Análisis del costo computacional de la criptografía en microcontroladores

Cristian Manuel Sierra Jerez y Abad Plata Vera

Trabajo de Grado para Optar al Título de Ingeniero Electrónico

Director

Jaime Guillermo Barrero Pérez

Magíster en potencia eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones E3T

Bucaramanga

2025

Agradecimientos

Cristian Sierra:

A Dios, por haberme permitido la oportunidad de estudiar y culminar esta etapa tan importante de mi vida, brindándome sabiduría y salud en cada momento. A mis padres, por su amor incondicional, su ejemplo constante y su apoyo en cada decisión que he tomado, siendo mi gran motivación. A mi hermano, por ser ejemplo de disciplina y compromiso. A Valentina, por su amor y compañía a lo largo de este proceso. A mis compañeros, quienes con su amistad y colaboración hicieron de esta experiencia algo memorable y enriquecedor, lleno de aprendizajes y buenos momentos.

Abad Plata:

A Dios, por haberme brindado la vida, la sabiduría y la fortaleza necesarias para superar cada desafío y llegar hasta este momento tan significativo. A mis padres, por su amor incondicional, su ejemplo y su apoyo constante, que han sido el motor de mi perseverancia y el fundamento de mis logros. A mi familia, por acompañarme con palabras de aliento, comprensión y cariño durante todo este proceso, recordándome siempre la importancia de seguir adelante con fe y determinación. A mis amigos, por su compañía, por compartir conmigo los momentos de esfuerzo y alegría, y por hacer de este camino una experiencia inolvidable.

A todos ustedes, dedicamos con profunda gratitud este trabajo, fruto de años de aprendizaje, dedicación y esfuerzo que simboliza no solo el cierre de una etapa, sino también el inicio de nuevos retos y oportunidades en nuestras vidas profesionales.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

Apéndice A. Video demostrativo en CyberChef

Apéndice B. Repositorio del proyecto

Apéndice C. Video guía instalación de librerías

Apéndice D. Video guía del análisis con PPK2

Apéndice E. Anexos y Entregables

Los apéndices están adjuntos y puede visualizarlos en la base de datos de la biblioteca UIS

Resumen

Título: Análisis del costo computacional de la criptografía en microcontroladores [1*]

Autores: Cristian Manuel Sierra Jerez y Abad Plata Vera[2*]

Palabras clave: Algoritmos criptográficos, Costo computacional, Microcontroladores.

Descripción: La criptografía se entiende como la ciencia de ocultar o transformar la información mediante el uso de claves generadas por operaciones matemáticas que requieren recursos computacionales para ejecutarse, permitiendo cifrar o descifrar la información. Los microcontroladores son dispositivos electrónicos usados para monitorear variables y procesos en soluciones del internet de las cosas (IoT) donde según su arquitectura y características su costo varía en el mercado. Esta investigación implementó una metodología de evaluación sobre la viabilidad de la criptografía aplicada en los microcontroladores ESP32 DEV Module, Raspberry Pi Pico, STM32F103CB y Arduino Uno R3. De los cuales, el ESP32 es el único que incorpora un acelerador criptográfico por hardware, lo que le otorga una ventaja significativa en la ejecución de algoritmos al reducir la carga de cómputo sobre el procesador principal. La metodología se enfocó en la evaluación del consumo de memoria, tiempo de cómputo, consumo energético y nivel de seguridad de los algoritmos criptográficos, apoyándose en equipos de alta precisión capaces de medir corrientes desde 200 nA hasta 1 A con cambios de escala muy cortos. Por ello, el objetivo principal de esta investigación es implementar algoritmos criptográficos en microcontroladores de bajo costo para el análisis de su eficiencia y costo computacional. En definitiva, la investigación demostró que la implementación de criptografía en microcontroladores es viable siempre que exista un balance adecuado entre algoritmo y plataforma según las necesidades de cada aplicación, permitiendo alcanzar eficiencia en velocidad, consumo energético y seguridad.

[1*] Trabajo de grado.

[2*] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones E3T. Director: Jaime Guillermo Barrero Pérez, M.Sc en Potencia eléctrica.

Abstract

Title: Analysis of the computational cost of cryptography in microcontrollers [3*]

Author(s): Cristian Manuel Sierra Jerez y Abad Plata Vera [4*]

Keywords: Cryptographic algorithms, Computational cost, Microcontrollers.

Description: Cryptography is understood as the science of hiding or transforming information through the use of keys generated by mathematical operations that require computational resources to execute, allowing information to be encrypted or decrypted. Microcontrollers are electronic devices used to monitor variables and processes in Internet of Things (IoT) solutions, where their cost varies in the market depending on their architecture and characteristics. This research implemented an evaluation methodology on the feasibility of cryptography applied to the ESP32 DEV Module, Raspberry Pi Pico, STM32F103CB, and Arduino Uno R3 microcontrollers. Of these, the ESP32 is the only one that incorporates a hardware cryptographic accelerator, which gives it a significant advantage in the execution of algorithms by reducing the computational load on the main processor. The methodology focused on evaluating the memory consumption, computation time, energy consumption, and security level of cryptographic algorithms, relying on high-precision equipment capable of measuring currents from 200 nA to 1 A with very short scale changes. Therefore, the main objective of this research is to implement cryptographic algorithms in low-cost microcontrollers to analyze their efficiency and computational cost. In short, the research showed that implementing cryptography in microcontrollers is feasible as long as there is an appropriate balance between algorithm and platform according to the needs of each application, allowing for efficiency in speed, energy consumption, and security.

[3*] Degree Work

[4] School of Physical-Mechanical Engineering. Department of Electrical, Electronic and Telecommunications Engineering. Advisor: Jaime Guillermo Barrero Pérez, M.Sc in electrical power

Introducción

El mundo actual ha experimentado una transformación radical en la forma en que aprendemos, trabajamos y vivimos, convirtiendo a la tecnología en una herramienta indispensable. Los avances tecnológicos en el ámbito del Internet de las Cosas (IoT) han abierto un nuevo horizonte en una era donde la tecnología se integra cada vez más en el día a día de las personas.

Sin embargo, el desarrollo de aplicaciones IoT conlleva diversos desafíos tecnológicos y de seguridad, como lo demuestra un estudio reciente de Rueda (2021). Entre los más relevantes se encuentran la comunicación inalámbrica, el consumo de energía, las limitaciones en hardware, la escalabilidad, la identificación de nodos sensores, la sincronización en la transmisión y el intercambio de claves. Estos desafíos pueden afectar el rendimiento, la confiabilidad de los recursos, la seguridad de los datos y la gestión de identidad, aspectos que deben considerarse cuidadosamente en el proceso de diseño e implementación.

Estas necesidades de seguridad se intensifican en áreas específicas como el IIoT (Internet Industrial de las Cosas), IoMT (Internet de las Cosas Médicas) y el IoT aplicado a servicios para el hogar. En estas áreas, las tecnologías IoT están directamente relacionadas con el bienestar de las personas, pues manejan datos tanto de procesos industriales como de salud, los cuales forman parte de la información personal y privada de cada individuo o empresa. Un estudio presentado por Olivarez, *et al.* (2023) revela un aumento del 400% en ataques dirigidos a dispositivos IoT entre 2018 y 2020, con una tendencia al alza en los años siguientes. Esta creciente amenaza exige la implementación de medidas de seguridad robustas y la adopción de protocolos estandarizados para proteger la integridad y confidencialidad de los datos en estas áreas críticas.

Diversos estudios se han enfocado en crear algoritmos de criptografía ligera para proteger la información en sistemas IoT limitados. Se han propuesto alternativas a la gestión o distribución de claves, como las presentadas por Gouvêa, *et al.* (2012) y Wang, *et al.* (2007), donde se implementa criptografía de clave pública en microcontroladores con multiplicadores en hardware. También se ha analizado los requerimientos de diferentes algoritmos criptográficos ultraligeros en aplicaciones específicas, como en la agricultura (Chaturvedi, *et al.*, 2022). Sin embargo, investigaciones como la de Rana, *et al.* (2022) revelan vulnerabilidades en estos algoritmos ultraligeros ante ataques de fuerza bruta, lo que hace necesario seguir indagando sobre la viabilidad de algoritmos criptográficos ya estandarizados.

Por lo tanto, la viabilidad de los algoritmos criptográficos estandarizados en microcontroladores de bajo costo varía según sus características en hardware, la complejidad propia de cada algoritmo, los requisitos de procesamiento de los algoritmos, la velocidad de procesamiento propia de cada microcontrolador y el nivel de seguridad que pueden proporcionar. La contribución de este trabajo se basa en la implementación de una metodología de evaluación mediante la cual se analiza el rendimiento de los algoritmos criptográficos estandarizados basados en el consumo de memoria, consumo energético, tiempo de cómputo y nivel de seguridad que pueden ofrecer. En consecuencia, el presente trabajo se organiza de manera que, la sección 3 desarrolla el marco teórico, abordando los principales algoritmos criptográficos y los microcontroladores seleccionados. La sección 4 expone la métrica propuesta para evaluar los algoritmos en sistemas de recursos limitados, mientras que la sección 5 detalla la configuración experimental empleada para ejecutar los algoritmos y realizar las mediciones correspondientes. En la sección 6 se presentan y analizan los resultados obtenidos. Finalmente, la sección 7 ofrece las conclusiones y plantea futuros trabajos.

1. Objetivos

1.1 Objetivo General

Implementar algoritmos criptográficos en microcontroladores de bajo costo para el análisis de su eficiencia y costo computacional.

1.2 Objetivos Específicos

- Identificar cuatro algoritmos criptográficos estandarizados que se consideren los más seguros y cuatro microcontroladores de bajo costo con sus respectivas características en hardware.
- Ejecutar los algoritmos criptográficos sobre los microcontroladores con su lenguaje y entorno de programación apropiado haciendo uso de librerías de *open source* ya desarrolladas.
- Diseñar una métrica que permita evaluar el rendimiento y eficiencia de los algoritmos criptográficos basados en el consumo de energía, consumo de memoria, tiempo de cómputo y nivel de seguridad que ofrecen sobre los microcontroladores.

2. Marco Teórico

2.1 Clasificación De La Criptografía: Un Enfoque Desde Los Algoritmos Criptográficos Estandarizados

La criptografía se ha consolidado como una disciplina científica esencial para la protección de la información y las comunicaciones frente a vulnerabilidades o manipulaciones externas. Si bien sus orígenes se remontan a métodos rudimentarios empleados en la antigüedad para cifrar mensajes, en la actualidad constituye un campo multidisciplinario que integra matemáticas avanzadas, teoría computacional, estadística y probabilidad.

En una era marcada por la digitalización, la criptografía se establece como pilar fundamental de la ciberseguridad, al garantizar la confidencialidad, integridad y autenticidad de los datos en entornos interconectados. Su evolución ha impulsado una creciente inversión en investigación y desarrollo, dando lugar a diversas ramas especializadas como la autenticación, las firmas digitales, la esteganografía y la criptografía cuántica (NIST, 2024). Dentro de estas ramas, las técnicas de cifrado ocupan un lugar central, siendo parte del enfoque principal de esta investigación. Estas técnicas se dividen en dos grandes categorías: algoritmos de clave privada y algoritmos de clave pública.

2.1.1 Algoritmos de clave privada

Los algoritmos de clave privada, también conocidos como algoritmos simétricos, utilizan una única clave para el cifrado y descifrado de la información. Esta clave debe mantenerse en estricta confidencialidad y ser conocida únicamente por las partes autorizadas en la comunicación. En este contexto, una de sus principales ventajas es la alta eficiencia

computacional en comparación con los algoritmos de clave pública o asimétricos (Marqas, *et al.,* 2020), lo que los hace especialmente adecuados para entornos donde se requiere velocidad y manejo de grandes volúmenes de datos. AES (*Advanced Encryption Standard*) y ChaCha20 son ejemplos ampliamente utilizados de esta categoría, reconocidos por su velocidad, seguridad y eficacia en aplicaciones modernas.

2.1.1.1 Cifrado por bloques (*Block Cipher*) – *Advanced Encryption Standard* (AES).

El AES es un algoritmo de cifrado simétrico por bloques ampliamente adoptado en estándares internacionales y aplicaciones industriales debido a su solidez criptográfica, eficiencia computacional y resistencia comprobada frente a múltiples vectores de ataque. AES opera sobre bloques de 128 bits (16 bytes), y su fortaleza radica tanto en su estructura interna como en la complejidad de sus operaciones matemáticas.

El proceso de cifrado se estructura en tres etapas principales: expansión de clave, cifrado y descifrado. En la expansión de clave, la clave original se somete a una serie de transformaciones que generan subconjuntos llamados claves de ronda. La cantidad de estas claves depende del tamaño de la clave de entrada. Por ejemplo:

- AES-128 (clave de 16 bytes) genera 11 claves de 16 bytes.
- AES-192 (24 bytes) genera 13 claves de 16 bytes.
- AES-256 (32 bytes) produce 15 claves de 16 bytes.

El texto plano a cifrar es dividido en bloques; cada bloque de texto plano es de 128 bits y se procesa a través de múltiples rondas iterativas. Inicialmente, se aplica una operación XOR entre el bloque de entrada y la primera clave de ronda. Luego, en cada ronda, el bloque es transformado mediante operaciones bien definidas: *SubBytes* (sustitución no lineal), *ShiftRows*

(desplazamiento de filas), *MixColumns* (mezcla de columnas) y *AddRoundKey* (combinación con la clave de ronda actual). Estas transformaciones garantizan difusión y confusión[5], principios fundamentales de la criptografía moderna. El resultado tras la última ronda es el bloque de texto cifrado.

El proceso de descifrado aplica transformaciones inversas con el mismo conjunto de claves, pero en orden descendente. Aunque las operaciones utilizadas, como *InvSubBytes*, *InvShiftRows* e *InvMixColumns*, son diferentes, se diseñan para revertir exactamente el proceso de cifrado, asegurando que el texto plano original pueda recuperarse sin pérdida ni alteración.

AES puede funcionar bajo diversos modos de operación, de los cuales uno de los más utilizados es CBC (*Cipher Block Chaining*). En este modo, cada bloque de texto plano se somete a una operación XOR con el bloque cifrado anterior antes de ser cifrado. Para el primer bloque, se utiliza un vector de inicialización (IV) único y aleatorio[6]. Esto introduce aleatoriedad adicional al proceso, de manera que, incluso si el texto plano se repite, los resultados cifrados serán distintos siempre que el IV cambie. Sin embargo, el manejo incorrecto del IV o su reutilización puede comprometer seriamente la seguridad del algoritmo.

[5] En criptografía, “difusión” se refiere a cómo un pequeño cambio en el texto plano afecta muchas partes del texto cifrado, mientras que “confusión” busca ocultar la relación entre la clave y el texto cifrado.

[6] El vector de inicialización (IV) es un valor aleatorio utilizado para cifrar el primer bloque de datos al no tener un bloque cifrado previo para dar inicio, asegurando que mensajes idénticos no generen salidas cifradas iguales. Debe ser único y, preferentemente, aleatorio para cada sesión de cifrado.

2.1.1.2 Cifrado por flujo (*Stream Cipher*) - ChaCha20. ChaCha20 es un algoritmo de cifrado simétrico por flujo, ampliamente valorado por su rendimiento, seguridad y resistencia frente a ataques criptoanalíticos modernos. Su diseño se basa en la generación de un flujo de claves pseudoaleatorias condensadas en un *keystream*[7], que se combina con el mensaje mediante una operación XOR, tanto para cifrar como para descifrar. En consecuencia, este cifrado requiere tres parámetros fundamentales los cuales son una clave secreta de 256 bits (32 bytes), un *nonce* de 96 bits (12 bytes), y un contador de bloques de 32 bits.

Dicho esto, el funcionamiento de ChaCha20 inicia con la construcción de una matriz de estado de 4x4 palabras de 32 bits (16 posiciones). La primera fila contiene una constante fija de 16 bytes: "*expand 32-byte k*"[8], codificada en ASCII. Las filas dos y tres almacenan los 32 bytes de la clave, distribuidos uniformemente en ocho palabras. Finalmente, la cuarta fila incorpora el contador en su primera posición y el *nonce* en las tres restantes, completando así los 64 bytes requeridos para iniciar el algoritmo donde esta matriz constituye el punto de partida para la transformación criptográfica.

El cifrado se realiza aplicando 20 rondas de transformación, divididas en 10 rondas dobles, mediante una serie de operaciones conocidas como *quarter rounds*[9]. Las rondas dobles alternan entre mezclas en columnas y en diagonales de la matriz, asegurando una dispersión completa de la información y una resistencia elevada contra ataques por análisis diferencial y lineal. Una vez concluidas las 20 rondas, se realiza una suma modular palabra por palabra entre la matriz transformada y la matriz original, produciendo así el *keystream*. Por lo cual, ChaCha20

[7] *keystream* es una secuencia pseudoaleatoria de bits generada a partir de una clave y un *nonce* (valor único por mensaje); se combina con el texto plano mediante una operación XOR para producir el texto cifrado. Su seguridad depende de la imprevisibilidad del *keystream* y del uso único del *nonce* por clave.

[8] El valor "*expand 32-byte k*" representa una constante fija representada por bytes (4 bytes por columna de la matriz en su primera fila) donde cada carácter del valor "*expand 32-byte k*" representa un byte.

[9] Los *quarter rounds* son operaciones básicas que mezclan cuatro palabras de 32 bits usando sumas modulares, rotaciones y XOR. Estas rondas introducen difusión y confusión en los bloques internos del algoritmo, y son clave para su seguridad y eficiencia.

mantiene un enfoque simétrico donde el mismo *keystream* generado a partir de una combinación única de clave, *nonce* y contador se aplica mediante una operación XOR al texto plano para producir el texto cifrado, y de igual forma, al texto cifrado para recuperar el texto original. En consecuencia, este enfoque, además de ser altamente eficiente, facilita su implementación segura en múltiples plataformas y contextos operacionales.

2.1.2 Algoritmos de clave pública

Los algoritmos de clave pública, también llamados asimétricos, emplean un par de claves diferenciadas: una pública, que puede compartirse, y una privada, que debe mantenerse en confidencialidad. El principio básico es que lo que cifra una clave sólo puede descifrarlo la otra. En este contexto, los algoritmos de clave publica a diferencia de los algoritmos de clave simétrica, no están optimizados para grandes volúmenes de datos, debido a su mayor complejidad computacional. Sin embargo, su elevado nivel de seguridad los convierte en piezas clave para establecer canales seguros y verificar identidades en entornos digitales (Rao, UH, Nayak, U. (2014)).

Entre los algoritmos de clave pública más reconocidos y empleados en la actualidad se encuentran RSA (Rivest - Shamir – Adleman), basado en la dificultad de factorizar números primos grandes y ECC (*Elliptic Curve Cryptography*), que ofrece una seguridad equivalente con claves de menor tamaño. Ambos esquemas son fundamentales en infraestructuras modernas de seguridad, tales como protocolos TLS/SSL, firmas digitales y sistemas de autenticación robusta.

2.1.2.1 Rivest - Shamir - Adleman (RSA). RSA es un algoritmo de cifrado asimétrico ampliamente reconocido en el ámbito de la seguridad informática por su robustez y confiabilidad. Su funcionamiento se basa en principios de la teoría de números, particularmente en la dificultad computacional de factorizar enteros grandes donde en primer lugar, el proceso comienza con la generación de claves donde se seleccionan dos números primos grandes, p y q, a partir de los cuales se calcula $n = p * q$. Este valor n forma parte de las claves pública/privada y determina el tamaño del módulo sobre el cual se realizan las operaciones. A continuación, se calcula la función $\lambda(n)$, obtenida como el mínimo común múltiplo de $(p - 1)$ y $(q - 1)$, es decir $\lambda(n) = mcm(p - 1, q - 1)$. Posteriormente, se selecciona un exponente público e, tal que $1 < e < \lambda(n)$ y que sea coprimo con $\lambda(n)$. Este valor constituye, junto con n, la clave pública (n, e). En segundo lugar, para generar la clave privada, se calcula el valor d, correspondiente al inverso multiplicativo de e módulo $\lambda(n)$, cumpliendo con la congruencia $e * d = 1 \, mod \, \lambda(n)$ donde la clave privada queda definida por el par (n, d).

Finalmente, el proceso de cifrado consiste en tomar un mensaje m, previamente convertido a su representación numérica (por ejemplo, a través de codificación ASCII), y aplicar la fórmula: $c = m^e \, mod \, n$ donde c representa el mensaje cifrado donde, para recuperar el mensaje original, el receptor utiliza su clave privada y aplica el proceso inverso. De esta forma, este esquema asegura que solo el titular de la clave privada puede descifrar el mensaje, garantizando así la confidencialidad y la seguridad de la comunicación.

2.1.2.2 Elliptic curve cryptography (ECC). El ECC se fundamenta en las propiedades algebraicas de las curvas elípticas sobre cuerpos finitos. Su principal ventaja es ofrecer un nivel de seguridad equivalente al de RSA, pero con claves de menor tamaño, lo que implica menor consumo de cómputo, memoria y ancho de banda (Guajardo et al., 2001). Donde, una curva elíptica sobre un campo finito F_p (con p número primo) se define mediante la ecuación general: $y^2 = x^3 + ax + b \mod p$ donde a y b son constantes que deben satisfacer la condición de no singularidad definida como $4a^3 + 27b^2 \neq 0 \mod p$. Esta condición garantiza que la curva no tenga puntos singulares, es decir, no presenta cúspides ni autointersecciones[10].

Por ejemplo, en una implementación típica, el emisor y el receptor acuerdan una curva elíptica y un punto generador G de orden n. Luego, cada parte genera una clave privada d que es un entero aleatorio tal que $1 < d < n$ y calcula su clave pública como $Q = d * G$. Así, gracias a la eficiencia matemática de su estructura y al tamaño reducido de sus claves donde por ejemplo, una clave de 256 bits en ECC ofrece un nivel de seguridad comparable a una clave de 3072 bits en RSA, este algoritmo se ha convertido en una de las opciones preferidas para entornos modernos que requieren alta seguridad y bajo consumo de recursos.

2.1.3 Funciones Hash

Las funciones hash criptográficas constituyen un pilar fundamental en la seguridad informática moderna. Su principal propósito es transformar una entrada de longitud arbitraria en una salida de longitud fija, denominada "resumen" o *digest*, que representa de manera única al mensaje original. A diferencia de los algoritmos de cifrado, estas funciones no están diseñadas

[10] La ausencia de cúspides, autointersecciones y puntos singulares garantiza que en cada punto de la curva exista una única tangente bien definida. Esto es esencial para que las operaciones algebraicas como la suma de puntos, sean válidas, consistentes y seguras en el contexto criptográfico.

para ser reversibles, es decir, a partir del valor hash no es posible reconstruir el mensaje de entrada.

2.1.3.1 ***Secure Hashing Algorithm*** **(SHA-256).** SHA-256 es una función hash criptográfica que transforma una entrada de datos de longitud variable en una salida de longitud fija de 256 bits (32 bytes) donde a diferencia de los algoritmos de cifrado, SHA-256 no está diseñado para ser reversible, sino para producir una representación única del contenido original, garantizando propiedades como la integridad, la unidireccionalidad y la resistencia a colisiones.

El funcionamiento de SHA-256 se organiza en varias fases claramente definidas en los siguientes procesos: preprocesamiento (*padding* y partición), expansión de palabras, compresión iterativa y generación del hash final. En la fase de *padding*, el mensaje original m se rellena añadiendo un bit '1' al final del mensaje seguido de una secuencia de ceros de modo que ($m_l + 1 + z) = 448 \, mod \, 512$ donde l representa la longitud del mensaje original y z la cantidad de ceros añadidos. Esto significa que faltan 64 bits para completar un bloque de 512 bits por lo que finalmente se añaden 64 bits que representan la longitud original del mensaje antes del *padding* en notación binaria de 64 bits. Seguidamente, el mensaje se divide en bloques de 512 bits donde cada bloque se subdivide en 16 palabras de 32 bits, las cuales se expanden a 64 palabras mediante funciones de rotación, desplazamiento y operaciones XOR. Esta expansión asegura una fuerte dependencia entre los datos iniciales y cada paso del proceso, aumentando la complejidad computacional frente a ataques.

El núcleo del algoritmo es el proceso de compresión, que utiliza ocho registros de trabajo a, b, c, d, e, f, g, h, inicializados con constantes predefinidas denominadas valores hash iniciales H_0, H_1, H_2, H_3... H_7. Las cuales, durante 64 rondas, las palabras expandidas y constantes

específicas interactúan con estos registros mediante operaciones lógicas (AND, OR, XOR), rotaciones y sumas módulo 2^{32}. Así, estas operaciones generan un entrelazado no lineal de los datos, asegurando un alto grado de difusión que al finalizar las 64 rondas, hace que los valores intermedios se sumen a los valores hash iniciales y se actualicen. La concatenación final de estos ocho valores de 32 bits produce el hash de salida de 256 bits, este resultado actúa como una huella digital criptográfica única del mensaje, siendo computacionalmente inviable encontrar dos entradas distintas que generen el mismo hash, lo cual garantiza la seguridad e integridad de los datos procesados.

2.2 Microcontroladores De Bajo Costo En Sistemas Del Internet De Las Cosas Iot

Los microcontroladores de bajo costo juegan un papel crucial en los sistemas IoT, ya que satisfacen las crecientes demandas de conectividad y escalabilidad que requieren estas tecnologías. Su accesibilidad económica, eficiencia energética y el soporte de una amplia comunidad de desarrolladores permiten la implementación de soluciones IoT en diversos campos como la domótica, la salud digital, la agricultura inteligente, el transporte, entre otros.

Esta evolución acelerada de tecnologías IoT ha impulsado el mercado hacia el desarrollo de microcontroladores cada vez más potentes. Sin embargo, no todos mantienen un precio accesible, lo que resalta la necesidad de establecer métricas claras para seleccionar dispositivos ideales en proyectos de bajo costo.

2.2.1 ESP32 DEV Module

El ESP32 se ha consolidado como una de las opciones más utilizadas en IoT debido a su relación costo/beneficio. Este módulo integra un procesador Xtensa LX6 de doble núcleo con

una frecuencia de operación que alcanza los 240 MHz, lo que le otorga una capacidad de procesamiento muy superior frente a microcontroladores clásicos. Dispone de 520 KB de SRAM interna, suficiente para el manejo de buffers de comunicación y procesos criptográficos, además de soporte para memoria Flash externa de hasta 16 MB, que permite almacenar firmware y librerías complejas. Su punto más fuerte radica en la conectividad: incorpora Wi-Fi 802.11 b/g/n y Bluetooth 4.2 (clásico y BLE) en un mismo chip, lo que lo convierte en un dispositivo altamente versátil. Asimismo, ofrece modos de bajo consumo, entre ellos el deep sleep, que permiten mantener tareas mínimas con un gasto energético muy reducido, característica indispensable en aplicaciones IoT alimentadas por baterías (Espressif, 2020).

2.2.2 Raspberry Pi Pico (RP2040)

Basada en el microcontrolador RP2040, representa la apuesta de la Fundación Raspberry Pi por un microcontrolador accesible con módulos de conectividad inalámbrica disponibles. Este chip cuenta por una parte, con un procesador ARM Cortex-M0+ de doble núcleo que funciona hasta 133 MHz, acompañado de 264 KB de SRAM y 2 MB de memoria Flash QSPI además que permiten integrar módulos de comunicación externos. Por otra parte, en cuanto a eficiencia energética, ofrece modos de suspensión como *sleep* y *dormant*, que permiten pausar los núcleos o desactivar periféricos para prolongar la vida útil de dispositivos portátiles (Raspberry Pi Ltd, 2022).

2.2.3 STM32F103CB (STM32)

Conocido popularmente como Blue Pill, se caracteriza por un balance entre costo reducido y capacidad de cómputo. Basado en un ARM Cortex-M3 de 32 bits a 72 MHz, cuenta

con 131 KB de memoria Flash y 20 KB de SRAM, lo que lo convierte en una alternativa potente dentro de la gama media de microcontroladores. Su conectividad no es inalámbrica de forma nativa, pero dispone de una amplia variedad de interfaces como UART, SPI, I2C, CAN y USB, que permiten integrar módulos de comunicación externos. Además, incluye un ADC de 12 bits con varios canales, así como temporizadores avanzados para aplicaciones de control. Como otros miembros de la familia STM32, incorpora modos de bajo consumo como *sleep* y *stop*, lo que lo hace útil en sistemas IoT donde la optimización energética es un requisito clave (STMicroelectronics, 2019).

2.2.4 Arduino Uno R3 (ATmega328p)

Basado en el microcontrolador ATmega328P, sigue siendo una de las plataformas más populares a nivel educativo y en prototipado rápido. Su arquitectura de 8 bits y frecuencia de 16 MHz son limitadas frente a las opciones anteriores, pero su sencillez y estabilidad lo mantienen vigente. Dispone de 32 KB de memoria Flash, 2 KB de SRAM y 1 KB de EEPROM, lo que restringe el tamaño de los programas y datos que puede manejar, pero resulta suficiente para aplicaciones básicas de censado y control. Por su parte, en cuanto a conectividad, solo ofrece interfaces cableadas como UART, I2C y SPI, siendo necesario recurrir a módulos externos para añadir Wi-Fi o Bluetooth. Finalmente, su consumo energético es bajo en comparación con microcontroladores más potentes, aunque sus modos de suspensión son menos sofisticados (Microchip, 2024).

3. Métrica Propuesta: Evaluación De Los Algoritmos En Sistemas De Recursos Limitados

La métrica propuesta en este trabajo se basa en la selección cuidadosa de microcontroladores y algoritmos criptográficos, la configuración del entorno de desarrollo apropiado, la implementación de los algoritmos criptográficos y las pruebas de desempeño. Por ello, esta métrica sigue los lineamientos necesarios para dar cumplimiento al objetivo general de la investigación, desarrollándose de la siguiente manera:

- **Selección de algoritmos criptográficos y microcontroladores de bajo costo:**

Como punto de partida, se identificaron cuatro microcontroladores de bajo costo con diferentes arquitecturas de hardware y cinco algoritmos criptográficos estandarizados. La selección de algoritmos se basó en una revisión bibliográfica detallada y en la relevancia de los algoritmos según su aplicación en entornos tecnológicos reales siendo seleccionados los algoritmos AES, ChaCha20, SHA2-256, RSA y ECC, los cuales son implementados en diferentes ámbitos tecnológicos estandarizados mundialmente para el manejo de información. Mientras que la selección de microcontroladores se basó en la revisión de las hojas de datos de cada microcontrolador donde se consideraron los siguientes criterios:

- Costo comercial no mayor a $50.000 pesos colombianos
- Conectividad Wi-Fi o módulos compatibles para transmisión inalámbrica
- Relevancia en entornos de desarrollo IoT

Los microcontroladores que cumplieron con estos requisitos fueron ESP32, Raspberry Pi Pico, Arduino Uno R3 y STM32F103CB. Sus características técnicas detalladas están descritas en el marco teórico.

- **Elección del lenguaje de programación y el entorno de desarrollo:**

Se eligió el lenguaje de programación C pues, según Ionescu, et al. (2020), permite una mejor asignación de memoria y un aprovechamiento eficiente de los recursos en los microcontroladores a nivel de hardware, en comparación con desarrollos en Python. El entorno de desarrollo seleccionado fue la IDE de Arduino, además de su lenguaje de programación natural en C, cuenta con soporte para una gran variedad de placas de desarrollo donde se realizará el análisis de los algoritmos seleccionados. Finalmente, se desarrollaron librerías propias para los algoritmos AES, ChaCha20, SHA-256 y RSA, siguiendo rigurosamente las especificaciones teóricas expuestas en el marco teórico. No obstante, en el caso de ECC, se recurrió a librerías *Open source* ya creadas como la tinyECC (Annigeri, 2021) que, al tener la optimización apropiada para su ejecución en microcontroladores de 8 bits, permite una implementación más estable en dispositivos de recursos limitados.

- **Implementación de los algoritmos criptográficos y criterios de evaluación:**

La implementación de los algoritmos se realizó a partir de códigos que representan el procesamiento para la transmisión de una cadena de 16 caracteres sin formato (texto plano), la cual se propuso ser manejada en formato hexadecimal para manejos eficientes durante cada proceso de cifrado. Finalmente, se evalúa el consumo de memoria, tiempo de cómputo y nivel de seguridad aportado a cada sistema embebido. Los resultados se compararon con el consumo básico de cada microcontrolador con sus periféricos según las hojas de datos.

3.1 Tiempo de cómputo y consumo de memoria

El tiempo de cómputo se midió mediante la función integrada “micros()” de Arduino IDE, la cual devuelve el tiempo en microsegundos desde el inicio del programa. Para cada proceso de cifrado y descifrado, se almacenó el tiempo inicial antes de ejecutar el algoritmo y se calculó el tiempo transcurrido al finalizar, obteniendo así la duración exclusiva de la operación. Cada prueba se repitió 100 veces en cada microcontrolador, de modo que el promedio resultante refleja una medición confiable y representativa del rendimiento.

Luego, para la medición del consumo de memoria, se utilizó una funcionalidad integrada en el entorno de desarrollo Arduino IDE. Esta característica permite, al momento de compilar cada algoritmo, obtener información detallada sobre el uso de memoria en el microcontrolador programado donde la IDE identifica automáticamente la capacidad total de memoria disponible según el modelo específico del microcontrolador y calcula la carga que representa el código tanto en la memoria RAM como en la ROM. Gracias a estas métricas proporcionadas por el entorno, se realizaron las mediciones necesarias para determinar la cantidad de memoria utilizada durante los procesos de cifrado y descifrado de los diferentes algoritmos implementados.

3.2 Consumo Energético

El análisis asociado con el consumo de energía se realizó empleando la placa *Power Profiler Kit* II la cual tiene su propio software que se ejecuta en el nRF Connect para Escritorio. El *Power Profiler Kit II* (PPK2) es una herramienta esencial para la medición precisa del consumo energético en microcontroladores cuyo costo comercial ronda los 141 dólares. Este dispositivo, diseñado por *Nordic Semiconductor*, permite medir corrientes de niveles extremadamente bajos, en el rango de nano amperios, hasta valores de 1 amperio, lo que lo hace

ideal para analizar tanto los estados de bajo consumo (modos de suspensión) como los picos de corriente durante la ejecución activa de algoritmos, lo que es particularmente útil en investigaciones como la presente, donde se evalúa el impacto energético de la criptografía en microcontroladores. Además, el PPK2 es compatible con hardware externo y kits de desarrollo (DK) de Nordic, lo que amplía su versatilidad para diversas aplicaciones.

El PPK2 opera en dos modos principales, siendo el primero como medidor de amperios y el segundo como fuente de alimentación con medición integrada (SMU) (Nordic Semiconductor, 2024) donde en primer lugar, el modo medidor de amperios se requiere una fuente de alimentación externa que suministre niveles de VCC entre 0.8 y 5V al dispositivo bajo prueba (DUT) y en segundo lugar, en el modo de fuente, el PPK2 proporciona directamente niveles de VCC dentro del mismo rango y puede suministrar hasta 1A de corriente gracias a su regulador integrado. Así, este diseño permite medir tanto corrientes promedio como eventos de alta resolución, lo que resulta fundamental para caracterizar el comportamiento energético de los microcontroladores durante operaciones criptográficas.

3.3 Nivel de seguridad

La evaluación del nivel de seguridad ofrecido por los algoritmos criptográficos implementados en los microcontroladores se fundamentó en la correcta ejecución de cada algoritmo y la verificación del resultado obtenido. Dado que los algoritmos seleccionados (AES, ChaCha20, RSA, ECC y SHA-256) son reconocidos por su solidez criptográfica a nivel internacional, el criterio de validación se centró en comparar los resultados generados por los microcontroladores con aquellos esperados según estándares establecidos, utilizando para ello

herramientas reconocidas como CyberChef[11]. Esta plataforma permite validar y verificar operaciones criptográficas, facilitando la comprobación de que el resultado obtenido en entornos de recursos limitados coincide con el esperado en plataformas de mayor capacidad computacional. Determinando que el nivel de seguridad es adecuado cuando el resultado del cifrado realizado en el microcontrolador coincide fielmente con el resultado generado por la herramienta de referencia, lo que indica una implementación correcta del algoritmo al margen de las limitaciones propias del hardware, la lógica criptográfica se mantiene intacta y confiable (Video demostrativo del proceso de ejecutado en CyberChef disponible en el Apéndice A).

Cabe destacar que la seguridad criptográfica no reside únicamente en la robustez del algoritmo, sino en la correcta gestión y protección de las claves empleadas donde tal como señala la Universidad de Valencia (2024), la confidencialidad de las claves es el factor determinante en la seguridad de un sistema criptográfico. Por tanto, si bien esta investigación valida la correcta ejecución de los algoritmos en los microcontroladores, se reconoce que una solución integral de seguridad en entornos como el Internet de las Cosas requiere, además, mecanismos seguros de distribución de claves, autenticación y transmisión de datos.

4. Configuración Experimental

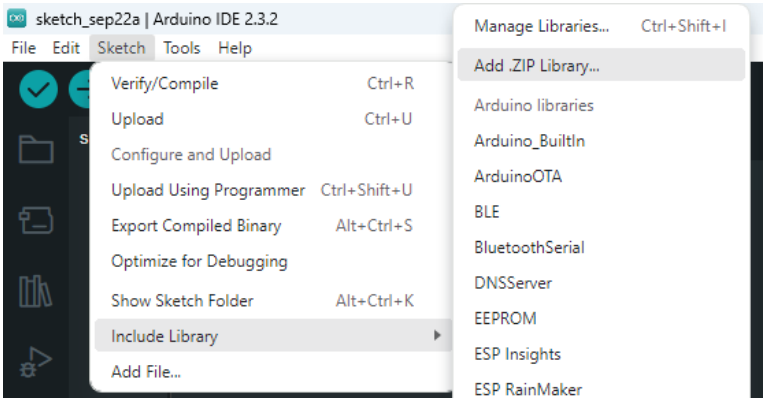
4.1 Ejecución de los algoritmos sobre un entorno IoT

En la etapa de ejecución fue necesario disponer de cada una de las librerías diseñadas (Librerías disponibles en el Apéndice B) donde el entorno de Arduino IDE permite incorporarlas

[11] CyberChef es una herramienta web desarrollada por GCHQ para analizar, transformar y visualizar datos mediante operaciones como codificación, cifrado, hashing, entre otras. Es especialmente útil en tareas de seguridad informática y análisis criptográfico. Disponible en: https://gchq.github.io/CyberChef/

en formato comprimido (.zip) mediante la ruta: Sketch > Include Library > Add .ZIP Library (Video guía de la inclusión de librerías disponible en el Apéndice C).

Figura 1.
Ruta de inclusión librerías Arduino.



Cada librería desarrollada incluye diferentes sketches de prueba, diseñados para facilitar la evaluación de los algoritmos y cubrir de manera específica cada métrica de análisis:

- **BasicUsage**: implementa el proceso completo de cifrado y descifrado.
- **TimePerformance**: permite medir los tiempos de cómputo asociados a las operaciones de cifrado y descifrado, de acuerdo con en el apartado 4.1.
- **Encryption**: ejecuta exclusivamente el proceso de cifrado, sin impresiones por puerto serial, con el fin de obtener mediciones más precisas sobre el uso de memoria en esta fase.
- **Decryption**: ejecuta exclusivamente el proceso de descifrado, también sin impresiones seriales, para asegurar una evaluación exacta del consumo de memoria correspondiente.
- **PowerConsumptionEncrypting**: realiza el cifrado de manera repetitiva dentro del bucle loop(), donde la operación se ejecuta n veces antes de ingresar en el estado de espera; este valor n es configurable por el usuario. Él loop() es

necesario porque, debido a la alta velocidad de los microcontroladores, una única operación de cifrado produciría un intervalo demasiado breve para ser identificado claramente en la interfaz gráfica del PPK2. Al finalizar cada conjunto de operaciones, se incorpora una función de retardo (delay) bloqueante, lo que permite distinguir con claridad los períodos correspondientes al procesamiento de cifrado y al estado de reposo del microcontrolador.

- **PowerConsumptionDecrypting**: de forma análoga, ejecuta el descifrado repetidamente dentro del bucle loop(), aplicando la operación n veces antes del estado de espera, donde *n* es igualmente configurable. Al igual que en el cifrado, esta repetición garantiza que los intervalos de actividad puedan visualizarse adecuadamente en el PPK2. Posteriormente, se incluye un retardo bloqueante que facilita a la herramienta de medición reflejar con precisión los periodos de procesamiento y de inactividad del microcontrolador.

En conjunto, cada sketch cumple una función específica dentro de los parámetros a evaluar, garantizando así una metodología estructurada y reproducible.

4.2 Validación experimental

La validación de los algoritmos de cifrado se llevó a cabo utilizando el sketch BasicUsage.ino desarrollado en la librería propia de cada algoritmo. En este procedimiento se definieron los parámetros necesarios para el cifrado, tales como la clave, el vector de inicialización (en los casos necesarios) y el texto plano de prueba.

Figura 2.

Parámetros de ejemplo cifrado AES 128.

```
uint8_t exampleKey[16] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10};

uint8_t iv[16] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10};

uint8_t plainText[plainTextSize] = { 0x43, 0x72, 0x69, 0x70, 0x74, 0x6F, 0x67, 0x72, 0x61, 0x66, 0x69, 0x61, 0x31, 0x32, 0x33, 0x34};
```

Con el fin de verificar la correcta implementación de los algoritmos, los parámetros utilizados en Arduino IDE fueron replicados en la herramienta de validación en línea CyberChef donde al aplicar el mismo conjunto de valores, se obtuvo un texto cifrado idéntico al generado en los microcontroladores, como se muestra en las figuras 3 y 4 para el algoritmo AES confirmando la fidelidad de la implementación en la etapa de cifrado. Del mismo modo, el proceso de descifrado fue validado ingresando los mismos parámetros en CyberChef donde la herramienta devolvió el texto plano original, coincidiendo exactamente con los resultados obtenidos en cada microcontrolador. Esta correspondencia en ambos procesos (cifrado y descifrado) demuestra que la implementación de los algoritmos en microcontroladores respeta los principios teóricos del estándar y mantiene consistencia frente a herramientas de referencia reconocidas.

Figura 3.

Resultados cifrado y descifrado Cyberchef.

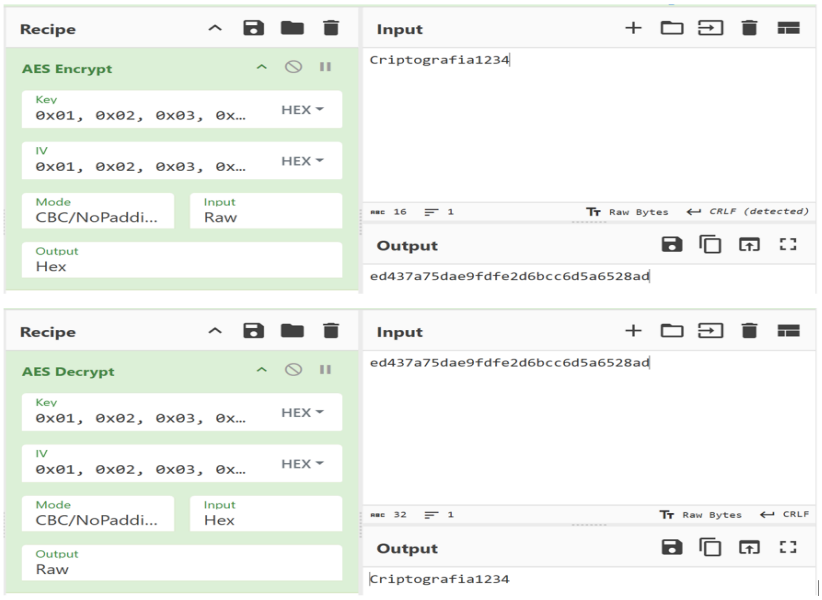
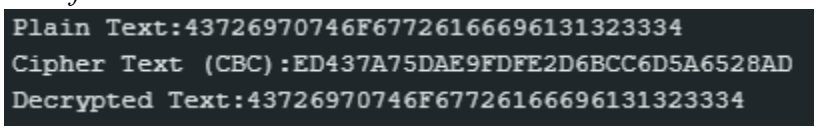


Figura 4.
Resultados cifrado y descifrado librería.



Para garantizar un manejo riguroso de los datos durante el proceso de validación, cada librería fue configurada para trabajar en formato hexadecimal tanto con el texto plano como con los parámetros asociados. Esta decisión asegura una mayor compatibilidad con las herramientas externas de validación y reduce posibles ambigüedades en la interpretación de los resultados. Cabe destacar que los valores de *Plain Text* y *Decrypted Text* presentados en la Figura 4 corresponden a la representación en hexadecimal de la cadena de caracteres “Criptografia1234” procesada en CyberChef (Figura 3), lo cual confirma la integridad entre la entrada original y la salida obtenida tras el descifrado.

Las validaciones correspondientes a los demás algoritmos implementados ChaCha20, SHA-256, RSA y ECC se llevaron a cabo siguiendo un procedimiento similar al descrito para AES. Por lo cual, en todos los casos se configuraron los parámetros de entrada en los sketches respectivos y se contrastaron los resultados obtenidos en los microcontroladores con los generados por la herramienta de validación en línea CyberChef donde la coincidencia plena entre ambas salidas confirmó la correcta implementación de cada algoritmo, garantizando la fidelidad de los procesos de cifrado, descifrado y hashing respecto a sus definiciones teóricas.

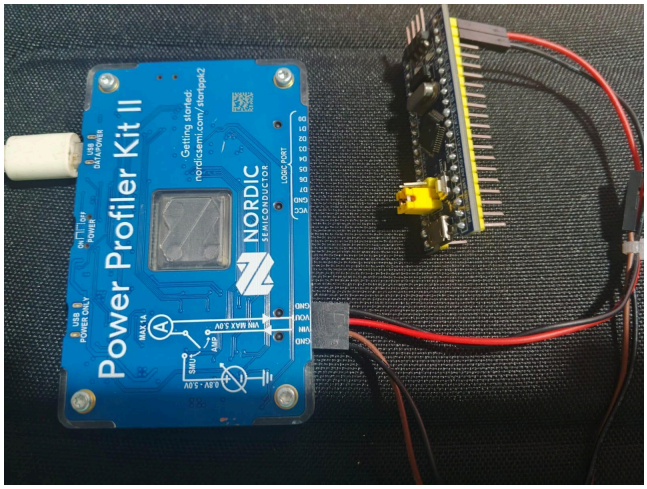
4.3 Configuración del *power profiler kit* ii (PPK2)

- **Conexiones y código**

En este proceso resulta fundamental identificar correctamente los pines Vout y GND, ya que deben conectarse a los pines de alimentación de la tarjeta en evaluación (Figura 5). Cabe

resaltar que, cuando el microcontrolador se conecta mediante el puerto USB, el pin Vout funciona como salida de tensión; en cambio, si no se utiliza dicha conexión, este mismo pin pasa a actuar como fuente de alimentación del microcontrolador. Esto se debe a que el PPK2, además de medir el consumo energético, funciona como fuente de alimentación externa para el microcontrolador durante el análisis.

Figura 5.
Conexión del power profiler kit 2 al microcontrolador STM32.



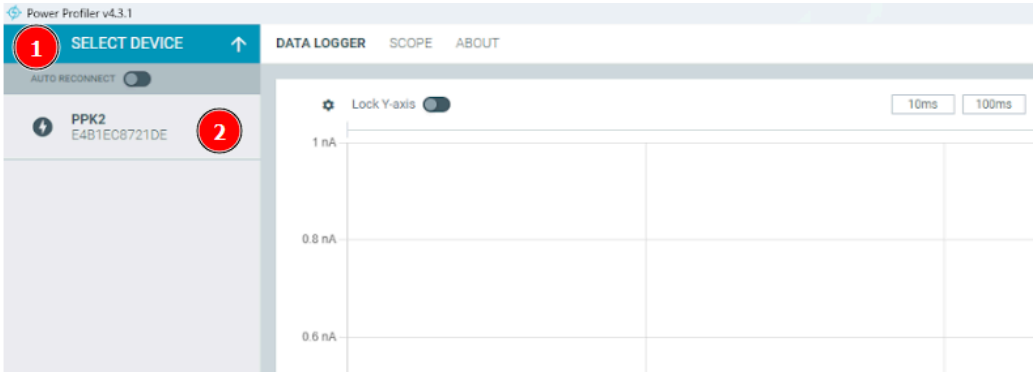
Previo al uso de la interfaz gráfica proporcionada por Nordic Semiconductor, es necesario cargar en la tarjeta el código a analizar (Video guía del proceso completo para analizar corrientes mediante el PPK2 en el Apéndice D). Para este propósito, en cada una de las librerías se dispone de los programas de prueba previamente mencionados *PowerConsumptionEncrypting* y *PowerConsumptionDecrypting*.

- **Uso de la interfaz gráfica del PPK2**

Con el código cargado en el microcontrolador y la conexión física establecida (Figura 5), el siguiente paso es acceder a la interfaz gráfica del PPK2 donde al conectar el dispositivo al ordenador, se habilita en la parte superior izquierda de la ventana principal el menú de selección.

Desde allí, al pulsar '*Select Device*', se establece la conexión con la placa reconocida por el sistema (Figura 6).

Figura 6.
Interfaz gráfica PPK2.



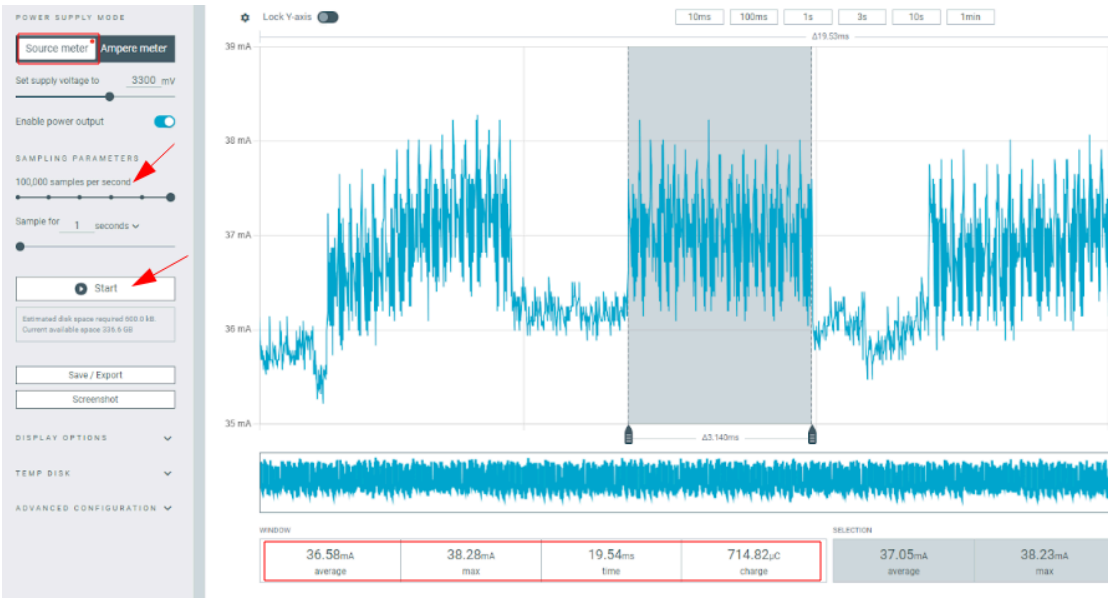
Al seleccionar la placa reconocida por el sistema, aparecerá en el panel lateral izquierdo las principales opciones de configuración, que incluyen la selección del voltaje de alimentación (3.3 V o 5 V), el control para habilitar o deshabilitar la entrega de energía hacia el microcontrolador, la definición de la frecuencia de muestreo para ajustar la resolución temporal de los datos y la configuración del periodo de análisis, que establece la ventana temporal de observación (Figura 7).

Una vez configurados estos parámetros, el análisis puede iniciarse mediante el botón Start. Al mismo tiempo, el eje X de la interfaz se ajusta automáticamente para mostrar la señal capturada. El resultado se observa en un gráfico similar al de la Figura 7, donde:

- Los valores bajos corresponden a periodos de inactividad o reposo del microcontrolador.
- Los valores altos representan la ejecución del proceso de cifrado.

La principal ventaja de esta herramienta es la posibilidad de seleccionar de forma interactiva segmentos de las mediciones, sobre los cuales el software calcula automáticamente parámetros clave como el consumo promedio, el máximo registrado, la duración de la ventana y la carga eléctrica en coulombs lo que permite un análisis detallado del consumo energético de algoritmos criptográficos en microcontroladores.

Figura 7.
Resultados de medición ejemplo.



5. Resultados Y Discusión

En esta sección se presentan los resultados y análisis en términos de tiempo de cómputo, consumo de memoria y consumo energético en los microcontroladores seleccionados donde las características a evaluar y el costo de los microcontroladores se condensan en la siguiente tabla:

Tabla 1.

Especificaciones de los microcontroladores.

Modelo (Microcontrolador)	Frecuencia operación (MHZ)	Flash (KB) disponible	SRAM(KB) disponible	Costo Comercial
ESP32 (ESP DEV Module)	240	1310	327	$5 – $8
Raspberry Pi Pico (RP2040)	133	2093	262	$4 – $6
STM32F103CB (STM32)	72	131	16	$4 – $10
Arduino Uno R3 (ATmega328p)	16	32	2	$12 – $20

- **Resultados de tiempo, memoria y corriente para la ESP32**

Tabla 2.

Resultados ESP DEV Module.

	Tiempo de cifrado [Byte/us]	Tiempo de descifrado [Byte/us]	Flash en bytes al cifrar [%]	RAM en bytes al cifrar [%]	Flash en bytes al descifrar [%]	RAM en bytes al descifrar [%]	Corriente promedio (cifrado) [mA]	Corriente promedio (descifrado) [mA]
AES128	2,88	7,00	14268 (1.089%)	256 (0.078%)	14816 (1.130%)	256 (0.078%)	73,26	71,41
AES192	3,43	8,46	14268 (1.089%)	288 (0.088%)	14816 (1.130%)	288 (0.088%)	72,21	71,66
AES256	3,98	9,92	14284 (1.090%)	336 (0.103%)	14832 (1.132%)	336 (0.103%)	72,78	71,51
Chacha20	0,47	0,47	13604 (1.038%)	80 (0.024%)	13632 (1.040%)	80 (0.024%)	71,06	70,68
SHA256	0,86	NA	13864 (1.058%)	152 (0.046%)	NA	NA	70,04	NA
RSA	2,90	19,90	13224 (1.009%)	32 (0.010%)	13616 (1.039%)	544 (0.166%)	67,32	66,33
ECC	549,89	266,66	17684 (1.349%)	276 (0.084%)	17080 (1.303%)	144 (0.044%)	67,48	68,01

En el caso del ESP32, su procesador de 32 bits a alta frecuencia, junto con el soporte nativo para operaciones criptográficas, explica los tiempos reducidos de cifrado y descifrado en algoritmos como AES y ChaCha20, manteniendo un consumo energético estable en torno a los

70 mA. A ello se suma un bajo uso de memoria Flash y RAM el cual no supera el 2% de la disponibilidad expuesta en la Tabla 1, lo que lo convierte en una plataforma eficiente para aplicaciones en tiempo real donde el balance entre velocidad y consumo es crítico.

- **Resultados de tiempo, memoria y corriente para la STM32F103CB**

Tabla 3.

Resultados STM32F103CB (STM32).

	Tiempo de cifrado [Byte/us]	Tiempo de descifrado [Byte/us]	Flash en bytes al cifrar [%]	RAM en bytes al cifrar [%]	Flash en bytes al descifrar [%]	RAM en bytes al descifrar [%]	Corriente promedio (cifrado) [mA]	Corriente promedio (descifrado) [mA]
AES128	12,30	27,44	1320 (1.007%)	232 (1.439%)	1876 (1.431%)	232 (0.001%)	36,55	35,73
AES192	15,30	33,22	1328 (1.013%)	272 (1.687%)	1884 (1.437%)	272 (1.687%)	36,65	35,87
AES256	17,74	38,92	1336 (1.019%)	312 (1.935%)	1892 (1.443%)	312 (1.935%)	36,6	36,06
Chacha20	1,78	1,78	960 (0.732%)	308 (1.910%)	964 (0.735%)	308 (1.910%)	40,39	40,79
SHA256	3,34	NA	1072 (0.818%)	148 (0.918%)	NA	NA	37,35	NA
RSA	16,01	100,39	1128 (0.861%)	20 (0.124%)	1528 (1.166%)	524 (3.250%)	36,15	34,36
ECC	1998,66	978,33	10920 (8.331%)	284 (1.761%)	5984 (4.565%)	144 (0.893%)	35,14	36,71

El STM32F103CB, con arquitectura ARM Cortex-M3 de 32 bits, ofrece resultados intermedios. Aunque sus tiempos de ejecución no alcanzan la eficiencia del ESP32, supera ampliamente al Arduino UNO tanto en velocidad como en uso de memoria. Su consumo de corriente, cercano a los 35 mA, lo posiciona como una opción adecuada para aplicaciones embebidas que requieren un equilibrio entre autonomía energética y capacidad de cómputo.

- **Resultados de tiempo, memoria y corriente para el Arduino Uno R3**

Tabla 4.
Resultados Arduino Uno R3 (ATmega328p).

	Tiempo de cifrado [Byte/us]	Tiempo de descifrado [Byte/us]	Flash en bytes al cifrar [%]	RAM en bytes al cifrar [%]	Flash en bytes al descifrar [%]	RAM en bytes al descifrar [%]	Corriente promedio (cifrado) [mA]	Corriente promedio (descifrado) [mA]
AES128	62,31	196,02	2084 (6.461%)	665 (32.614%)	2538 (7.868%)	921 (45.169%)	21,65	21,04
AES192	74,08	237,50	2178 (6.752%)	705 (34.576%)	2632 (8.160%)	961 (47.131%)	21,49	21,36
AES256	85,83	278,97	2172 (6.734%)	745 (36.538%)	2626 (8.141%)	1001 (49.093%)	21,92	21,22
Chacha20	58,76	58,76	3070 (9.518%)	347 (17.018%)	3070 (9.518%)	347 (17.018%)	20,19	19,97
SHA256	339,92	NA	4738 (14.689%)	637 (31.241%)	NA	NA	19,85	NA
RSA	449,18	4502,62	1918 (5.946%)	243 (11.918%)	2448 (7.589%)	687 (33.693%)	19,42	19,47
ECC	279398,34	121101,49	7164 (22.210%)	274 (13.438%)	6522 (20.219%)	458 (22.462%)	19,48	19,55

En contraste, el Arduino UNO, basado en un microcontrolador de 8 bits con recursos limitados de memoria y baja frecuencia de operación, presenta un rendimiento significativamente inferior. Los tiempos de ejecución de los algoritmos se incrementan de forma considerable, en especial los asimétricos como RSA y ECC, cuyos valores resultan órdenes de magnitud mayores. El elevado porcentaje de memoria utilizada que supera casi el 50% de la disponibilidad expuesta en la Tabla 1 compromete la viabilidad del sistema en escenarios que demanden escalabilidad, reflejando así las limitaciones inherentes a su arquitectura. No obstante, un resultado llamativo se observó en el consumo de corriente, que fue inferior al registrado en microcontroladores de 32 bits con arquitecturas más avanzadas. Esto indica que, aunque el Arduino UNO no es viable en

términos de tiempo, memoria y capacidad de cómputo, su comportamiento en consumo de corriente resulta más favorable en procesos de cifrado concretos.

- **Resultados de tiempo, memoria y corriente para la Raspberry Pi Pico**

Tabla 5.

Resultados Raspberry Pi Pico (RP2040).

	Tiempo de cifrado [Byte/us]	Tiempo de descifrado [Byte/us]	Flash en bytes al cifrar [%]	RAM en bytes al cifrar [%]	Flash en bytes al descifrar [%]	RAM en bytes al descifrar [%]	Corriente promedio (cifrado) [mA]	Corriente promedio (descifrado) [mA]
AES128	6,59	14,44	1236 (0.059%)	224 (0.085%)	1660 (0.079%)	224 (0.085%)	22,40	22,47
AES192	7,75	17,31	1236 (0.059%)	256 (0.098%)	1660 (0.079%)	256 (0.098%)	22,35	22,56
AES256	8,95	20,26	1236 (0.059%)	304 (0.116%)	1660 (0.079%)	304 (0.116%)	22,48	22,46
Chacha20	0,98	0,98	548 (0.026%)	64 (0.024%)	564 (0.027%)	64 (0.024%)	22,79	22,83
SHA256	2,18	NA	988 (0.047%)	144 (0.055%)	NA	NA	23,15	NA
RSA	7,5	63,8	228 (0.011%)	4 (0.002%)	172 (0.008%)	128 (0.049%)	22,67	22,5
ECC	1915,54	876,49	4360 (0.208%)	124 (0.047%)	4000 (0.191%)	124 (0.047%)	23,02	23,2

Por último, la Raspberry Pi Pico, con un ARM Cortex-M0+, presenta métricas competitivas, especialmente en algoritmos simétricos como AES y ChaCha20, donde alcanza resultados comparables al ESP32, aunque con un consumo ligeramente superior (22–23 mA). Esto confirma que plataformas más recientes, incluso dentro de gamas de bajo costo, pueden resultar altamente adecuadas para tareas criptográficas, siempre que se seleccione un algoritmo acorde a sus capacidades.

- **Resultados de consumo energético**

A continuación, se presentan los resultados asociados al consumo energético, estimados mediante una métrica aproximada de energía en nano *Joules* (nJ). Esta medida permite evaluar, el costo energético de los procesos de cifrado y descifrado en cada microcontrolador. Con ello, se amplía el análisis más allá de la velocidad de ejecución, integrando también el impacto energético como un factor clave en la comparación de algoritmos y microcontroladores.

Tabla 6.

Consumo energético estimado por byte (Cifrado)

	AES256 [nJ]	Chacha20 [nJ]	SHA256 [nJ]	RSA [nJ]	ECC [nJ]
ESP32	955,89	110,21	198,77	644,25	122451,70
Arduino UNO	9406,96	5931,82	33737,06	43615,37	27213398,00
STM32	2142,63	237,25	411,67	1909,91	231768,61
Raspberry	1005,98	111,67	252,33	850,13	220478,65

Tabla 7.

Consumo energético estimado por byte (Descifrado)

	AES256 [nJ]	Chacha20 [nJ]	RSA [nJ]	ECC [nJ]
ESP32	2340,95	109,62	4355,89	59847,30
Arduino UNO	29598,71	5867,18	438330,06	11837671,00
STM32	4631,40	239,60	11383,02	118517,83
Raspberry	2275,19	111,86	71775,00	101672,84

Los resultados muestran que, por un lado, el ESP32 y la Raspberry Pi Pico son las plataformas más competitivas, con bajos tiempos y consumo en algoritmos simétricos gracias a sus arquitecturas modernas. En cambio, el Arduino UNO evidencia un rendimiento limitado que lo descarta para operaciones criptográficas exigentes.

Por otro lado, en cuanto a los algoritmos, AES y ChaCha20 se confirman como los más eficientes en todas las plataformas, adecuados para comunicaciones seguras y cifrado continuo. SHA-256, aunque más demandante, mantiene un desempeño aceptable en microcontroladores de

32 bits. Finalmente, RSA y, sobre todo, ECC registran el mayor costo energético lo que refleja la complejidad matemática que los caracteriza.

Por último, para hashing con SHA-256, el STM32 y la Raspberry logran el mejor equilibrio entre consumo energético y uso de memoria, superando al ESP32 y al Arduino. Así pues, en algoritmos de clave pública, solo resultan viables el ESP32 con ECC y la Raspberry con RSA/ECC. El Arduino UNO queda totalmente descartado, pues sus altos costos computacionales y energéticos impiden su aplicación práctica en entornos industriales.

6. Conclusiones Y Perspectivas

- Los algoritmos simétricos, particularmente AES y ChaCha20, demostraron ser los más apropiados para entornos embebidos, al ofrecer un balance óptimo entre velocidad de ejecución, consumo de memoria y eficiencia energética.
- Plataformas modernas como ESP32 y Raspberry Pi Pico se consolidan como las más competitivas en el uso de criptografía, al sostener cargas criptográficas intensivas en tiempo real con un consumo energético moderado.
- El algoritmo de hash SHA-256 se posiciona como una alternativa intermedia, adecuada para aplicaciones centradas en la integridad de datos, tales como la verificación de firmware o la autenticación ligera.
- Microcontroladores como el STM32 y la Raspberry Pi Pico mostraron un desempeño equilibrado entre recursos computacionales y consumo energético al implementar funciones hash.

- Los resultados confirman que, aunque RSA y ECC no son viables para cifrado masivo en microcontroladores por su alto costo computacional y energético, mantienen utilidad en operaciones puntuales de intercambio de claves, autenticación y firmas digitales, siendo únicamente ESP32 y Raspberry adecuados para estas tareas, mientras que Arduino UNO queda descartado para la mayoría de procesos criptográficos por sus limitaciones en hardware.
- La combinación Microcontrolador–Algoritmo permite concluir que, para cifrado masivo de datos en tiempo real, las configuraciones más adecuadas son el ESP32 y la Raspberry Pi Pico junto con AES o ChaCha20, al equilibrar velocidad y eficiencia energética. En escenarios donde la prioridad es la autonomía y el bajo consumo, el STM32 con AES se presenta como una opción competitiva, especialmente en sistemas embebidos alimentados por baterías.

Este trabajo abre la puerta a explorar nuevos escenarios que profundicen en el análisis de la criptografía en entornos embebidos. Entre ellos, la incorporación de un sistema operativo en tiempo real (RTOS) permitiría comprender cómo la planificación de tareas y la concurrencia afectan tanto el rendimiento criptográfico como el consumo energético en dispositivos IoT. Asimismo, la evaluación de algoritmos post-cuánticos en microcontroladores surge como un desafío crucial para anticipar su eficiencia y viabilidad práctica frente a los esquemas tradicionales.

Bibliografía

Annigeri, S. (2021, enero). Repositorio GitHub: tinyECC-ArduinoIDE. Consultado el 5 de junio de 2024 en https://github.com/ShubhamAnnigeri/tinyECC-ArduinoIDE

Cadence (2016). Xtensa LX6 Customizable DPU. Consultado el 5 de mayo de 2024 en www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

Chaturvedi, S. P., Mukherjee, R. y Yadav, A. (2022). Comparison between Ultra Lightweight Cryptographic Techniques on Microcontrollers for Smart Agriculture. Proceedings of WIECON-ECE, Naya Raipur, India, pp. 62-65. doi: doi.org/10.1109/WIECON-ECE57977.2022.10151441

Dworkin, M. (2023, mayo). Federal Inf (NIST FIPS): Advanced Encryption Standard (AES). Consultado el 26 de mayo de 2024 en tsapps.nist.gov/publication/get_pdf.cfm?pub_id=936594

Gouvêa, C. P. L, Oliveira, L. B. y López, J. (2012). Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. Rev. Springer, vol. 2, J Cryptogr Eng 2, pp 19–29. doi: https://doi.org/10.1007/s13389-012-0029-z

Guajardo, J., Blümel, R., Krieger, U., Paar, C. (2001). Efficient Implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x Family of Microcontrollers. In: Kim, K. (eds) Public Key Cryptography. PKC 2001. Lecture Notes in Computer Science, vol 1992. Springer, Berlin, Heidelberg. doi: https://doi.org/10.1007/3-540-44586-2_27

Ionescu, V. M. y Enescu, F. M. (2020). Investigating the performance of MicroPython and C on ESP32 and STM32 microcontrollers. Proceedings of SIITME, Pitesti, Romania, pp. 234-237 doi: https://doi.org/10.1109/SIITME50350.2020.9292199

Kapoor, V., Abraham, V. S., y Singh, R. (2008). Elliptic curve cryptography. Ubiquity, Vol. 2008, No. May, pp. 1-8. doi: doi.org/10.1145/1386853.1378356

Marqas, R. B., Almufti, S. M., & Ihsan, R. R. (2020). Comparing Symmetric and Asymmetric cryptography in message encryption and decryption by using AES and RSA algorithms. Xi'an Jianzhu Keji Daxue Xuebao/Journal of Xi'an University of Architecture & Technology, 12(3), 3110-3116.

Microchip. (2024). ATmega328P. Consultado el 20 de mayo de 2024 en https://www.microchip.com/wwwproducts/en/ATmega328p

NIST (2024, octubre). NIST Announces 14 Candidates to Advance to the Second Round of the Additional Digital Signatures for the Post-Quantum Cryptography Standardization Process. Consultado el 1 de mayo de 2025 en www.nist.gov/news-events/news/2024/10/nist-announces-14-candidates-advance-second-round-additional-digital

Nordic Semiconductor (2024). Power Profiler Kit II. Consultado el 10 de diciembre de 2024 en docs.nordicsemi.com/bundle/ug_ppk2/page/UG/ppk/PPK_user_guide_Intro.html

Olivarez, P. D. G., Gil, A. J. L., & De Los Santos, A. C. M. (2023). Principales técnicas criptográficas aplicadas a la seguridad de la información en IoT: una revisión sistemática. Ingenio Tecnológico, Vol. 5. doi: portal.amelica.org/ameli/journal/266/2663842005/

Rana, M., Mamun, Q., Islam, R. (2022). Lightweight cryptography in IoT networks: A survey, Future Generation Computer Systems, Vol. 129, pp 77-89. doi: doi: doi.org/10.1016/j.future.2021.11.011

Rao, UH, Nayak, U. (2014). Cryptography. In: The InfoSec Handbook. Apress, Berkeley, CA, pp 168-172. doi: doi.org/10.1007/978-1-4302-6383-8_8

Raspberry Pi Ltd. (2024). Raspberry Pi Pico Datasheet An RP2040-based microcontroller board. Consultado el 20 de mayo de 2024 en datasheets.raspberrypi.com/pico/pico-datasheet.pdf

Rueda-Rueda, J. S. (2021). El reto del desarrollo seguro de aplicaciones IoT en un mercado acelerado. Ingenio, Vol. 18, No. 1, pp 54-61. doi: doi: doi.org/10.22463/2011642X.2667

Sklavos, N. y Koufopavlou, O. (2003). On the hardware implementations of the SHA-2 (256, 384, 512) hash functions. Proceedings of ISCAS '03, Vol. 5, Bangkok, Thailand, pp. V-V. doi: doi.org/10.1109/ISCAS.2003.1206214

STMicroelectronics. (2019, abril). STM32F103CB. Consultado el 20 de agosto de 2025 en www.alldatasheet.com/datasheet-pdf/pdf/513835/STMICROELECTRONICS/STM32F103CB.html

Universidad de Valencia. Introducción a la criptología. Consultado el 20 de diciembre de 2024 en www.uv.es/~sto/articulos/BEI-2003-04/criptologia.html

Wang, L., Zhao, H. y Bai, G. (2007). A cost-Efficient Implementation of Public-key Cryptography on Embedded Systems. Proceedings of EDST, Beijing, China, pp. 194-197. doi: dx.doi.org/10.1109/EDST.2007.4289808

Yadav, P., Gupta, I. y Murthy, S. K. (2016). Study and analysis of eSTREAM cipher Salsa and ChaCha. Proceedings of ICETECH, Coimbatore, India, pp. 90-94. doi: doi.org/10.1109/ICETECH.2016.7569218

Anexos

Apéndice A. Video demostrativo en CyberChef: https://youtu.be/s6Zq82rp6A8

Apéndice B. Repositorio del proyecto: https://github.com/Sierra-CMSJ/CSIoT

Apéndice C. Video guía instalación de librerías: https://youtu.be/z1Gd2wGIqZI

Apéndice D. Video guía del análisis con PPK2: https://youtu.be/WEodeSfAtvQ

Apéndice E. Anexos y Entregables: https://drive.google.com/drive/folders/1DIblQokUGU_RMYDIUyA5nVtY3Ftu6Izd?usp=drive_link