Trabajo de Fin de Grado

Criptoanálisis mediante machine learning de funciones no clonables físicamente (PUFs)

Departamento de Ingeniería Electrónica y Comunicaciones

Autor:

Alejandro López Calvo

Directores:

Miguel García Bosque

Guillermo Díez Señorans

Facultad de Ciencias

Junio 2021

Agradecimientos

Me gustaría dar las gracias a mis directores, Miguel García Bosque y Guillermo Díez Señorans, por la atención, disponibilidad y ayuda que me han facilitado a lo largo del desarrollo de todo el trabajo. También querría agradecer al departamento de Ingeniería Electrónica y Comunicaciones el buen trato que han tenido conmigo. Para finalizar, me gustaría dar las gracias a mis padres, Antonio y Pilar; a mi hermana, Natalia; y a mi novia, María; por haberme apoyado y animado durante todo el grado.

Lista de acrónimos

PUF Physically Unclonable Function
NVM Non-Volatile Memory
ID Abreviatura de “Identification”
ReLU Rectified Lineal Unit
APUF Arbiter-based Physically Unclonable Function
CRP Challenge-Response Pair
VLSI Very Large-Scale Integration
FPGA Field Programmable Gate Array
AES Advanced Encryption Standard
CTR Counter
SHA Secure Hash Algorithm

Índice de figuras

Índice de tablas

Índice

1. Introducción

La tecnología de la información está en crecimiento, ya que facilita el intercambio y almacenamiento de bienes e información de forma continua. Pero a medida que este campo se va desarrollando, la dificultad de todos estos procesos va en aumento. Tal auge, junto con la globalización, han provocado que con la subcontratación de la producción a países extranjeros, así como con el envío de artículos y efectos fabricados por todo el mundo, sea cada vez más difícil de controlar los procesos correspondientes con cada uno de éstos.

Dada la gran cantidad de productos que, de manera progresiva, están a nuestro alcance, tenemos la perentoria necesidad de autenticarlos; pero dependiendo del procedimiento de autenticación del que dotemos al producto, puede resultar un proceso costoso. Análogamente, cada vez existe una mayor cantidad de datos sensibles para terceros y se incrementan las alternativas de ataque.

Es lógico pensar que, por todas estas razones, también haya aumentado la demanda de expertos en ciberseguridad. Existen muchos datos que deben ser protegidos en bases de datos y convertidos, por ende, en inaccesibles ante cualquier eventual amenaza o apropiación ilegítima de ellos: datos médicos, correos electrónicos, datos bancarios, información gubernamental, y muchos más. Cada vez con más frecuencia, se producen más ataques a toda clase de datos; y, a pesar de que las aplicaciones de ciberseguridad van mejorando y adaptándose de forma constante, los atacantes siguen encontrando formas de romper esas barreras de seguridad [1].

Una posible solución que puede ayudar a reducir o resolver los problemas de autenticación y seguridad son las llamadas funciones no clonables físicamente (PUF). Estas funciones utilizan las variaciones intrínsecas producidas durante el proceso de fabricación para generar unos identificadores únicos similares a huellas dactilares de dispositivos físicos. Entonces podremos identificar y autentificar chips y generar claves para fines criptográficos. Es decir, los productos podrán ser identificados por sus propiedades características individuales y por tanto autentificarse. Además, estas propiedades intrínsecas de cada producto pueden ser utilizadas en los canales de comunicación para legitimar la contraparte o para cifrar los mensajes que queramos transmitir. Esto solucionaría todos los problemas que hemos mencionado; ya que, aparte de controlar la cadena de producción de las mercancías, los clientes pueden verificar que los productos son originales: muy útil para productos de lujo, información relevante sobre algo o productos sensibles para la salud.

Existen distintos tipos de PUFs, siendo algunas de las más conocidas las PUFs de oscilador de anillo y las PUFs de árbitro (APUF). Desafortunadamente, pueden ser sensibles a ciertos tipos de ataque de modelado [6].

2. Objetivos

El objetivo de este trabajo será realizar un estudio sobre distintos ataques a las funciones no clonables físicamente o PUFs. Particularmente, se hará énfasis en los ataques de modelado basados en machine learning porque han resultado ser muy precisos frente a algunos tipos de PUFs. Específicamente, el estudio se centrará en las PUFs de árbitro, y en función de la seguridad que muestren frente a estos ataques se buscarán distintas soluciones que mejoren su robustez mediante técnicas de criptografía.

3. Función no clonable físicamente (PUF)

3.1. ¿Qué es una PUF?

Una PUF, o función no clonable físicamente [1], es una entidad que utiliza variaciones aleatorias e incontrolables producidas durante el proceso de fabricación para generar una salida única para cada dispositivo, que generalmente es un número binario. De forma muy simplificada: es la "huella dactilar"de un objeto.

Una PUF va a estar formada por distintos componentes definidos por variaciones de parámetros locales. Estos componentes van a ser distintos unos de otros y están intrínsecamente presentes desde el momento de su creación, como resultado de variaciones estocásticas producidas durante el proceso de fabricación, de modo que habrá diferencias entre ellos que llamaremos desajustes locales. Estos parámetros locales pueden combinarse, compararse o leerse directamente para generar una salida binaria. La variación de los componentes no se puede controlar, así que una PUF no puede replicarse.

La PUF requiere de una señal de entrada; por tanto, es una función. Esta señal influye en la salida, pero en la medida en la que lo hará dependerá de los distintos enfoques de PUF. A la entrada se le llama desafío y puede alterar la combinación interna de los componentes que no coinciden, por lo que cambia la salida, que llamaremos respuesta. La entrada también puede definir cuál de los componentes se tiene que utilizar para generar la salida.

Si analizamos el acrónimo de PUF tenemos:

- **Physical:** quiere decir que es algo físico, no un algoritmo o algo similar. Cuando decimos que es físicamente no clonable, nos estamos refiriendo a un concepto físico. No se puede replicar de una manera física.
- **Unclonable:** significa que no se puede replicar.
- **Function:** equivale a que, para un valor de entrada, tendremos un valor de salida específico. El problema es, que en la práctica, una entrada o desafío en una PUF produce distintas salidas porque la salida suele ser ruidosa.

Con todo lo visto hasta ahora, podemos generalizar la definición de PUF [1] como:

Una PUF es una entidad física que produce un valor de salida al menos en dependencia de estructuras físicas que son difíciles de clonar.

3.2. PUFs débiles y fuertes

Esta clasificación está basada en las propiedades de seguridad que tienen las PUFs. Para determinar si una PUF es fuerte o débil nos fijaremos en el comportamiento de su pareja desafío-respuesta.

Una PUF es fuerte si un adversario, tras haber dispuesto de acceso a ella durante un largo periodo de tiempo, es incapaz de predecir la respuesta a un desafío elegido al azar con una alta probabilidad. Esto implicará que:

- La PUF tiene un conjunto de pares desafío-respuesta muy elevado y no se puede ir comparando todos desafíos hasta ver cuál coincide.
- La PUF es impredecible.

Las que no cumplan estos dos requisitos se considerarán débiles.

En este estudio trabajaremos únicamente con PUFs fuertes porque sólo éstas son susceptibles de ser atacadas mediante machine learning, ya que las PUFs débiles son modelables trivialmente simplemente leyendo todos los pares de desafío-respuesta. Por consiguiente, de aquí en adelante cuando hablemos de una PUF deberemos considerar que se tratará de una fuerte.

3.3. Ventajas: seguridad y costes

Sabido es que la demanda en ciberseguridad aumenta constantemente, requiriendo soluciones en identificación, autentificación y encriptación. Para ello, se usan números únicos o impredecibles que se generan fuera del dispositivo y luego se guardan en una memoria no volátil (NVM), amén de tener que transferir las claves criptográficas en un entorno seguro. Todos estos procesos producen costes.

Sin embargo, las PUF generan un número a partir de las propiedades intrínsecas del dispositivo, por lo que la generación de números y su posterior almacenamiento no se realizan fuera de los chips, abaratando así los costes de producción. Esta característica hace a las PUF muy interesantes y las convierte en una tecnología competitiva. Entonces, utilizar una PUF en un producto puede ser ventajoso [1] porque:

- **Reduce costes:** los sistemas comunes generan la identificación única fuera del chip y luego se la transfieren, almacenándola en NVMs. Pero la producción de un chip

sin NVM es más barata. Utilizando una PUF, la identificación única está inherentemente disponible en el propio chip. Como la salida de la PUF suele tener un numero de bits muy grande comparado con el número de IDs requeridos, podemos suponer que la salida es única.

- **Aumento de la seguridad:** la probabilidad de predecir la salida de una PUF a una determinada entrada es extremadamente baja. Como los datos confidenciales se generan dentro del chip, no es necesario tener que generarlos en un entorno externo y seguro. Tampoco es necesario almacenar estos datos en NVMs. Por todas estas razones, los niveles de seguridad que podemos alcanzar son elevadísimos.

3.4. Tipos de PUF

Cada PUF debe tener características únicas que la distingan de otras instancias de su misma clase. Esta característica única se puede ver como una función booleana de entrada n bits y de salida m bits, tal que:

$$f : \{0,1\}^n \rightarrow \{0,1\}^m \tag{1}$$

Cada mapeo booleano deberá ser único para cada instancia y tendrá que ser constante en el tiempo a pesar de posibles variaciones ambientales o del entorno de ruido eléctrico [6]. El conjunto de n bits de entrada y la correspondiente salida de m bits es considerado como un "par de desafío-respuesta"(CRP).

Dos conceptos que resultan importantes al hablar de PUFs son unicidad y reproducibilidad. La unicidad hace referencia a que las características entre distintas instancias son únicas y la reproducibilidad a que los mapas booleanos, o tablas de verdad, son constantes en el tiempo o cuando varían las condiciones ambientales.

Generalmente, las PUFs poseen ambas características, pero en la práctica se han encontrado algunos casos concretos en los que las características entre instancias son similares (poca unicidad) o que los mapas booleanos no son constantes (poca reproducibilidad). Además, también son sensibles a diversos tipos de ataques, de los cuales los ataques de modelado son los más eficaces.

Para poder realizar un modelo computacional preciso contra una PUF, normalmente el atacante necesita tener acceso a un conjunto de pares desafío-respuesta, a partir de los cuales puede llegar a predecir la respuesta de cualquier desafío arbitrario con un gran porcentaje de acierto.

Aunque existen diversos tipos de PUF, en esta sección explicaremos únicamente dos de las arquitecturas más conocidas: la PUF de oscilador de anillo y la PUF de árbitro. Ésta última será la que usaremos en este trabajo, ya que un artículo [6] afirma que es la más susceptible a los ataque de modelado. Por tanto, en este estudio nos centraremos en

analizar la seguridad de las PUFs de árbitro frente a distintos tipos de ataque de modelado y en buscar soluciones de criptografía que permitan aumentar su seguridad.

3.4.1. PUF de oscilador de anillo

Un circuito PUF de oscilador de anillo [2] se compone de n osciladores de anillo (RO) distribuidos de manera idéntica, RO_1 a RO_n, con frecuencias f_1 a f_n respectivamente. Un par de frecuencias f_a y f_b, siendo $a \neq b$, de n salidas de oscilador en anillo, se seleccionan mediante un par de multiplexores, usando el desafío PUF como los bits de selección de los multiplexores. Debido a las variaciones intrínsecas producidas durante el proceso de fabricación, f_a y f_b tienden a diferir entre sí. Entonces, un bit de respuesta r_{ab} se produce mediante la cuantificación de dos cantidades de valor real, f_a y f_b, utilizando un método de comparación simple como:

$$r_{ab} = \begin{cases} 1 & \text{si } f_a > f_b \\ 0 & \text{para cualquier otro valor} \end{cases} \tag{2}$$

Podemos observar su funcionamiento de forma más precisa en el esquema de la figura 1:

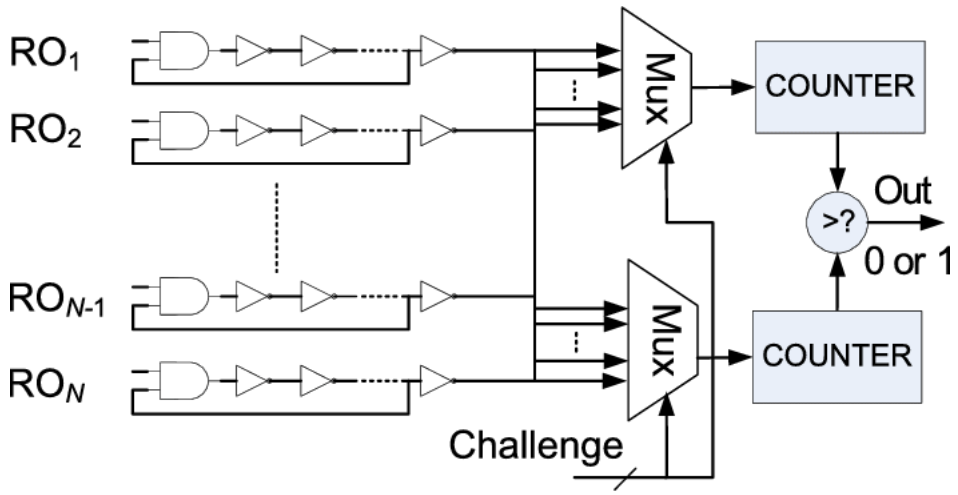


Figura 1: PUF de oscilador de anillo.

3.4.2. PUF de árbitro (APUF)

La APUF [6] es un tipo de PUF que basa su funcionamiento en una cascada de varias etapas de retardo de dos puertos estructuralmente idénticos, con un "árbitro" al final de la cascada. Para cada etapa de retardo, que son conmutadores de intercambio de ruta, la conectividad de los puertos de entrada a los de salida es controlada por un bit de control (por ejemplo, el i-ésimo bit de desafío $c[i]$ correspondiente a la i-ésima etapa). Los dos puertos de entrada de la primera etapa están en corto de modo que un pulso de

entrada puede propagarse por las dos rutas de retardo. Estos caminos de las dos señales dependerán del desafío aplicado. Por los efectos de la variación del proceso, a pesar de que el diseño sea idealmente idéntico, estas dos señales llegarán al árbitro después de retrasos ligeramente diferentes. Esto dará como salida un 0 lógico o un 1 lógico del árbitro.

Podemos visualizar este proceso en la figura 2:

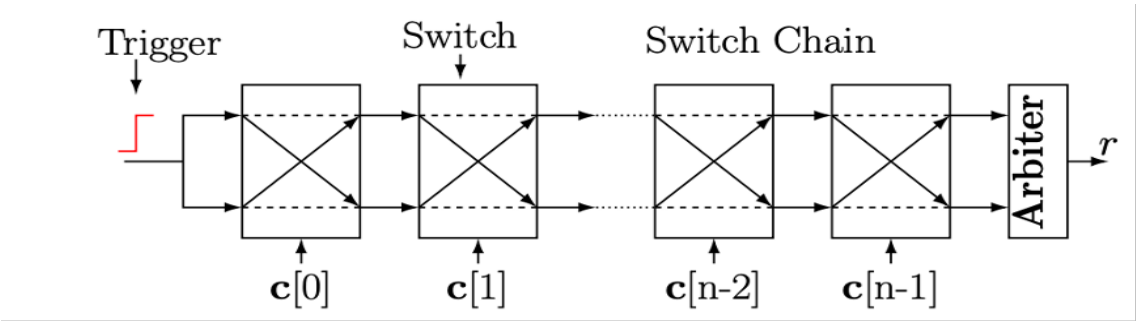


Figura 2: Arbiter PUF (APUF) de n bits.

Para cada desafío de n bits $c \in \{0,1\}^n$, la respuesta de la APUF vendrá dada por:

$$r = \begin{cases} 1 & \text{si } \Delta_c < 0 \\ 0 & \text{para cualquier otro valor} \end{cases} \tag{3}$$

siendo Δ_c la diferencia de retardo al final de las etapas de conmutación en cascada, cuyo valor viene dado por $\Delta_c = W^T\Phi$. W será el vector de peso, de dimensión $(n+1)$ números reales y cuyas componentes dependen del retardo de la ruta. Φ es el vector de paridad derivado del desafío dado c, cuyas componentes vienen dadas por:

$$\Phi[n] = 1 \tag{4}$$

$$\Phi[i] = \prod_{j=i}^{n-1}(1 - 2c[j]) \;\; ; \;\; i = 0, 1, ..., n-1 \tag{5}$$

Entonces, el modelado exitoso de una APUF se reduce a predecir el vector de peso W.

4. Redes neuronales

Antes de comenzar a explicar las redes neuronales, será necesario definir una serie de términos que forman la base del funcionamiento de estas redes.

- **Machine learning:** se define como el uso y desarrollo de sistemas informáticos capaces de aprender y adaptarse sin seguir instrucciones explícitas, mediante el uso de algoritmos y modelos estadísticos para analizar y extraer inferencias a partir de patrones en los datos.

- **Neurona artificial:** será el bloque fundamental de cualquier red neuronal artificial. Analizaremos su comportamiento comparándola con una neurona biológica: la unidad más básica de un cerebro biológico.

4.1. La neurona

El cerebro humano es capaz de procesar datos de una manera sorprendente y su estructura se basa en la unión de unidades elementales: las neuronas.

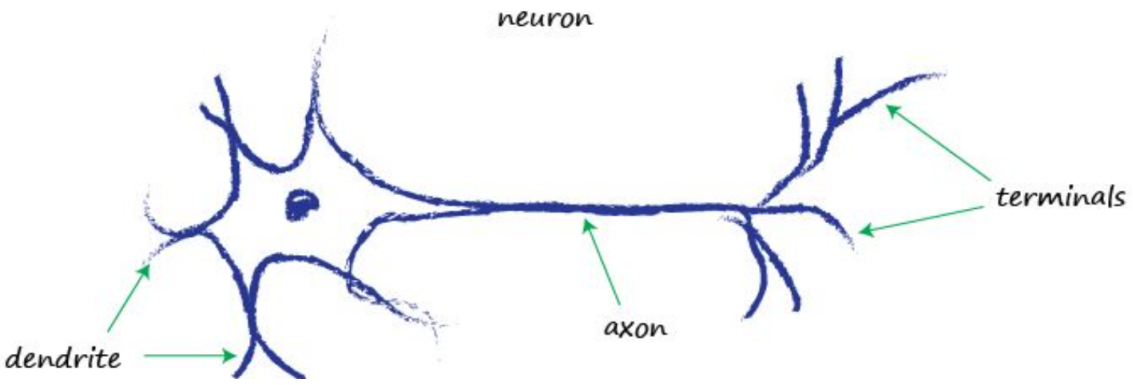


Figura 3: Diagrama de una neurona.

Una neurona es la unidad más básica de un cerebro biológico y se puede conectar a otras mediante impulsos eléctricos. Cada neurona recibe señales de otras a través de las dendritas y, en función de si la suma de las señales supera un determinado umbral, emitirá potencial de acción mediante el axón a otras neuronas. Es decir, según los distintos estímulos de entrada que reciba, la neurona emitirá un impulso o no.

El proceso de aprendizaje del cerebro humano se basa en modificar la intensidad de las conexiones entre las distintas neuronas de la red neuronal biológica para conseguir la respuesta más adecuada. Para una red neuronal artificial, formada por neuronas artificiales, los mecanismos de propagación de información y de aprendizaje serán los mismos.

4.2. Red neuronal artificial

El machine learning, o aprendizaje automático, es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan que las computadoras "aprendan". Una de las técnicas más comunes son las redes neuronales artificiales, que se tratan de sistemas de enlaces de neuronas que colaboran entre sí para producir un estímulo de salida. Por tanto, el objetivo será tratar de emular el funcionamiento de una red neuronal biológica a partir del funcionamiento de la unidad más básica: la neurona.

La red neuronal artificial estará formada por neuronas artificiales; en función de las entradas (representan a las dendritas), obtendremos una salida que se transmitirá a otra neurona (representa el potencial de acción emitido por el axón).

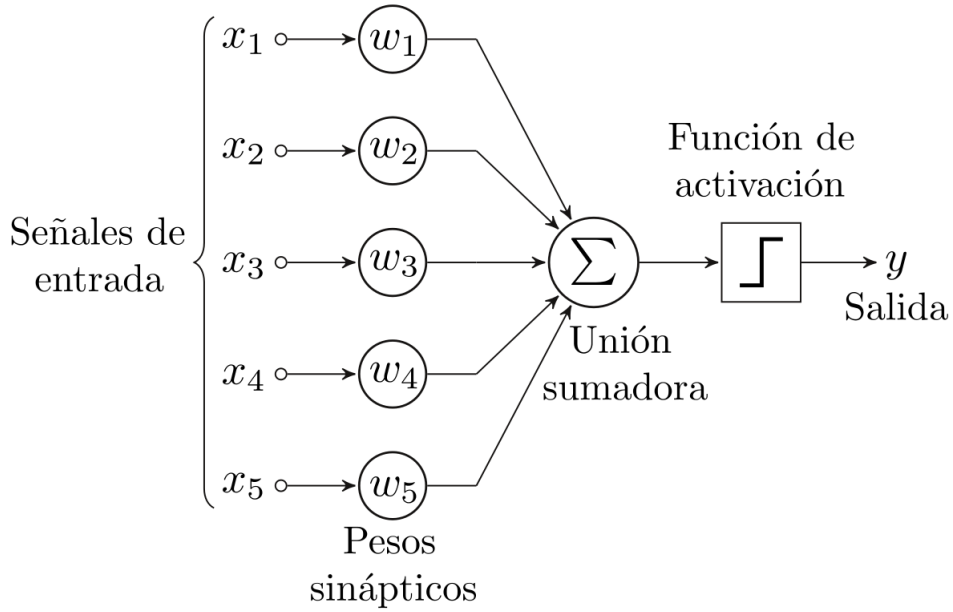


Figura 4: Esquema de una neurona artificial.

El funcionamiento básico [5] representado en la figura 4 viene dado por una serie de entradas, x_i con $i = 1, 2, 3, ..., m$, donde cada una se multiplica por su correspondiente peso sináptico w_i. A la suma de estos valores se le aplica una función de activación no lineal g para obtener la salida. Como una neurona biológica emite un estímulo o no dependiendo de si se alcanza un umbral, se ha añadido un término de sesgo w_0 que permitirá activar la función de activación o no:

$$y = g(w_0 + \sum_{i=1}^{m} x_i w_i) \tag{6}$$

Los pesos sinápticos harán referencia a la fuerza de unión entre las distintas neuronas. Al igual que el cerebro ajusta la intensidad entre las conexiones de las distintas neuronas para aprender, el entrenamiento o proceso de aprendizaje de una red neuronal artificial consistirá en ajustar los pesos sinápticos hasta conseguir la salida adecuada en función de las entradas.

La función de activación, que ofrece una salida dependiendo de las entradas y los pesos, tiene como objetivo introducir no linealidades en la red neuronal. Esto nos permite tratar con los datos de una manera más precisa porque los datos reales no suelen ser lineales.

Los modelos más comunes que se utilizan como función de activación no lineal son la función Sigmoide y la función ReLU:

$$\text{Función Sigmoide} \rightarrow g(z) = \frac{1}{1 + e^{-z}} \tag{7}$$

$$\text{Función ReLU} \rightarrow f(x) = max(0, x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{8}$$

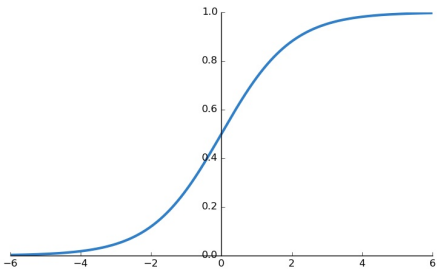


Figura 5: Función Sigmoide.

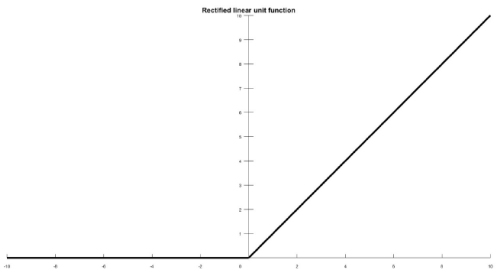


Figura 6: Función ReLU.

Tomando como entrada cualquier número real, la función Sigmoide da como salida un valor entre 0 y 1, de modo que es muy útil para problemas de probabilidades ya que éstas también tienen que tener un valor entre 0 y 1. Por otra parte, la función ReLU también es muy popular porque es muy simple: su valor es 0 en el cuadrante negativo y es la función de identidad en el cuadrante positivo.

Para terminar de crear la red neuronal habrá que diseñar su arquitectura. Ésta dependerá del tipo de problema que se quiera resolver, y en el caso de este estudio, se realizarán análisis con distintos diseños para observar diferentes comportamientos de la red. Aun así, todas arquitecturas tendrán la misma estructura básica:

- **Capa de entrada:** introduce los datos iniciales. Solo puede haber una y estará formada por tantas neuronas como datos de entrada haya.
- **Capas ocultas:** en función de la complejidad del problema podrá haber varias capas ocultas con varias neuronas o nodos en cada una de ellas. Estas capas se encargan de procesar los datos.
- **Capa de salida:** devuelve los datos finales. Solo hay una capa de salida y estará formada por tantas neuronas como salidas haya.

4.3. Entrenamiento de la red neuronal

Con el diseño de la arquitectura finalizado, la red ya es capaz de realizar predicciones. Para que sean precisas, primero debemos entrenarla. El entrenamiento es la fase más importante de la construcción del modelo y su objetivo es ajustar los pesos de la red neuronal para minimizar el error entre la salida de la red y la salida real, de modo que la red pueda "aprender"del problema.

4.3.1. Pérdida o coste

El parámetro que cuantificará cuánto de equivocada fue una predicción se llama coste o pérdida:

$$\mathcal{L}(f(x^{(i)}; W), y^{(i)}) \tag{9}$$

La función toma como entradas las salidas predichas ($f(x^{(i)}; W)$) y las salidas reales ($y^{(i)}$): si la diferencia es muy grande, entonces la pérdida será muy grande. De la misma manera, si los dos valores son cercanos, la pérdida será menor y la predicción más precisa.

El objetivo será siempre intentar minimizar la pérdida del modelo para que la predicción esté lo más cerca posible de la verdad.

Realmente, la función que queremos minimizar es la llamada pérdida empírica, que es el promedio de cada pérdida de cada ejemplo individual:

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x^{(i)}; W), y^{(i)}) \tag{10}$$

Si el problema tiene como salidas una clasificación binaria, es decir, la red neuronal sacará como repuesta un 1 o un 0, entonces la función de pérdida empírica que tomaremos será la pérdida de entropía cruzada binaria:

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}; W)) + (1 - y^{(i)}) log(1 - f(x^{(i)}; W)) \tag{11}$$

4.3.2. Optimización: descenso de gradiente

Para aproximarnos a la solución ideal deberemos emplear algoritmos de optimización de forma iterativa. El algoritmo que vamos a usar es el descenso de gradiente que se basa en la variación del error con respecto a la variación de los pesos. El objetivo será tratar de encontrar los pesos que minimizarán la función de pérdida empírica de nuestro conjunto de datos:

$$W^* = argmin_w \, (-\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}; W)) + (1 - y^{(i)}) log(1 - f(x^{(i)}; W))) \tag{12}$$

$$W^* = argmin_w \, J(W) \tag{13}$$

siendo W el conjunto de todos los pesos en nuestra red.

Cabe entender el proceso de optimización de una manera más sencilla a partir de un ejemplo:

Supongamos que tenemos una red neuronal con únicamente dos pesos: w_0 y w_1. Representamos en la figura 7 cuál es la pérdida empírica que esperaríamos obtener en función de cada peso:

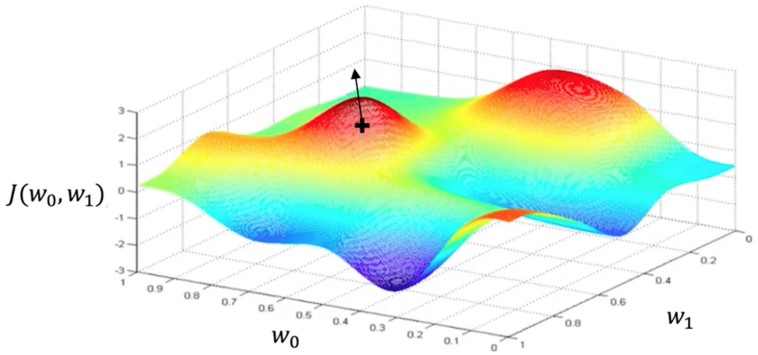


Figura 7: Dirección del gradiente.

La optimización será el uso de un algoritmo para encontrar el punto más bajo de la figura 7, correspondiente con la mínima pérdida empírica, que nos indicará los w_0 y w_1 óptimos.

Para realizar este proceso, elegiremos un punto de inicio, cualquiera. A partir de ese punto, calculamos el gradiente ($\frac{\partial J(W)}{\partial W}$), que nos indica la dirección del ascenso con mayor pendiente, esto es, por qué camino se sube.

Como el objetivo es encontrar la menor pérdida empírica, tomamos el negativo de ese gradiente y avanzamos un poco en esa dirección. Más claramente, como el gradiente nos indica la dirección de subida, tomamos la dirección opuesta para acercarnos al mínimo poco a poco. A continuación, realizamos el mismo procedimiento en el nuevo punto, que estará mas bajo que el anterior, y llegaremos a otro punto menor. Iteramos este proceso hasta converger en un mínimo:

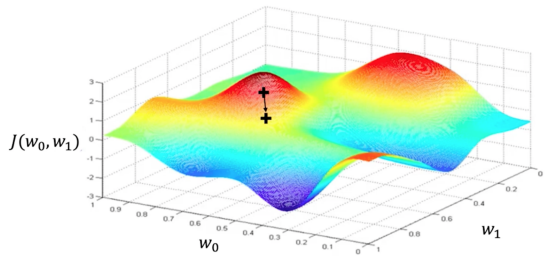


Figura 8: Pequeño paso en dirección opuesta al gradiente.

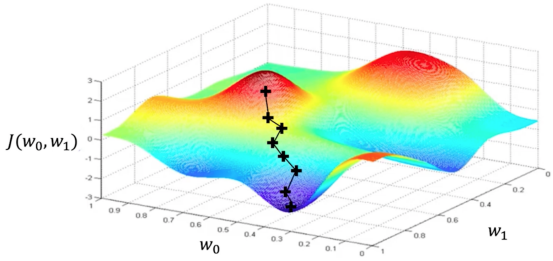


Figura 9: Proceso iterado hasta la convergencia en un mínimo.

Aunque no sabremos si hemos convergido en un mínimo local o global.

A este algoritmo se le conoce como descenso de gradiente. En resumen, iniciaremos los pesos aleatoriamente y realizamos un bucle hasta la convergencia: empezando en un punto inicial al azar, calculamos el gradiente que nos indica qué dirección está hacia arriba y daremos un pequeño paso en la dirección opuesta:

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W} \tag{14}$$

siendo η la tasa de aprendizaje.

Este ejemplo que hemos supuesto es relativamente simple, pero el proceso de entrenar una red neuronal en la práctica es un procedimiento más complejo. La función de pérdida empírica puede tener muchos mínimos locales y el descenso de gradiente puede quedarse atascado en uno de ellos.

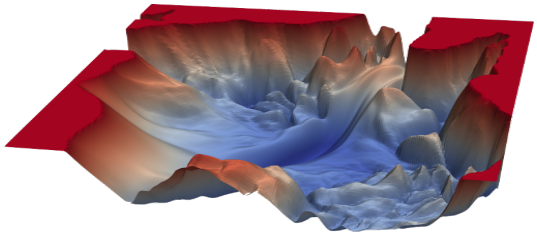


Figura 10: Función de pérdida empírica en función de los pesos de una red neuronal compleja.

Por consiguiente, encontrar el mínimo global puede ser una tarea muy sensible a los parámetros, especialmente al punto inicial del optimizador en este paisaje.

Fijémonos de nuevo en (14). La próxima actualización de peso será el peso actual menos la tasa de aprendizaje por el gradiente. La tasa de aprendizaje η cuantificará cómo de rápido queremos hacer nuestro aprendizaje, es decir, nos dirá cómo de grande debe ser cada paso que demos respecto al gradiente. Entonces, el gradiente nos indica la dirección y η nos indicará una escala de cuánto queremos confiar en ese gradiente. Si la tasa de aprendizaje es demasiado pequeña, nuestro modelo puede atascarse en algún mínimo local y nunca escapar de ahí, además de converger lentamente, por lo que serían necesarias muchas iteraciones y el tiempo de entrenamiento sería muy elevado. Ahora bien, si la tasa de aprendizaje es demasiado grande, puede que sobrepasemos los mínimos y que diverjamos.

Para establecer una tasa de aprendizaje que sea lo suficientemente grande como para evitar los mínimos locales pero que sea lo suficientemente pequeña para que no diverja y que pueda converger en el mínimo global, podemos aplicar distintos procedimientos.

El más sencillo sería probar con tasas de aprendizaje distintas. Este proceso funciona realmente, pero es muy tedioso. Uno óptimo sería diseñar una tasa de aprendizaje que aprenda del paisaje y se adapte. Ésta aumentará o disminuirá en función de la magnitud del gradiente en un punto o de lo rápido que estamos aprendiendo.

4.3.3. Conjuntos de entrenamiento y validación

Para finalizar, se debe determinar cómo se va a entrenar a la red neuronal. El número de etapas se refiere al número de veces que se trabaja con todos los datos de entrenamiento. Por el contrario, el número de iteraciones hace referencia al número de veces que se actualizan los pesos. Es importante que el número de iteraciones sea el adecuado para que

la red "aprenda" a resolver el problema, en vez de "memorizar" la solución de los datos de entrenamiento sin saber resolver el problema, ya que devolvería datos incorrectos para cualquier entrada que no fuese del conjunto de datos de entrenamiento. A este caso se le denomina sobreentrenamiento y para evitar que esto suceda se usará un conjunto de los datos como set de entrenamiento y otro conjunto como set de validación:

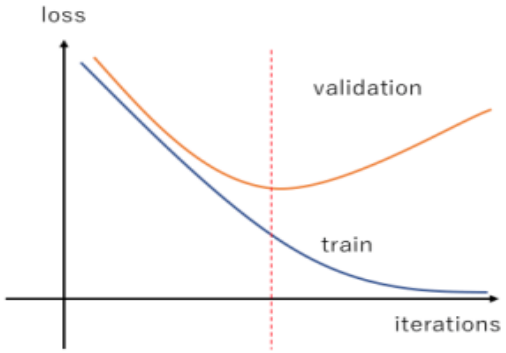


Figura 11: Comparación entre set de entrenamiento y set de validación.

En el límite que marca la línea discontinua, donde las curvas empiezan distanciarse, será donde la red neuronal estaría empezando a sobreentrenarse, de modo que habría que finalizar el número de iteraciones.

A continuación, veremos cómo aplicando estos conocimientos de redes neuronales podremos diseñar un ataque de modelado basado en machine learning para comprometer la no-clonabilidad de una PUF, es decir, para predecir todas sus respuestas a partir de un conjunto de CRPs dado.

5. Resultados

Las PUF fuertes han resultado ser sensibles a distintos ataques de modelado [6], por lo que la solidez frente a éstos es un factor fundamental para tales sistemas. En este estudio, realizaré distintos ataques de machine learning a una PUF basados en redes neuronales con Tensorflow. En concreto, usaré una Arbiter PUF (APUF) de 128 bits. Veremos que con una red neuronal de complejidad estructural relativamente baja podemos obtener una gran precisión de modelado.

Finalmente, en función de los resultados obtenidos, buscaré soluciones basadas en criptografía para mejorar la seguridad de la APUF frente a estos ataques de modelado.

Recordemos que las PUFs electrónicas son circuitos electrónicos basados en la aleatoriedad provocada por la variación en el proceso a nanoescala de su fabricación. Por ende, cada PUF debe tener una característica única que la distinga de otras instancias y que

vendrá dada por la función booleana:

$$f : \{0,1\}^n \rightarrow \{0,1\}^m \tag{15}$$

Esta función deberá ser única para cada PUF.

Como hemos visto anteriormente, son muy sensibles a los ataques de modelado, por lo que una APUF no se puede implementar sin procesar las respuestas debido a su vulnerabilidad. Aun así, tienen ventajas únicas, como su simplicidad en los principios de diseño y funcionamiento, la baja sobrecarga de hardware o su fácil implementación de VLSI.

Para los experimentos, el conjunto de pares de desafío-respuesta que usaré es una recopilación de una simulación en Matlab de una PUF [6], ya que las implementaciones de APUF basadas en FPGA tienen muy poca unicidad. La técnica que emplearé para atacar la APUF simulada de 128 bits utilizará únicamente vectores de paridad derivados del desafío como entrada, y no requeriremos de ninguna información adicional como información de canal lateral o relacionada con la reproducibilidad. Sólo necesitaremos conocer la estructura de la PUF y disponer de CRPs para realizar el ataque.

5.1. Machine learning para el modelado de una APUF

En este estudio, la red neuronal la programaremos con la biblioteca de código abierto para aprendizaje automático desarrollada por Google: TensorFlow-Keras.

Una red neuronal utiliza funciones de activación para aprender mapeos y patrones complejos de los datos. Esta función define la relación entre los valores de entrada y sus valores de salida. Una de las funciones de activación más usada es la función Sigmoide, acotada en el rango $[0, 1]$. Tiene una derivada positiva en cada punto y ésta se puede calcular de manera sencilla, pero sufre el efecto de desaparición de gradiente al aumentar su rango: el gradiente se irá desvaneciendo a valores muy pequeños, impidiendo el cambio de valor del peso.

También conocemos la función de activación ReLU, que permite que la red obtenga de manera sencilla representaciones dispersas y como es lineal por partes no sufrirá el efecto de desaparición de gradiente. Pero ReLU sólo se puede usar en las capas ocultas.

Por lo tanto, en este estudio usaremos una red neuronal con función de activación Sigmoide para la capa de salida y la función de activación ReLU para las capas ocultas.

Por añadidura, sabemos que la red irá ajustando los pesos con el objetivo de minimizar la pérdida. La convergencia al mínimo global es sensible a los pesos iniciales asignados. De este modo, nosotros inicializaremos aleatoriamente los pesos basándonos en una distribución uniforme que se normaliza en función del tamaño de la entrada y salida (*Glorot uniform* en Keras). Usaremos el optimizador *Adam* para actualizar los pesos y por último usaremos la función de entropía cruzada binaria como función de pérdida, que recordemos

que venía dada por la siguiente expresión:

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)}; W)) + (1 - y^{(i)}) log(1 - f(x^{(i)}; W)) \tag{16}$$

siendo $y^{(i)}$ las respuestas reales, $f(x^{(i)}; W)$ las pronosticadas y n el número de respuestas consideradas. El esquema de la red neuronal será el de la figura 12:

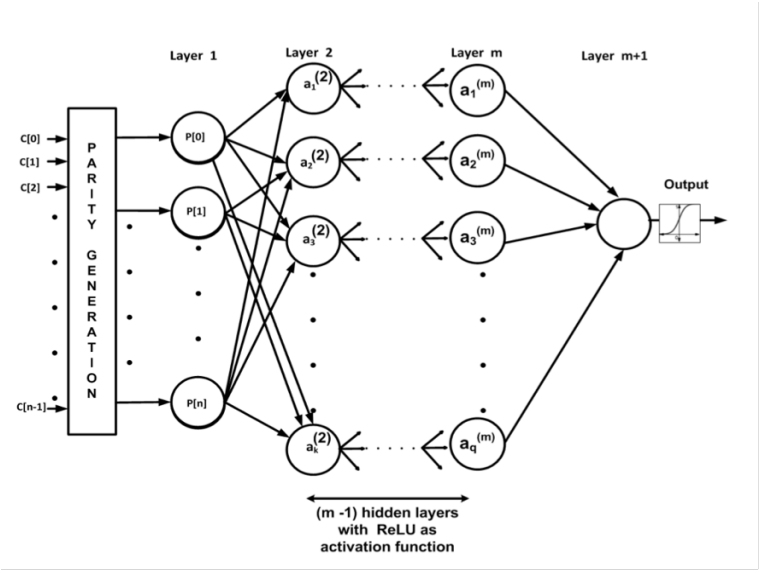


Figura 12: Diagrama de la red neuronal.

5.2. Configuración del ataque de modelado

Los parámetros que hemos usado en nuestra red para realizar los ataques de modelado se pueden observar en la tabla 1:

Parámetros	**Valores**
Inicializador Kernel	*Glorot uniform*
Tasa de aprendizaje	*0.001*
Inicializador Bias	*Ceros*
Optimizador	*Adam*
Función de pérdida empírica	*Entropía cruzada binaria*
Función de activación de las capas ocultas	*ReLU*
Función de activación de la capa de salida	*Sigmoide*

Tabla 1: Parámetros utilizados.

Los parámetros de retardo de etapa de la APUF son independientes e idénticamente distribuidos. Puesto que la reproducibilidad de una PUF real no es perfecta, se ha introducido un ruido aleatorio aditivo que se distribuye como $\mathcal{N}(0, 0.01)$. La variante de

la PUF será una APUF de 128 bits y los desafíos se han generado de forma aleatoria en conjuntos de medio millón. Como entrada usaremos vectores de paridad, los cuales se correlacionan linealmente con la diferencia de retardo de la APUF según (5). Así, hemos usado estos vectores de paridad junto con las respuestas correspondientes como entrada para la arquitectura de machine learning con el objetivo de capacitación y prueba.

El 80 % del conjunto de CRPs se usó como conjunto de entrenamiento y el 20 % restante como conjunto de validación.

La red neuronal de machine learning usada para el modelado ha sido implementada en Python 3.7.10, en el marco de Keras, con el backend de TensorFlow.

5.3. Análisis de los resultados experimentales

Capas ocultas	**Nodos por capa**	**CRPs de entrenamiento**	**Tiempo de entrenamiento (s)**	**Precisión de la predicción (%)**
1	3	400000	634	**99,86**
1	3	50000	151	**99,83**
1	3	20000	90	**99,60**
1	3	10000	40	**99,35**
1	3	8000	48	**98,62**
1	3	5000	67	**98,40**
1	10	10000	46	**97,90**
1	5	10000	48	**98,60**
1	2	10000	47	**99,25**
3	3	10000	48	**98,70**
2	3	10000	50	**99,25**

Tabla 2: Resultados del ataque de modelado a la APUF.

Podemos comprobar conforme a los resultados obtenidos en la tabla 2 que con arquitecturas de machine learning relativamente simples podemos llegar a obtener unas predicciones muy precisas. En concreto, con una única capa oculta con 3 neuronas ha sido suficiente para modelar la APUF de 128 bits. Además, esta precisión de modelado es muy alta incluso tomando conjuntos de entrenamiento relativamente pequeños.

Hemos ejemplificado que, efectivamente, las PUFs fuertes pueden resultar muy sensibles a los ataques de modelado. Al ser éste su principal inconveniente, nos enfocaremos en el resto del trabajo en investigar y analizar distintas técnicas que puedan mejorar notablemente su seguridad con el fin de poder considerar a las PUFs una tecnología útil y competitiva.

6. Criptoanálisis: Robustez ante técnicas de criptografía

En vista de que podemos modelar una APUF con una alta precisión con una estructura computacional de complejidad relativamente baja, nos proponemos diseñar alguna solución para aumentar su nivel de seguridad. Particularmente, nos basaremos en la criptografía, que es el conjunto de técnicas matemáticas que se encargan de garantizar la seguridad de la información mediante la alteración de representaciones lingüísticas de ciertos mensajes con el objetivo de hacerlos ininteligibles a observadores no autorizados.

Explicaremos tres modos de cifrado muy populares que son los que utilizaremos posteriormente en nuestro estudio.

Al enmascarar las respuestas de la PUF, esperamos que la red neuronal no sea capaz de predecirlas, mejorando notablemente la seguridad de la PUF.

6.1. Operación XOR de las respuestas

La primera técnica que usaremos no será una criptográfica como tal porque primero queremos comprobar cómo de precisa es la red neuronal ante cambios en las respuestas, que podríamos asemejarlo a una pequeña introducción de ruido en nuestro conjunto de datos; o sea, cuál sería el impacto de un error en el conjunto de CRPs de entrenamiento.

6.1.1. Aumento de la seguridad de la APUF mediante la operación XOR de las respuestas

El proceso es el siguiente:

Sean $r[i]$ el conjunto de todas las respuestas, con $i = 1, 2, .., 500000$. El nuevo conjunto de respuestas vendrá dado por la expresión:

$$r^{'}[i] = r[i] \oplus r[i+1] \tag{17}$$

El operador $\oplus$ expresa la operación XOR. Podemos ver cuál es su tabla de verdad:

Input		**Output**
$r[i]$	$r[i+1]$	$r[i] \oplus r[i+1]$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3: Operación XOR.

Las nuevas respuestas serán una combinación de dos respuestas originales, por lo que también deberemos combinar los desafíos de entrada. La entrada que meteremos ahora vendrá dada por:

$$c_i = c_i + c_{i+1} \tag{18}$$

donde el símbolo $'+'$ significaría en este contexto la yuxtaposición de los bits de los dos desafíos. Antes, cada desafío c_i estaba formado por 128 bits. Ahora, al combinarlos por parejas, los nuevos desafíos que usaremos como entradas tendrán una dimensión de 256 bits.

6.1.2. Análisis de los resultados de la operación XOR de las respuestas

Con esta modificación, obtenemos los siguientes resultados en el modelado de la APUF:

Capas ocultas	**Nodos por capa**	**CRPs de entrenamiento**	**Tiempo de entrenamiento (s)**	**Precisión de la predicción (%)**
1	3	10000	77	**71,25**
1	3	20000	78	**73,50**
1	3	35000	116	**75,41**
1	3	36000	118	**97,76**
1	3	50000	156	**97,85**
1	3	400000	945	**99,00**
1	5	10000	50	**89,65**
1	5	30000	77	**96,03**
1	5	36000	153	**98,68**
1	10	10000	51	**89,85**
1	10	30000	83	**96,72**
1	10	36000	150	**98,81**
4	40	10000	65	**60,20**
4	40	36000	214	**92,47**
4	40	50000	236	**94,19**

Tabla 4: Resultados del ataque de modelado a la APUF cifrada mediante operación XOR.

Podemos observar según los resultados recogidos en la tabla 4 que, cogiendo conjuntos de CRPs relativamente pequeños, la red ya no consigue modelar la APUF con la misma precisión que podía antes. Este método no consigue aumentar lo suficiente la seguridad porque con estructuras computacionales relativamente simples obtenemos unos porcentajes del $70 - 75\,\%$ de precisión. Análogamente, si aumentamos el conjunto de CRPs de entrenamiento sin variar la arquitectura de la red, conseguimos realizar un modelado exitoso con una alta precisión. También podemos observar que si dotamos a nuestra red

neuronal de una estructura más compleja y profunda, como pueden ser más nodos en la capa oculta o incluso añadir más capas ocultas con un número elevado de nodos, la red neuronal consigue entrenarse mejor y finalmente podemos obtener un modelado de alta precisión.

En conclusión, este método para mejorar la seguridad de la PUF no ha resultado ser el mejor, ya que con ataques de modelado basados en machine learning se puede obtener una precisión en la predicción de las respuestas muy elevada. A continuación, recurriremos a técnicas criptográficas para intentar obtener un enmascaramiento exitoso de nuestras respuestas de la APUF.

6.2. AES: Advanced Encryption Standard

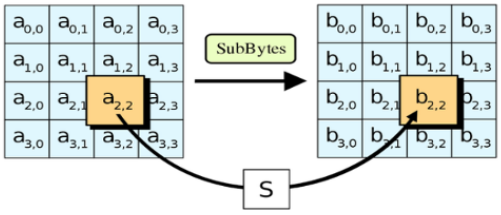
También conocido como Rijndael, es un modo de cifrado por bloques [3]. Tiene tres variantes: de 128 bits, de 192 bits y de 256 bits. En este estudio nosotros usaremos la variante de 128 bits: el AES-128. Se trata de un sistema de clave simétrica; a saber, la clave usada para el cifrado y para el descifrado debe ser conocida por el emisor y el receptor, lo que lo hace más seguro y además requiere de menos potencia computacional para enviar la información porque siempre va segura.

A continuación, explicaremos el funcionamiento de este algoritmo criptográfico, así como el del modo de operación CTR, y analizaremos los resultados obtenidos.

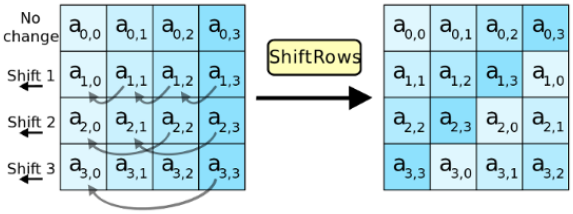
6.2.1. AES-128 en modo CTR

El segundo método de cifrado que usaremos será el algoritmo criptográfico AES-128. En principio, este algoritmo sirve únicamente para encriptar un bloque fijo de bits (para el AES-128 el tamaño de este bloque será de 128 bits). Si queremos encriptar una secuencia mayor como es nuestro caso (500000 respuestas), habrá que iterar el AES 500000/128 veces. Existen diversos modos de realizar este proceso, llamados modos de operación, y el que nosotros usaremos será el modo de operación CTR (Counter Mode).

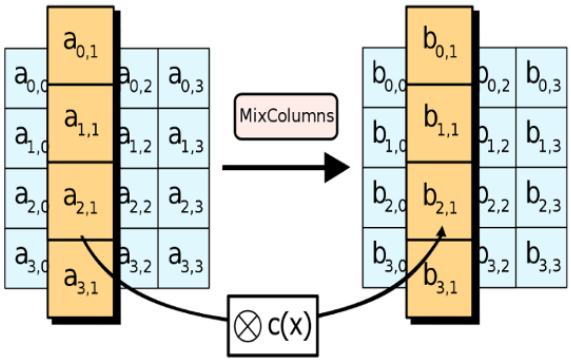
El funcionamiento del algoritmo se basa en la formación de bloques de 128 bits que se dividen en una matriz de dimensión 4x4 de 16 bytes. Así, 8 bits por byte nos dan los 128 bits. Cuando cifremos, el tamaño de la matriz no varía por lo que el tamaño del texto que queremos cifrar tampoco lo hará. Es un sistema de sustitución y permutación que consta de una ronda inicial de expansión de clave y un bucle de 4 pasos que se realizará 10 veces y que veremos a continuación:



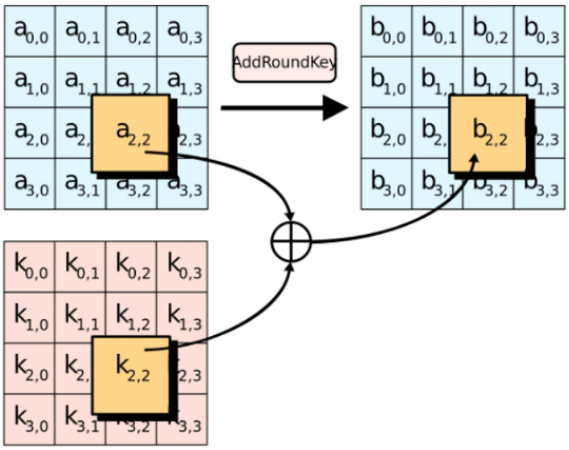
Paso 1: Cada byte es reemplazado con su entrada en una tabla de búsqueda fija por un valor alternativo indescifrable.



Paso 2: Los bytes de cada fila se mueven hacia la izquierda un número de lugares que difiere de una fila a otra.



Paso 3: En este paso, cada columna se multiplica por un polinomio constante c(x).



Paso 4: Cada byte se combina con un byte de la subclave mediante una operación XOR.

Figura 13: Pasos del algoritmo AES-128.

Se realizará este bucle 10 rondas.

6.2.2. CTR: Counter Mode

El AES-128 que hemos explicado en el apartado anterior se ha utilizado en modo CTR, que es un modo de operación que convierte un cifrado de bloque en uno de flujo. Su funcionamiento se muestra en la figura 14:

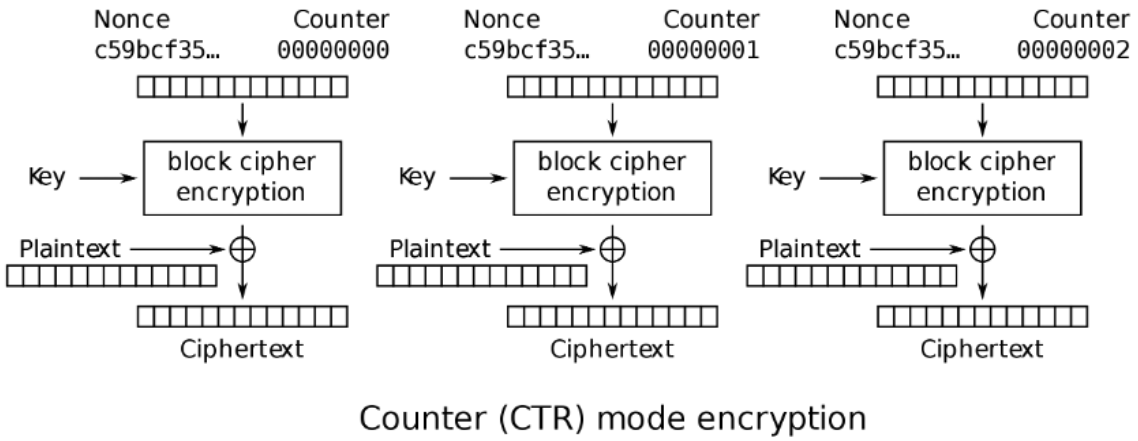


Figura 14: Modo de encriptación Counter Mode (CTR).

Como podemos observar, se genera un bloque de flujo de claves cifrando valores sucesivos de un contador. Usaremos una clave (*nonce*) combinada con el contador mediante una operación invertible para producir un bloque de contador único para el cifrado. Para cifrar el bloque de contador (*block cipher encryption*) usaremos el método de cifrado AES que hemos visto antes.

Finalmente, combinamos el *plaintext* (texto que queremos cifrar) mediante una operación XOR con el contador cifrado, dando como resultado el *ciphertext* o texto cifrado.

6.2.3. Aumento de la seguridad de la APUF mediante AES-128 en modo CTR

El proceso será el siguiente: iremos encriptando con el AES (podremos variar desde 1 a 10 rondas para analizar distintos niveles de complejidad del cifrado: a más rondas, mejor encriptado y mayor seguridad) números enteros consecutivos combinados con una clave de 128 bits (*nonce*). A continuación, combinaremos estos números encriptados con los bits de las respuestas mediante una operación XOR. Así habremos conseguido enmascarar todas las respuestas en bloques de 128 en 128 y, según el número de rondas que usemos, éstas serán más o menos seguras. Variando el número de rondas lo que estamos haciendo es modificar el comportamiento del bloque *block cipher encryption*. Originalmente en AES ese bloque hace 10 rondas, pero nosotros analizaremos la precisión de nuestra red neuronal desde 1 a 10 rondas para estudiar la seguridad de nuestra APUF.

6.2.4. Análisis de los resultados del AES-128 en modo CTR

Los resultados obtenidos en función del número de rondas empleado en el AES son los mostrados en la tabla 5:

Rondas AES	**Capas ocultas**	**Nodos por capa**	**CRPs de entrenamiento**	**Tiempo de entrenamiento (s)**	**Precisión de la predicción (%)**
1	1	3	50000	47	**49,86**
1	1	10	50000	113	**50,19**
1	1	10	400000	957	**51,23**
1	4	40	10000	89	**51,00**
1	4	40	50000	209	**50,86**
1	4	40	400000	1116	**50,45**
...	...	...	...	...	**...**
10	1	3	50000	149	**50,36**
10	1	10	50000	178	**50,49**
10	1	10	400000	1134	**51,10**
10	4	40	10000	91	**50,45**
10	4	40	50000	224	**51,36**
10	4	40	400000	1174	**51,68**

Tabla 5: Resultados del ataque de modelado a la APUF cifrada mediante AES en modo CTR.

Estos resultados son muy interesantes porque, teóricamente, el AES-128 debe realizar siempre 10 rondas para encriptar la información. No obstante, aquí podemos comprobar que para el caso de una APUF, con solamente una ronda estaríamos consiguiendo un método para que la PUF no pueda ser modelada y sea segura, ya que la red no consigue predecir las respuestas de ninguna manera (por mera probabilidad, un 50 % de precisión es lo mínimo que se puede obtener para conjuntos de datos grandes, ya que sólo hay 2 posibles respuestas: 0 o 1). Independientemente de la cantidad de CRPs de entrenamiento elegidos, la red neuronal no ha sido capaz de modelar con éxito; ni con una estructura sencilla como puede ser una capa oculta con 3 nodos, ni con una capa oculta con varios nodos, ni con varias capas ocultas con múltiples nodos.

Por tanto, hemos podido encontrar una solución para aumentar la robustez de la APUF y conseguir que sea segura frente a ataques de modelado.

6.3. SHA-256

EL SHA-256 [3] es una función hash criptográfica, es decir, es un algoritmo que transforma un conjunto de datos en un único valor de longitud fija: el hash. Este método de cifrado ofrece un nivel muy alto de seguridad.

Sus características mas relevantes son:

- Funciona en una única dirección: podemos generar el hash de cualquier contenido, pero de un hash no podemos generar el contenido asociado a él. Como es irreversible, puede ser libremente distribuido o almacenado porque únicamente se usa para fines de comparación.
- Presenta un equilibrio entre seguridad y coste computacional.
- Independientemente de la extensión del contenido que queramos cifrar, la longitud del hash resultante será siempre la misma: una cadena de 64 símbolos de 4 bits cada uno.

6.3.1. Aumento de la seguridad de la APUF mediante SHA-256

Ésta será la tercera técnica de criptografía que estudiaremos para enmascarar las respuestas de la APUF.

Hemos visto que el algoritmo SHA-256 transforma un conjunto de datos en un único valor de longitud fija llamado hash. En este caso, el hash tendrá una longitud de 256 bits. El procedimiento que realizaremos con este método de cifrado será el siguiente:

- Cogemos grupos de n desafíos con $n \leq 256$. En concreto, escogemos grupos de $n = 10$ desafíos. Cada desafío nos proporciona un bit de respuesta, por lo que tendremos vectores de respuestas de 10 bits.
- Aplicamos la función hash a esos vectores, obteniendo vectores de 256 bits.
- Cogemos los 10 primeros bits de esos vectores y los tomamos como si fuesen las respuestas a los desafíos.

6.3.2. Análisis de los resultados del SHA-256

Los resultados del ataque de modelado se pueden observar en la tabla 6:

Capas ocultas	**Nodos por capa**	**CRPs de entrenamiento**	**Tiempo de entrenamiento (s)**	**Precisión de la predicción (%)**
1	3	10000	40	**52,60**
1	3	100000	186	**54,11**
1	3	400000	694	**55,15**
1	10	400000	696	**54,33**
4	40	100000	205	**53,00**
4	40	400000	732	**52,87**

Tabla 6: Resultados del ataque de modelado a la APUF cifrada mediante SHA-256.

Al igual que en el caso del AES-128, vemos que la red neuronal no puede predecir las respuestas, independientemente de la complejidad o profundidad de su arquitectura o del tamaño del conjunto de CRPs que estemos tomando. En consecuencia, habremos encontrado una segunda forma de aumentar la seguridad de la APUF para que sea inmune a los posibles ataques de modelado.

7. Conclusiones

Las PUF son muy sensibles a los ataques de modelado basados en machine learning y su solidez ante éstos es un factor fundamental para que sean sistemas fiables.

En este trabajo hemos demostrado que la PUF de árbitro se puede clonar matemáticamente. Con una red neuronal de una complejidad computacional relativamente baja, podemos predecir más del 99 % de las respuestas. Para garantizar la seguridad de estas PUFs, hemos propuesto y analizado diversos métodos para aumentar la seguridad de las respuestas mediante criptografía y los resultados han sido prometedores.

Si las enmascaramos con una operación XOR, una simple red neuronal formada por una capa oculta y tres nodos es capaz de predecir más del 70 % de las respuestas con tan solo 10000 CRPs. Si aumentamos el conjunto de CRPs o la complejidad de la red neuronal, conseguimos unas predicciones con más del 97 % de éxito, por lo que esta técnica no consigue proteger lo suficiente las respuestas.

Por otro lado, si aumentamos la robustez de nuestras respuestas con algoritmos criptográficos como el AES-128 en modo CTR o el SHA-256, observamos que la red neuronal no conseguirá predecir las respuestas, independientemente de lo compleja o profunda que sea su arquitectura o del tamaño de CRPs que estemos considerando. Obtendremos una precisión en torno al 50 %, que es la probabilidad teórica mínima, ya que sólo hay dos posibles respuestas que predecir: 0 o 1. Por tanto, habremos conseguido enmascarar nuestras respuestas con éxito y ahora nuestra PUF será segura frente a los ataques de modelado.

En este trabajo, los dos algoritmos criptográficos han resultado ser completamente seguros, pero el AES-128 requerirá de menos recursos computacionales ya que una única ronda es suficiente para proteger la PUF; además, su implementación será mas sencilla. Sin embargo, como tiene una estructura matemática muy ordenada, es difícil garantizar que sea suficientemente seguro si un atacante emplea una red neuronal más potente y encuentra la manera de explotar esa estructura.

A su vez, el SHA-256 también es rápido de computar, es resistente a ataques de preimagen (ataques que intentan encontrar un mensaje que tiene un valor hash específico) y es resistente a colisiones (cuando dos entradas distintas a una función de hash producen la misma salida); por consiguiente, su seguridad frente a ataques de modelado más potentes será mayor. La desventaja que presenta es que al ser unidireccional, una vez transformados los datos, éstos ya no se pueden recuperar a partir del hash, por lo que se usará

únicamente para fines de comparación. A pesar de esta limitación, el SHA-256 resultaría ser la mejor opción de las tres propuestas para proteger la PUF.

Debemos tener en cuenta que el campo de la informática y la criptología está en constante desarrollo, lo que significa que todos los algoritmos criptográficos pueden ser vulnerados en algún momento. En consecuencia, para implementar las PUFs en los productos de una manera segura y convertirlas en una tecnología interesante y competitiva, éstas deberán ir acompañadas de un nivel de ciberseguridad que se vaya actualizando acorde al de, como ya he destacado en la introducción, las eventuales amenazas que se puedan producir.

Para finalizar, si bien este trabajo se ha basado en ataques a las PUFs de árbitro, los métodos propuestos también serían efectivos para proteger otros tipos de PUF, tales como la PUF de oscilador de anillo u otras más complejas.

Bibliografía

[1] Christoph Böhm, Maximilian Hofer (auth.). *Physical Unclonable Functions in Theory and Practice*. Springer-Verlag New York (2013).

[2] Roel MAES. *Physically Unclonable Functions: Constructions, Properties and Applications*. 2012.

[3] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Handbook of applied cryptography.*

[4] *ML-Make Your Own Neural Network*

[5] Michael Nielsen. *Neural Networks and Deep Learning.*

[6] Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty. *Deep Learning based Model Building Attacks on Arbiter PUF Compositions.*

[7] Charles Herder, Meng-Day (Mandel) Yu, Farinaz Koushanfar, and Srinivas Devadas. *Physical Unclonable Functions and Applications: A Tutorial.*

[8] Srini Devadas. *Practical Applications of Physical Unclonable Functions (PUFs).*

[9] Helena Handschuh, Geert-Jan Schrijen, and Pim Tuyls. *Hardware Intrinsic Security from Physically Unclonable Functions.*

[10] G. Edward Suh, Srinivas Devadas. *Physical Unclonable Functions for Device Authentication and Secret Key Generation.*

[11] Guillermo Díez-Señorans, Miguel Garcia-Bosque, Carlos Sánchez-Azqueta, Santiago Celma. *Funciones físicamente no-clonables como primitivas de seguridad en tecnología CMOS.*