios. Un valor demasiado alto de este parámetro puede requerir la corrección de la función de distribución de probabilidad, tal y como se describe en **[27]**.

Se pueden adoptar distintas estrategias de digitalización de la aleatoriedad, basadas tanto en el conteo como en el tiempo de espera entre pulsos. Tanto el primer grupo como el tipo 3 del segundo grupo se caracterizan por su mayor sencillez (al no incluir elementos de medición del tiempo), pero sus tasas de generación son demasiado reducidas, al igual que en el tipo 1 y en la primera variante del tipo 2, de modo que el interés se centra en la segunda variante del tipo 2 y en el tipo 4. En comparación con la segunda variante del tipo 2 y suponiendo que la duración de los intervalos de muestreo en ese caso es igual a la precisión temporal utilizada en el tipo 4, el número de bits aleatorios que se pueden generar por pulso de avalancha en el tipo 4 es superior al doble del correspondiente en dicha variante del tipo 2, si bien la técnica de medida y

digitalización del tiempo resulta más compleja. En la segunda variante del tipo 2 la digitalización se realiza de forma síncrona basada en la resolución temporal (se muestrea la señal del SPAD y se codifica el intervalo de detección dentro de la ventana de muestreo), mientras que en el tipo 4 la digitalización se realiza de forma asíncrona basada en la precisión temporal (la señal del SPAD marca los instantes de inicio y final del intervalo de tiempo transcurrido entre dos pulsos de avalancha consecutivos). Esto último requiere una técnica más sofisticada, de modo que la estrategia de digitalización preferible en el caso de generadores de aleatoriedad basados en SPAD funcionando en condiciones de oscuridad es la correspondiente a la segunda variante del tipo 2.

La principal limitación de estos generadores, en comparación con los basados en pulsos de avalancha de origen fotónico, es su baja tasa de generación de aleatoriedad (la cual se puede aumentar utilizando varios dispositivos en paralelo). Sin embargo, esta característica, junto a la posibilidad de generar aleatoriedad uniforme y su alto nivel de integración, los hace especialmente idóneos como generadores de semilla realmente aleatoria y fácilmente integrable a nivel de hardware, lo que puede resultar de gran interés y utilidad en extensiones de micro-arquitecturas como la RISC-V/TRNG **[28]**.

Para finalizar se proponen dos líneas de trabajo futuro que pueden servir de continuidad al presente TFM. La primera línea estaría dirigida a profundizar en la caracterización y el modelado del funcionamiento de los SPADs en condiciones de oscuridad, tanto desde el punto de vista de la física del dispositivo como de su comportamiento estadístico **[31] [32] [33]**. La segunda línea tendría por objeto estudiar y analizar los distintos esquemas de configuración e integración de SPADs a nivel de chip, con especial atención a aquellos que permitan aumentar de forma considerable la tasa de generación de aleatoriedad en condiciones de oscuridad **[34] [35] [36]**.

Referencias

[1] Diaconis, P., Holmes, S., & Montgomery, R. (2007). Dynamical bias in the coin toss. *SIAM review*, *49*(2), 211-235.

[2] TrueRNG v3 – Hardware Random Number Generator.
https://ubld.it/truerng_v3

[3] 5G smartphone equipped with a Quantum Random Number Generator.
https://qrange.eu/news/5g-smartphone-equipped-quantum-random-number-generator

[4] Quside PCIe Series.
https://quside.com/pcie-series/

[5] ID Quantique. Random Number Generation White Paper.
https://marketing.idquantique.com/acton/attachment/11868/f-0226/1/-/-/-/-/What%20is%20the%20Q%20in%20QRNG_White%20Paper.pdf

[6] Stanco, A., Marangon, D. G., Vallone, G., Burri, S., Charbon, E., & Villoresi, P. (2020). Efficient random number generation techniques for CMOS single-photon avalanche diode array exploiting fast time tagging units. *Physical Review Research*, *2*(2), 023287.

[7] Johnston, D. (2018). Random Number Generators—Principles and Practices. In *Random Number Generators—Principles and Practices*. De| G Press.
Random Number Generators Part 9. Entropy Extractors Part 3.
http://y2u.be/i5EjIZEOHDw

[8] Johnston, D. (2018). Random Number Generators—Principles and Practices. In *Random Number Generators—Principles and Practices*. De| G Press.
Random Number Generators, Part 11 - NIST SP800-90B Entropy Extractors.
http://y2u.be/A7rK1ErubVU

[9] The Joy of Cryptography, by Mike Rosulek.
https://joyofcryptography.com/pdf/chap10.pdf

[10] Regazzoni, F., Amri, E., Burri, S., Rusca, D., Zbinden, H., & Charbon, E. (2021). A high speed integrated quantum random number generator with on-chip real-time randomness extraction. *arXiv preprint arXiv:2102.06238*.

[11] Laser Components. SAP-Series Silicon Geiger Mode Avalanche Photodiode.
https://www.lasercomponents.com/fileadmin/user_upload/home/Datasheets/lc-apd/sap-series-geiger-mode-apd.pdf

[12] Barker, E. B., & Kelsey, J. M. (2012). Sp 800-90a. Recommendation for random number generation using deterministic random bit generators.

[13] Turan, M. S., Barker, E., Kelsey, J., McKay, K. A., Baish, M. L., & Boyle, M. (2018). Recommendation for the entropy sources used for random bit generation. *NIST Special Publication*, *800*(90B), 102.

[14] Barker, E., Kelsey, J., McKay, K., Roginsky, A., & Sönmez Turan, M. (2022). *Recommendation for Random Bit Generator (RBG) Constructions (3rd Draft)* (No. NIST Special Publication (SP) 800-90C (Draft)). National Institute of Standards and Technology.

[15] Buller, D., Kaufer, A., Roginsky, A., & Turan, M. S. (2023). Discussion on the Full Entropy Assumption of the SP 800-90 Series.

[16] Bassham III, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., ... & Vo, S. (2010). *Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology.

[17] Fürst, H., Weier, H., Nauerth, S., Marangon, D. G., Kurtsiefer, C., & Weinfurter, H. (2010). High speed optical quantum random number generation. *Optics express*, *18*(12), 13029-13037.

[18] Ejdehakosh, S., Ansarian, M., & Karami, M. A. (2020). A new optical random number generator circuit design using single-photon avalanche diodes. *Optik*, *224*, 165698.

[19] Cai, Y., Chen, Y., Chen, X., Wu, G., & Wu, E. (2021). Quantum characteristics and applications of multi-pixel photon counter. *Microwave and Optical Technology Letters*, *63*(8), 2052-2057.

[20] Tontini, A., Gasparini, L., Massari, N., & Passerone, R. (2019). SPAD-Based Quantum Random Number Generator With an N^{th}-Order Rank Algorithm on FPGA. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *66*(12), 2067-2071.

[21] Xu, H., Massari, N., Gasparini, L., Meneghetti, A., & Tomasi, A. (2019). A SPAD-based random number generator pixel based on the arrival time of photons. *Integration*, *64*, 22-28.

[22] Nie, Y. Q., Zhang, H. F., Zhang, Z., Wang, J., Ma, X., Zhang, J., & Pan, J. W. (2014). Practical and fast quantum random number generation based on photon arrival time relative to external reference. *Applied Physics Letters*, *104*(5), 051110.

[23] Banerjee, A., Aggarwal, D., Sharma, A., & Yadav, G. (2020). Unpredictable and Uniform RNG based on time of arrival using InGaAs Detectors. *arXiv preprint arXiv:2010.12898*.

[24] Peres, Y. (1992). Iterating von Neumann's procedure for extracting random bits. *The Annals of Statistics*, 590-597.

[25] Lin, J., Wang, Y., Cao, Q., Kuang, J., & Wang, L. (2019). True random number generation based on arrival time and position of dark counts in a multichannel silicon photomultiplier. *Review of Scientific Instruments*, *90*(11), 114704.

[26] Durai, S. A., & Saro, E. A. (2006). Image compression with back-propagation neural network using cumulative distribution function. *World Academy of Science, Engineering and Technology*, *17*, 60-64.

[27] Menkart, N., Hart, J. D., Murphy, T. E., & Roy, R. (2022). Dark current and single photon detection by 1550 nm avalanche photodiodes: dead time corrected probability distributions and entropy rates. *Optics Express*, *30*(22), 39431-39444.

[28] Saarinen, M. J. O., Newell, G. R., & Marshall, B. (2022). Development of the RISC-V entropy source interface. *Journal of Cryptographic Engineering*, *12*(4), 371-386.

[29] RANDOM BITSTREAM TESTER.
https://mzsoltmolnar.github.io/random-bitstream-tester/

[30] Herramienta Web.
https://mrpiay.github.io/TRNG_SPAD_1/

[31] Panglosse, A., Martin-Gonthier, P., Marcelot, O., Virmontois, C., Saint-Pé, O., & Magnan, P. (2020). Dark count rate modeling in single-photon avalanche diodes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *67*(5), 1507-1515.

[32] Yu, W. J., Zhang, Y., Xu, M. Z., & Lu, X. M. (2020). Dark count in single-photon avalanche diodes: A novel statistical behavioral model. *Chinese Physics B*, *29*(4), 048503.

[33] Kang, Y., Lu, H. X., Lo, Y. H., Bethune, D. S., & Risk, W. P. (2003). Dark count probability and quantum efficiency of avalanche photodiodes for single-photon detection. *Applied physics letters*, *83*(14), 2955-2957.

[34] Regazzoni, F., Amri, E., Burri, S., Rusca, D., Zbinden, H., & Charbon, E. (2021). A high speed integrated quantum random number generator with on-chip real-time randomness extraction. *arXiv preprint arXiv:2102.06238*.

[35] Xu, H., Perenzoni, D., Tomasi, A., & Massari, N. (2018). A 16 x16 Pixel Post-Processing Free Quantum Random Number Generator Based on SPADs. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *65*(5), 627-631.

[36] Park, B. K., Park, H., Kim, Y. S., Kang, J. S., Yeom, Y., Ye, C., ... & Han, S. W. (2019). Practical true random number generator using CMOS image sensor dark noise. *IEEE Access*, *7*, 91407-91413.

Anexos

1. Cálculos

1	0.900	0.460	0.460	**26**	0.312	0.075	5.885	**51**	0.090	0.019	9.391	**76**	0.353	0.087	14.534
2	0.717	0.253	0.713	**27**	0.365	0.091	5.975	**52**	0.219	0.049	9.441	**77**	0.027	0.005	14.539
3	0.096	0.020	0.733	**28**	0.002	0.000	5.976	**53**	0.044	0.009	9.450	**78**	0.905	0.471	15.010
4	0.485	0.133	0.865	**29**	0.348	0.086	6.062	**54**	0.908	0.477	9.927	**79**	0.944	0.577	15.586
5	0.370	0.092	0.958	**30**	0.905	0.471	6.532	**55**	0.856	0.387	10.314	**80**	0.651	0.211	15.797
6	0.851	0.381	1.339	**31**	0.525	0.149	6.681	**56**	0.570	0.169	10.483	**81**	0.082	0.017	15.814
7	0.793	0.315	1.653	**32**	0.653	0.212	6.893	**57**	0.909	0.479	10.961	**82**	0.837	0.363	16.177
8	0.697	0.239	1.892	**33**	0.562	0.165	7.057	**58**	0.130	0.028	10.989	**83**	0.193	0.043	16.220
9	0.541	0.156	2.048	**34**	0.163	0.036	7.093	**59**	0.715	0.251	11.240	**84**	0.728	0.261	16.481
10	0.240	0.055	2.103	**35**	0.983	0.818	7.911	**60**	0.549	0.159	11.399	**85**	0.515	0.145	16.626
11	0.888	0.438	2.541	**36**	0.115	0.024	7.935	**61**	0.202	0.045	11.445	**86**	0.540	0.155	16.781
12	0.424	0.110	2.652	**37**	0.067	0.014	7.949	**62**	0.664	0.218	11.662	**87**	0.418	0.108	16.889
13	0.777	0.300	2.952	**38**	0.415	0.107	8.056	**63**	0.341	0.084	11.746	**88**	0.973	0.720	17.610
14	0.653	0.212	3.164	**39**	0.385	0.097	8.154	**64**	0.775	0.298	12.044	**89**	0.188	0.042	17.651
15	0.477	0.130	3.293	**40**	0.642	0.205	8.359	**65**	0.789	0.311	12.355	**90**	0.743	0.271	17.923
16	0.661	0.216	3.509	**41**	0.221	0.050	8.409	**66**	0.927	0.523	12.878	**91**	0.351	0.087	18.009
17	0.672	0.223	3.732	**42**	0.313	0.075	8.484	**67**	0.292	0.069	12.947	**92**	0.453	0.121	18.130
18	0.600	0.183	3.916	**43**	0.334	0.081	8.565	**68**	0.305	0.073	13.020	**93**	0.620	0.194	18.324
19	0.161	0.035	3.951	**44**	0.163	0.036	8.601	**69**	0.798	0.320	13.340	**94**	0.002	0.000	18.324
20	0.153	0.033	3.984	**45**	0.829	0.353	8.954	**70**	0.485	0.133	13.473	**95**	0.081	0.017	18.341
21	0.177	0.039	4.023	**46**	0.221	0.050	9.004	**71**	0.936	0.550	14.023	**96**	0.549	0.159	18.500
22	0.965	0.670	4.693	**47**	0.574	0.171	9.175	**72**	0.245	0.056	14.079	**97**	0.870	0.408	18.908
23	0.952	0.609	5.302	**48**	0.435	0.114	9.289	**73**	0.051	0.010	14.089	**98**	0.866	0.402	19.311
24	0.708	0.246	5.548	**49**	0.188	0.042	9.331	**74**	0.708	0.246	14.336	**99**	0.243	0.056	19.367
25	0.730	0.262	5.810	**50**	0.190	0.042	9.373	**75**	0.425	0.111	14.446	**100**	0.712	0.249	19.615

Tabla 1. Serie sintética correspondiente a 100 pulsos con una tasa de 5 pulsos por unidad de tiempo. De izquierda a derecha: valor aleatorio uniforme entre 0 y 1, tiempo transcurrido entre dos avalanchas consecutivas y tiempo de aparición de cada avalancha.

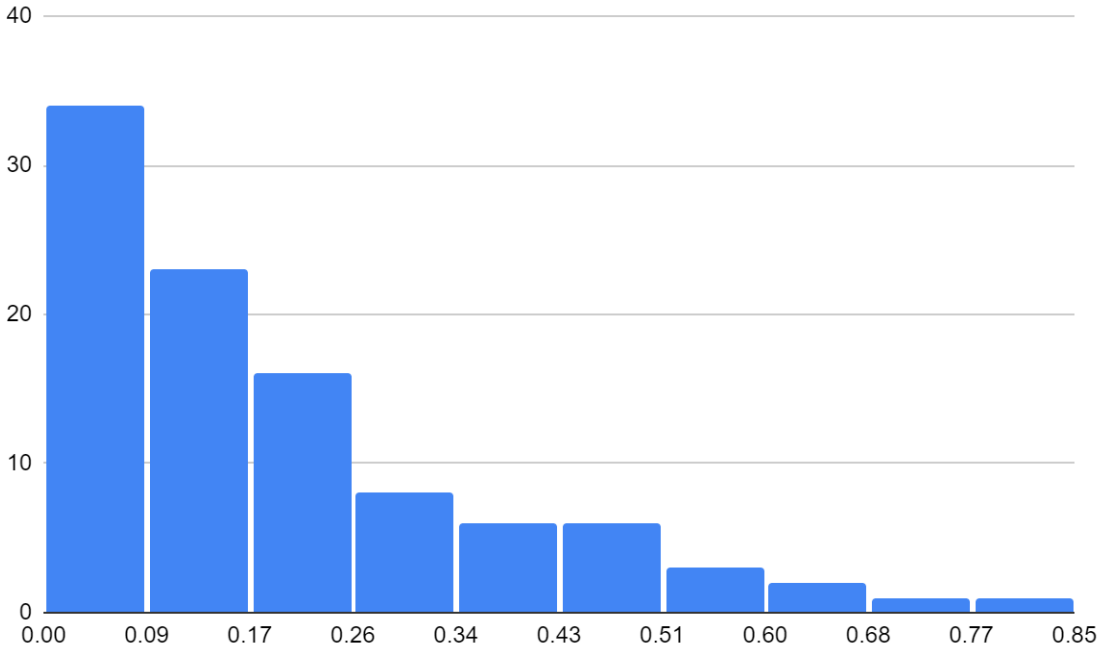


Figura 1. Histograma de los tiempos de espera entre pulsos consecutivos de la **Tabla 1**.

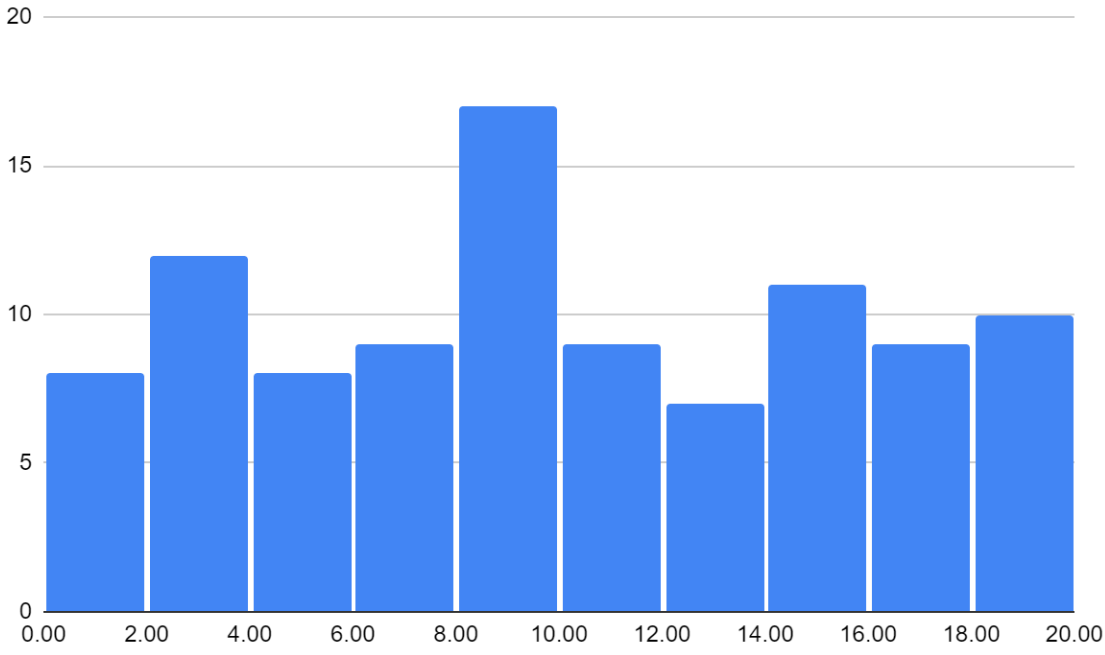


Figura 2. Histograma de los tiempos de aparición de pulsos de la **Tabla 1**.

1	0.460	0.8997	**26**	0.075	0.3119	**51**	0.019	0.0897	**76**	0.087	0.3533
2	0.253	0.7173	**27**	0.091	0.3647	**52**	0.049	0.2189	**77**	0.005	0.0270
3	0.020	0.0956	**28**	0.000	0.0018	**53**	0.009	0.0439	**78**	0.471	0.9049
4	0.133	0.4846	**29**	0.086	0.3485	**54**	0.477	0.9079	**79**	0.577	0.9441
5	0.092	0.3696	**30**	0.471	0.9049	**55**	0.387	0.8555	**80**	0.211	0.6512
6	0.381	0.8513	**31**	0.149	0.5250	**56**	0.169	0.5701	**81**	0.017	0.0825
7	0.315	0.7927	**32**	0.212	0.6528	**57**	0.479	0.9087	**82**	0.363	0.8371
8	0.239	0.6973	**33**	0.165	0.5617	**58**	0.028	0.1305	**83**	0.043	0.1929
9	0.156	0.5414	**34**	0.036	0.1634	**59**	0.251	0.7151	**84**	0.261	0.7283
10	0.055	0.2397	**35**	0.818	0.9832	**60**	0.159	0.5487	**85**	0.145	0.5153
11	0.438	0.8882	**36**	0.024	0.1153	**61**	0.045	0.2020	**86**	0.155	0.5399
12	0.110	0.4239	**37**	0.014	0.0665	**62**	0.218	0.6636	**87**	0.108	0.4185
13	0.300	0.7773	**38**	0.107	0.4148	**63**	0.084	0.3414	**88**	0.720	0.9727
14	0.212	0.6528	**39**	0.097	0.3853	**64**	0.298	0.7745	**89**	0.042	0.1878
15	0.130	0.4767	**40**	0.205	0.6420	**65**	0.311	0.7889	**90**	0.271	0.7425
16	0.216	0.6609	**41**	0.050	0.2209	**66**	0.523	0.9269	**91**	0.087	0.3515
17	0.223	0.6719	**42**	0.075	0.3129	**67**	0.069	0.2921	**92**	0.121	0.4531
18	0.183	0.6003	**43**	0.081	0.3344	**68**	0.073	0.3050	**93**	0.194	0.6201
19	0.035	0.1608	**44**	0.036	0.1634	**69**	0.320	0.7984	**94**	0.000	0.0025
20	0.033	0.1534	**45**	0.353	0.8288	**70**	0.133	0.4847	**95**	0.017	0.0809
21	0.039	0.1772	**46**	0.050	0.2206	**71**	0.550	0.9360	**96**	0.159	0.5489
22	0.670	0.9649	**47**	0.171	0.5738	**72**	0.056	0.2455	**97**	0.408	0.8701
23	0.609	0.9524	**48**	0.114	0.4355	**73**	0.010	0.0506	**98**	0.402	0.8663
24	0.246	0.7082	**49**	0.042	0.1883	**74**	0.246	0.7082	**99**	0.056	0.2429
25	0.262	0.7298	**50**	0.042	0.1896	**75**	0.111	0.4250	**100**	0.249	0.7118

Tabla 2. De izquierda a derecha: tiempos de espera entre pulsos consecutivos de la **Tabla 1** y valores correspondientes a la función de distribución acumulada de dichos tiempos.

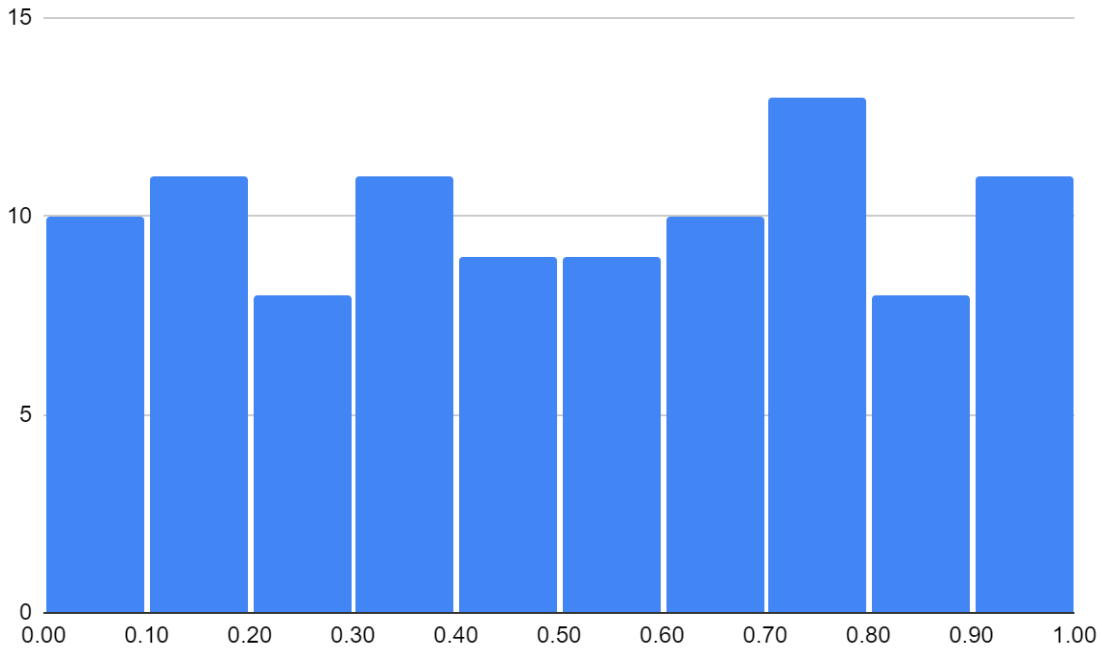


Figura 3. Histograma de los valores de la función de distribución acumulada de la **Tabla 2**.

2. Programa

```
'''
    ----------
    PROGRAMA
    ----------

    Generación, representación y digitalización
    de series de pulsos de avalancha en SPAD
    para la generación de números aleatorios.

'''

# módulos importados
import random
import math

# función para convertir decimal en binario
def dec2bin(dec, fill = 0):
    global unos_bins, ceros_bins
    if dec == 0:
        bits = '0'
        ceros_bins = ceros_bins + 1
    else:
        bits = ''
    while dec > 0:
        bit = dec % 2
        if bit == 0:
            ceros_bins = ceros_bins + 1
        else:
            unos_bins = unos_bins + 1
        bits = str(bit) + bits
        dec = dec // 2
    if fill > 0:
        pad = fill - len(bits)
        if pad > 0 :
            for i in range(pad):
                bits = '0' + bits
                ceros_bins = ceros_bins + 1
    return bits

# abrir archivos de salida
fdist = open('dist.txt', 'w')
fbins = open('bins.txt', 'w')

# datos de entrada
tasa = 1         # tasa promedio de aparición de pulsos
                 # (número de pulsos por unidad de tiempo)
ut = 5000        # unidades de tiempo por secuencia
div = 10000      # intervalos por unidad de tiempo (resolución temporal)
ts = ut * div    # intervalos por secuencia
bins = 8         # intervalos por ventana de muestreo
cod = math.ceil(math.log2(bins))    # bits de codificación
prob_bins = [0] * bins    # probabilidad de aparición en cada intervalo

# símbolos para representación
zero = '.'
one = '|'
```

```
# valores iniciales
t = 0            # intervalo de tiempo
tp = 0           # intervalo de aparición de un nuevo pulso
v_prev = -1      # ventana de aparición del pulso anterior
unos_bins = 0    # número total de '1' en la digitalización
ceros_bins = 0   # número total de '0' en la digitalización

# resultados
np = 0           # número de pulsos totales
bits_bins = 0    # bits totales de aleatoriedad en la digitalización
bits_line = 0    # bits totales de aleatoriedad por línea de archivo
bin_err = 0      # pulsos no digitalizados por aparición en la misma
                 # ventana de muestreo que el pulso previo

# generar secuencia, representar y digitalizar
while t < ts:
    u = random.random()             # variable aleatoria uniforme
    dt = -math.log(1.0 - u)/tasa    # tiempo entre pulsos
                                    # (en unidades de tiempo)
    dt = int(dt * div + 0.5)        # (en intervalos de tiempo)
    tp = tp + dt
    while t <= tp and t < ts:
        # saltos de línea para representación
        if t != 0:
            if t % 100 == 0:
                print('', file = fdist)
                if t % 1000 == 0:
                    print('', file = fdist)
        if bits_line >= 100:
            print('', file = fbins)
            bits_line = 0
        # si es un intervalo de aparición
        if t == tp:
            # representar aparición de pulso
            print(one, end = ' ', file = fdist)
            # incrementar número total de pulsos
            np = np + 1
            # número de orden del intervalo de aparición del pulso
            n = t % bins
            # ventana actual de muestreo
            v = t // bins
            # incrementar número de apariciones en intervalo
            prob_bins[n] = prob_bins[n] + 1
            # si el pulso aparece en la misma ventana que el anterior
            # ignorarlo e incrementar el contador de pulsos no digitalizados
            if v == v_prev:
                bin_err = bin_err + 1
            # de lo contrario
            else:
                # recordar ventana de aparición del pulso
                v_prev = v
                # codificar en binario el número de orden del intervalo
                print(dec2bin(n, cod), end = ' ', file = fbins)
                # actualizar bits totales de aleatoriedad
                bits_bins = bits_bins + cod
                # actualizar bits por línea de archiva
                bits_line = bits_line + cod
        else:
            # representar ausencia de pulsos
            print(zero, end = ' ', file = fdist)
        t = t + 1
```

```
# calcular probabilidades en intervalos
for i in range(bins):
    if np != 0:
        prob_bins[i] = round(prob_bins[i] / np, 3)
    else:
        prob_bins[i] = 0

# mostar resultados
print(f'{np / ut:.2f} pulsos/ut')
print(f'{bits_bins / ut:.2f} bits/ut')
print(f'{100 * bin_err / np:.2f} % pulsos en ventanas con pulsos previos')
print(f'{100*(unos_bins / (ceros_bins + unos_bins)):.2f} % "1"')
print("Probabilidades de aparición en intervalos de ventana de muestreo:")
print(prob_bins)
print("Probabilidad teórica:")
print(round(1 / bins, 3))

# cerrar archivos
fdist.close()
fbins.close()
```

3. Herramienta Web

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="description" content="Simulación y visualización de un TRNG
basado en SPAD">
        <meta name="keywords" content="TRNG, SPAD">
        <meta name="author" content="Javier Piay">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>TRNG_SPAD_1</title>
        <link rel="stylesheet" href="style.css">
    </head>
<body>

<script type= "text/javascript">

document.body.innerHTML =
`
<pre>
[Título]
Generación de números aleatorios basada en la
digitalización de los intervalos de muestreo donde
se detecta la aparición de los pulsos de avalancha
originados en un SPAD en condiciones de oscuridad.

[Datos]
- tasa: 1 pulso por unidad arbitraria de tiempo

[Variables]
- res: intervalos por unidad de tiempo (resolución temporal)
- bins: número de intervalos por ventana de muestreo

[Funcionamiento]
- Introducir los valores de "res" y "bins".
- Pulsar 'Generar' para ver los resultados de forma instantánea.
- Pulsar 'Animar/Detener' para ver los resultados de forma progresiva.

</pre>
<div>
      res:<input type="text" id="inp_res" value=10000>
</div><br>
<div>
      bins:<input type="text" id="inp_bins" value=16>
</div><br>
<button id="btn_seq" type="button">
      Generar
</button>
<button id="btn_ani" type="button">
      Animar
</button>
<br>
<p style="background-color:lightgrey;font-weight:bold;">
      1. Distribución de pulsos por unidad de tiempo
</p>
<p>
      - Tasa promedio: <span id="txt_tasa"></span>
</p>
```

```
<canvas id="can_ut"></canvas>
<br>
<p style="background-color:lightgrey;font-weight:bold;">
      2. Distribución de pulsos en intervalos de muestreo
</p>
<p>
      - Fracción de pulsos múltiples sin codificar: <span id="txt_mult"></span>
</p>
<canvas id="can_bins"></canvas>
<br>
<p id="txt_bins">
      [ ]
</p>
<p style="background-color:lightgrey;font-weight:bold;">
      3. Digitalización de la aleatoriedad
</p>
<p>
      Porcentaje de '0': <span id="txt_por0"></span>
</p>
<canvas id="can_seq"></canvas>
<br>
<br>
<div id="div_seq" style="background-color:black;color:white"></div>
`
//<div id="div_seq" style="font-family:courier;font-size:16px;background-
color:black;color:white"></div>

const txt_tasa = document.getElementById("txt_tasa");
const can_ut = document.getElementById("can_ut");
const ctx_ut = can_ut.getContext("2d")
const txt_mult = document.getElementById("txt_mult");
const can_bins = document.getElementById("can_bins");
const ctx_bins = can_bins.getContext("2d")
const txt_bins = document.getElementById("txt_bins");
const btn_seq = document.getElementById("btn_seq");
const btn_ani = document.getElementById("btn_ani");
const txt_por0 = document.getElementById("txt_por0");
const can_seq = document.getElementById("can_seq");
const ctx_seq = can_seq.getContext("2d")
const div_seq = document.getElementById("div_seq");

let res = document.getElementById("inp_res").value;
let bins = document.getElementById("inp_bins").value;
let digits = Math.ceil(Math.log2(bins));

const FPS = 50;
const tasa = 1;
const rows = 512 * 1;
const cols = 512;
const bitsize = 1;

let aniID, stopped,lastTimestamp;
let tp, put, ptot, ut, v_prev, mult;
let pbins = [];
let nceros, nunos;
let randbits, txtseq;

can_ut.width = 0.9 * window.innerWidth;
can_ut.height = 0.15 * window.innerHeight;
can_bins.width = 0.9 * window.innerWidth;
can_bins.height = 0.2 * window.innerHeight;
can_seq.width = bitsize * cols;
```

```
can_seq.height = bitsize * rows;

function ut_lin(){
	ctx_ut.clearRect(0, 0, can_ut.width, can_ut.height);
	ctx_ut.fillRect(0, can_ut.height, can_ut.width, -3);
}

function bins_lin(){
	ctx_bins.clearRect(0, 0, can_bins.width, can_bins.height);
	ctx_bins.fillRect(0, can_bins.height, can_bins.width, -3);
}

function poiss(){
	let v, bin, maxbin;
	let u = Math.random();
	let dt = -Math.log(1.0 - u) / tasa;
	dt = dt * res;
	tp = tp + dt;
	if(tp < res){
		put = put + 1;
		ptot = ptot + 1;
		ctx_ut.fillRect(tp * can_ut.width / res, 0, 1, can_ut.height);
		v = Math.floor(tp / bins);
		bin = Math.floor(tp) % bins;
		bins_lin();
		pbins[bin]++;
		maxbin = Math.max(...pbins);
		txt_bins.innerHTML = "[" + pbins + "]";
		if(v != v_prev){
			v_prev = v;
			drawseq(dec2bin(bin, digits));
		}
		else{
			mult = mult + 1;
		}
		txt_mult.innerHTML = mult + "/" + ptot + ' (' + Math.round(10000 * mult
/ ptot) / 100 + ' %)';
		for(let i = 0; i < bins; i++){
			ctx_bins.fillRect(2 + i * can_bins.width / bins,
can_bins.height, can_bins.width / bins - 4, -can_bins.height * pbins[i] / maxbin);
		}
	}
	else{
		txt_tasa.innerHTML = Math.round(10 * put) / 10;
		ut = ut + 1
		put = 0;
		tp = 0;
		ut_lin();
		v_prev = -1;
	}
}

function ani_poiss(timestamp) {
  aniID = requestAnimationFrame(ani_poiss);
  if (timestamp - lastTimestamp < 1000 / FPS)
	return;
  poiss();
  lastTimestamp = timestamp;
}

function ani_start(){
	btn_ani.innerHTML = "Detener";
```

```
	restart();
	stopped = false;
	ani_poiss();
}

function ani_stop(){
	if (aniID)
		window.cancelAnimationFrame(aniID);
	stopped = true;
	ut_lin();
	btn_ani.innerHTML = "Animar";
	txt_tasa.innerHTML = Math.round(100 * ptot / ut) / 100;
}

function btn_seq_click(){
	restart();
	while(randbits <= rows * cols){
		poiss();
	}
	ani_stop();
}

function btn_ani_click(){
	if(stopped == true)
		ani_start();
	else
		ani_stop();
}

function drawseq(binseq){
	const binarr = binseq.split('');
	for(let i = 0; i < binarr.length; i++){
		const bw = binarr[i] == 0 ? "black" : "white";
		ctx_seq.fillStyle = bw;
		ctx_seq.fillRect(bitsize * (randbits % cols), bitsize *
Math.floor(randbits / cols), bitsize, bitsize);
		randbits = randbits + 1;
		txt_por0.innerHTML = Math.round(10000 * (nceros/(nceros + nunos))) /
100 + ' %';
		txtseq += binarr[i] + ' ';
		if(randbits % 8 == 0){
			//area_seq.innerHTML = txtseq + '\r\n' + area_seq.innerHTML;
			div_seq.insertAdjacentHTML("afterbegin", '<span>' + txtseq +
'</span><br>');
			txtseq = '';
		}
	}
}

function dec2bin(dec, nbits){
	let digit, bits;
	if(dec == 0){
		bits = '0';
		nceros = nceros + 1;
	}
	else
		bits = '';
	while(dec > 0){
		digit = (dec % 2).toString();
		if(digit == '0')
			nceros = nceros + 1;
		else
```

```
                        nunos = nunos + 1;
                bits = digit + bits;
                dec = Math.floor(dec / 2);
        }
        let pad = nbits - bits.length;
        if(pad > 0){
                for(let i = 0; i < pad; i++){
                        bits = '0' + bits;
                        nceros = nceros + 1;
                }
        }
        return bits;
}

function restart(){
        res = document.getElementById("inp_res").value;
        bins = document.getElementById("inp_bins").value;
        digits = Math.ceil(Math.log2(bins));
        pbins = [];
        for(let i = 0; i < bins; i++)
                pbins[i] = 0;
        ut = 0;
        v_prev = -1;
        tp = 0;
        put = 0;
        ptot = 0;
        mult = 0;
        ut_lin();
        bins_lin();
        stopped = true;
        lastTimestamp = 0;
        nceros = 0;
        nunos = 0;
        randbits = 0;
        ctx_seq.fillStyle = "lightgray";
        ctx_seq.fillRect(0, 0, can_seq.width, can_seq.height);
        txtseq = '';
        div_seq.innerHTML = '';
}

btn_seq.addEventListener("click", btn_seq_click, false);
btn_ani.addEventListener("click", btn_ani_click, false);
restart();

</script>

</body>
</html>
```