UNIVERSIDAD NACIONAL DE COLOMBIA

Proyecto criptografía
Primer Avance

Análisis Comparativo de Algoritmos de Criptografía Ligera

Kevin Arturo Amaya Osorio

1 Resumen

La Criptografía Ligera (Lightweight Cryptography, LWC) constituye un campo de investigación enfocado en desarrollar esquemas de cifrado robustos que minimicen drásticamente los recursos computacionales necesarios para su ejecución. Este campo ha cobrado relevancia debido a la proliferación de dispositivos con capacidad de cómputo limitada, como los sensores de Internet de las Cosas (IoT), redes vehiculares y dispositivos de monitoreo médico, que requieren asegurar sus comunicaciones a través de canales inseguros [1]. A diferencia de la criptografía clásica (e.g., RSA), los algoritmos LWC deben equilibrar no solo la seguridad del mensaje, sino también métricas críticas de eficiencia como el consumo de energía, la latencia y la restricción de recursos físicos (procesamiento, RAM, ROM y almacenamiento).

Debido a las limitaciones de recursos, los algoritmos LWC pueden presentar vulnerabilidades que no son típicas de esquemas criptográficos robustos tradicionales. Entre ellas se incluyen: ataques basados en la reutilización de parámetros [2], ataques multiset [3] o incluso ataques de fuerza bruta [4]. Debido a la menor robustez inherente en términos de seguridad de estos esquemas, es importante evaluar el riesgo asumido al integrarlos en arquitecturas de comunicación. Por lo tanto, este trabajo se centrará en explorar la naturaleza de estos compromisos de seguridad y de rendimiento en sistemas de bajos recursos computacionales.

2 Objetivos

- Caracterizar el panorama de la Criptografía Ligera (LWC), identificando los algoritmos simétricos predominantes en entornos IoT, sus parámetros operativos clave y los desafíos de seguridad que pretenden mitigar.
- Establecer una comparativa entre el rendimiento computacional (latencia, consumo energético) y la robustez criptográfica de los algoritmos seleccionados, prestando especial atención a las vulnerabilidades introducidas por las implementaciones en hardware.
- Clasificar y documentar los principales vectores de ataque dirigidos a estos esquemas.

3 Algoritmos analizados

En este proyecto, el análisis se centrará exclusivamente en **algoritmos de encriptación simétricos**, ya que eliminan la necesidad de procedimientos de intercambio de claves separados y generalmente requieren menos recursos computacionales para su implementación en comparación con sus contrapartes asimétricos [1].

Los cifrados simétricos se clasifican en dos tipos principales: cifrados en bloque y cifrados de flujo. Los **cifrados en bloque** dividen el mensaje original en bloques de n bits de tamaño fijo para su procesamiento, y los de flujo operan secuencialmente, cifrando típicamente bit a bit o byte a byte, sin tamaños fijos. Los cifrados de bloque son considerados el estándar en una amplia mayoría de aplicaciones, impulsados por la popularidad de algoritmos como DES [1]. Aunque no se han demostrado debilidades conocidas en las implementaciones más

comunes de algoritmos simétricos, este sigue siendo un campo de investigación activo, el estado del arte actual se enfoca en el desarrollo y análisis de diversos ataques como los algebraicos [3].

Definición 1 (Cifrado de Bloque de n-bits)**.** *Sea k la longitud de la clave (en bits) y n la longitud del bloque (en bits).*

- *V_n es el conjunto de todos los posibles vectores binarios de n-bits. Este conjunto representa el espacio de texto plano y el espacio de texto cifrado.*
- *V_k es el conjunto de todos los posibles vectores binarios de k-bits. Por ejemplo para $k = 3$, $V_k = \{000, 001, 010, 011, 100, 101, 111\}$*
- *$\mathcal{K}$ es el **espacio de claves válidas**, un subconjunto del conjunto V_k ($\mathcal{K} \subseteq V_k$).*

*Un **cifrado de bloque n-bit** es una función $E : V_n \times \mathcal{K} \to V_n$ que opera sobre un bloque de texto plano $P \in V_n$ y una clave $K \in \mathcal{K}$. El resultado de la función es un bloque de texto cifrado C, lo cual se expresa como $C = E(P, K)$ o, de forma más concisa, $C = E_K(P)$. Esta función debe cumplir la condición de que, para cada clave $K \in \mathcal{K}$ fija, la función de cifrado $E_K(\cdot)$ es una **biyección** de V_n a V_n. La función inversa es la función de descifrado, denotada $D_K : V_n \to V_n$, para la cual se cumple que*

$$D_K(E_K(P)) = P$$

para todo bloque de texto plano $P \in V_n$.

Por otra parte, los **cifrados de flujo** representan un campo de estudio criptográfico relativamente más reciente. Históricamente, su implementación ha sido menos estandarizada y, en general, han sido percibidos como menos seguros que los cifrados de bloque. No obstante, los cifrados de flujo ofrecen mayor velocidad de cifrado y demandan menor memoria. Si bien perdieron protagonismo como estándar frente al auge del DES, han experimentado un resurgimiento en popularidad. Esta tendencia se debe al creciente énfasis en el cifrado en tiempo real para aplicaciones modernas como la transmisión (streaming) de video y audio.

El proyecto actual implementa exclusivamente **algoritmos de encriptado en bloque simétricos**, dado que los estándares dominantes en LWC son en su mayoría en bloque y los algoritmos de flujo todavía son un campo en desarrollo jóven. Los algoritmos que se van a explorar son:

1. **Data Encryption Standard (DES):**, derivado del algoritmo LUCIFER, fue publicado oficialmente el 15 de julio de 1977 por la Oficina Nacional de Estándares (NBS), y posteriormente adoptado como el estándar federal de cifrado por las agencias gubernamentales y entidades bancarias de EE.UU. [4].

   El DES utiliza una clave de 64 bits, de los cuales 56 bits son empleados para el secreto criptográfico y los 8 bits restantes se destinan a la corrección o verificación de paridad. La principal debilidad del algoritmo radicaba en la corta longitud de su clave (*56 bits*). Ya en el momento de su publicación, criptógrafos como *Diffie y Hellman* habían señalado la viabilidad teórica de construir una máquina capaz de romper la encriptación mediante un ataque de fuerza bruta [5].

   Un componente fundamental del *DES* es el *cifrado de Feistel*, el cual constituye un componente esencial en una amplia gama de algoritmos de cifrado por bloques. El funcionamiento general del algoritmo se describe a continuación [6]:

Algorithm 1 Estructura de Feistel

Entradas:

- P el bloque de texto plano de longitud $2m$ bits.
- N el número total de rondas.
- $K_1, K_2, \cdots, K_N$ las subclaves de ronda.
- F la función de ronda, $F : \{0,1\}^m \times \{0,1\}^k \to \{0,1\}^m$

Salida: Un mensaje encriptado C.

Procedimiento:

1) Sea $i = 0$
2) L_0 la mitad izquierda de P (m bits).
3) R_0 la mitad derecha de P (m bits).
4) Se calcula la nueva mitad
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$
5) Se asigna una nueva mitad izquierda como
$$L_i = R_{i-1}$$
6) Asignar $i = i + 1$. Si $i > N$, se detiene el algoritmo.

La *función de ronda* F implementa operaciones XOR usan partes de la clave original K para ofuscar la información original. Para el algoritmo *DES* se usan cajas de sustitución (*S boxes*) y permutaciones que idealmente impiden el cálculo de una función inversa que descifra el mensaje C sin el conocimiento de la clave K.

Para el algoritmo estándar DES se computan $N = 16$ rondas de Feistel, la implementación es dada por [6]

Algorithm 2 Algoritmo DES

Entradas:

- El mensaje $P = p_1, \cdots, p_{64}$.
- La clave $K = k_1, \cdots, k_{64}$.

Salida: Un mensaje encriptado $C = c_1, \cdots, c_{64}$.

Procedimiento:

1) L_0 la mitad izquierda de P (32 bits).
2) R_0 la mitad derecha de P (32 bits).
3) Se usa la tabla IP definida en el estándar [7] para generar un mensaje P' que es una permutación del mensaje original P (ej. la entrada 0 de la tabla es 58, por lo tanto el bit $p'_0 = p_{58}$). Este paso se emplea para incrementar la *difusión* del mensaje original.
4) Se toma el mensaje permutado P' y se divide en dos partes: $L_0 = p'_{33} \cdots p'_{64}$ y $R_0 = p'_0 \cdots p'_{32}$
5) Sea $N = 16$ y $i = 0$.
6) $i = i + 1$
7) Usando la tabla E de selección de bits dada en el estándar [7], se tiene que, para la parte derecha $R_{i-1} = r_1, \cdots, r_{32}$, $T = r_{32}, r_1, r_2 \cdots r_{32}, r_1$.
8) Sea $T' = T \oplus K_1$
9) Se definen $(B_1, \cdots, B_8) = T'$, donde B_i son representaciones de 6 bits
10) $T'' = S_1(B_1), \cdots, S_8(B_8)$. Este mapeo usa cajas de S_1 a S_8 que son definidas en el apéndice 1 del estándar [7].
11) Se usa la primitiva función P definida en el estándar apéndice 1 [7] para definir $T''' = P(T'')$
12) Si $i > N$, entonces se pasa al paso (12)
13) Se intercambian los bloques L_{16} y R_{16} y se retorna el mensaje usando la tabla IP^{-1}

$$C = IP^{-1}(b_1, \cdots, b_{64})$$

Los pasos 6 al 11 se pueden ver como DES aplica una **estructura de Feistel** [1] 16 veces. En la aplicación real del algoritmo se requiere generar subclaves de K cada cierto periodo de tiempo. Este paso se omite para la implementación en este proyecto y se pueden usar subclaves precalculadas, sin embargo, en las aplicaciones reales es importante notar que no se puede omitir este paso y debe hacerse regularmente.

El algoritmo DES adquirió gran importancia por ser uno de los primeros estándares criptográficos de bloque publicados y ampliamente difundidos. Este hecho impulsó un intenso interés académico, lo que se tradujo tanto en el desarrollo de hardware optimizado para su implementación como en la investigación de métodos de criptoanálisis [3]. Aunque el DES original se considera obsoleto debido a su corta longitud de clave de 64 bits, los principios fundamentales de su diseño no lo son. La arquitectura subyacente, la red de Feistel [1], sigue siendo un pilar en la criptografía simétrica moderna, siendo utilizada en numerosos algoritmos de bloque de Criptografía Ligera.

Décadas de desarrollo han mitigado varias de sus vulnerabilidades principales del DES. La solución más notable es el *Triple-DES (3DES)*, que consiste en aplicar el cifrado DES tres veces, corrigiendo efectivamente la debilidad de la longitud de clave y eliminando posibles ataques por fuerza bruta, aunque aplicar *3DES* requiere más recursos y, por lo tanto, no es viable para varias aplicaciones de LWC [4]. El interés por mantener y mejorar las implementaciones basadas en DES radica en la gran cantidad de hardware optimizado para este estándar [3], el reaprovechamiento de este hardware, especialmente en dispositivos con restricciones severas de recursos, ofrece ventajas considerables en términos de ahorro en tiempo de desarrollo y escalabilidad.

2. **Advanced Encryption Standard (AES):**

Debido a las múltiples vulnerabilidades detectadas en el estándar DES, un hecho que quedó demostrado con el desarrollo del "DES-Cracker" [5], se anunció una iniciativa para seleccionar un nuevo estándar. El objetivo era que este nuevo algoritmo sustituyera a DES como el nuevo **Estándar Federal de Proce-**

samiento de Información (FIPS) [8]. Sin embargo, es importante aclarar que este nuevo estándar no se concibió como un reemplazo total o directo de DES, ya que su alcance era más limitado. Se buscaba que fuera utilizado principalmente para documentos oficiales que fuesen secretos, pero no necesariamente clasificados. A diferencia de DES, este nuevo estándar tampoco fue diseñado para cubrir otras funciones específicas, como los protocolos de cifrado para información bancaria.

El estándar que finalmente resultó ganador fue el algoritmo Rijndael. Este fue seleccionado porque demostró funcionar eficientemente en una amplia gama de hardware, incluyendo dispositivos de bajas especificaciones, sin comprometer críticamente su seguridad. Además, el hecho de que este algoritmo esté disponible de forma libre y sin patentes, ha sido un factor clave que ha contribuido a su rápida y amplia adopción [8].

A diferencia del estándar DES, el algoritmo desarrollado por Rijndael tiene tres distintas especificaciones respecto a la longitud de la clave dependiendo de los bits: AES-128, AES-192 y AES-256. Para esta comparación en el marco de Criptografía Ligera, se busca implementar AES-128, con una clave K de 128 bits, un tamaño de bloque de 4 y usan 10 rondas. El algoritmo es descrito como [9]:

Algorithm 3 Algoritmo AES

Entradas:

- El mensaje (bloque de datos) P, de 128 bits (16 bytes).
- La clave K, de N_k bytes ($16, 24$ o 32 bytes para $128, 192$ o 256 bits, respectivamente).

Salida: Un mensaje encriptado C, de 128 bits.

Parámetros:

- El número de rondas N_r: 10 para clave de 128 bits, 12 para 192 bits, 14 para 256 bits.
- El array de 4×4 bytes llamado **State**.
- El **RoundKey** para cada ronda, derivado de K por la Expansión de Clave.

Procedimiento (AES-128 para $N_r = 10$):

1) **Inicialización:** El bloque de entrada P se copia al array **State**.
2) **Expansión de Clave (Key Expansion):** Generar $N_r + 1$ claves de ronda $\mathbf{RoundKey}_0, \cdots, \mathbf{RoundKey}_{N_r}$.
3) **Ronda Inicial:**
$$\mathbf{State} \leftarrow \mathbf{State} \oplus \mathbf{RoundKey}_0$$
4) **Rondas Principales ($i = 1$ hasta $N_r - 1$):** Repetir la secuencia de cuatro transformaciones:
   a. **SubBytes:** Cada byte en **State** se sustituye usando una S-box (Substitution Box) fija.
   b. **ShiftRows:** Las filas del **State** se desplazan cíclicamente un número variable de bytes.
   c. **MixColumns:** Se realiza una mezcla lineal de las columnas del **State** (multiplicación polinomial en $\mathrm{GF}(2^8)$).
   d. **AddRoundKey:** Se aplica la operación XOR al **State** con el $\mathbf{RoundKey}_i$ de la ronda actual.
5) **Ronda Final ($i = N_r$):** Se repiten las primeras tres transformaciones, pero se **omite** la **MixColumns**:
   a. **SubBytes**
   b. **ShiftRows**
   c. **AddRoundKey:** Se aplica XOR al **State** con el $\mathbf{RoundKey}_{N_r}$ final.
6) **Salida:** El array **State** final es el mensaje cifrado C.

Cada una de las operaciones de SubBytes, ShiftRows, MixColumns y AddRoundKey son transformaciones lineales que están definidas en el estándar de AES [9].

Por lo tanto, a diferencia del DES, que utiliza una Red de Feistel, el algoritmo Rijndael (AES) se basa en una Red de Sustitución-Permutación (SPN). La diferencia conceptual clave radica en cómo procesan el bloque de datos en cada ronda. Una red Feistel divide el bloque en dos mitades (L/R) y, en cada ronda, aplica las transformaciones solo a una mitad, la cual luego se combina con la otra mitad (que pasó intacta). En cambio, una red SPN trata el bloque de datos completo como una sola unidad (llamada

"Estado" en AES) y aplica capas de transformaciones (sustituciones y permutaciones) a todo el estado en paralelo durante cada ronda.

En general los algortimos SPN son dados de la siguiente manera [10]

Algorithm 4 Algoritmo de Red de Sustitución-Permutación (SPN)

Entradas:

- P el bloque de texto plano de longitud m bits.
- N el número total de rondas.
- $K_0, K_1, \cdots, K_N$ las $N+1$ subclaves de ronda.
- S la función de sustitución (S-BOX), $S : \{0,1\}^m \to \{0,1\}^m$.
- L la capa de permutación/lineal (P-BOX), $L : \{0,1\}^m \to \{0,1\}^m$.

Salida: Un mensaje encriptado C.

Procedimiento:

1) Sea W el estado intermedio, $W = P$.
2) **Mezcla de Clave Inicial (Whitening):**

$$W = W \oplus K_0$$

3) **Rondas Principales:** Para $i = 1$ hasta $N - 1$:
   - *(Sustitución)* $W = S(W)$
   - *(Permutación)* $W = L(W)$
   - *(Mezcla de Clave)* $W = W \oplus K_i$
4) **Ronda Final (sin permutación):**
   - *(Sustitución)* $W = S(W)$
   - *(Mezcla de Clave)* $W = W \oplus K_N$
5) El cifrado resultante es $C = W$.

Esta estructura SPN es mucho más adecuada para el procesamiento paralelo. Esto permite que los algoritmos que la usan, como AES, sean generalmente más rápidos y eficientes, ya que sus operaciones (ej. sustituir 16 bytes a la vez) son fáciles de optimizar y ejecutar simultáneamente en el hardware moderno [8].

3. **PRESENT**

El algoritmo **PRESENT** marcó un paso importante en la evolución de los cifrados por bloques, redefiniendo las prioridades del diseño criptográfico. A diferencia de sus predecesores, que se optimizaban principalmente para la velocidad en software, el desarrollo de PRESENT se centró en la eficiencia de hardware, es decir, minimizar el espacio físico (área) requerido para su implementación en un chip.

Esta eficiencia se mide en **Equivalentes en Puertas (GE)**, y el artículo que introduce **PRESENT** [2] utiliza esta métrica para demostrar su ventaja de forma contundente. Mientras que los estándares establecidos como AES requerían aproximadamente 3600 GE y DES unos 3200 GE, PRESENT logró una implementación funcional con solo 2280 GE [2].

Esta drástica reducción en el costo de hardware permitió, por primera vez, integrar criptografía robusta en dispositivos de recursos extremadamente limitados (como etiquetas RFID o nodos de sensores), mejorando bastante la seguridad de un ecosistema de dispositivos.

La implementación del algoritmo es dada por [11]:

Algorithm 5 Algoritmo PRESENT (Cifrado)

Entradas:

- El mensaje (bloque de datos) P, de 64 bits.
- La clave K, de 80 bits (para la versión principal).

Salida: Un mensaje encriptado C, de 64 bits.

Parámetros:

- El número de rondas N_r: 31.
- El registro de estado (**State**), de 64 bits.
- 32 subclaves de ronda ($\mathbf{RoundKey}_1, \ldots, \mathbf{RoundKey}_{32}$), generadas por la Expansión de Clave.
- La **SBOX** (sustitución de 4-bits).
- La $\mathbf{P-LAYER}$ (permutación de 64-bits).

Procedimiento (PRESENT-80 para $N_r = 31$):

1) **Inicialización:** El bloque de entrada P se copia al registro **State**.
2) **Expansión de Clave (KeySchedule_80bit):** Generar 32 claves de ronda $\mathbf{RoundKey}_1, \ldots, \mathbf{RoundKey}_{32}$ a partir de K.
3) **Rondas Principales ($i = 1$ hasta N_r (31)):** Repetir la secuencia de tres transformaciones:
   a. **AddRoundKey:** Se aplica la operación XOR al **State** con el $\mathbf{RoundKey}_i$ de la ronda actual.
   $$\mathbf{State} \leftarrow \mathbf{State} \oplus \mathbf{RoundKey}_i$$
   b. **sBoxLayer:** Cada uno de los 16 nibbles (4-bits) en **State** se sustituye usando la **SBOX**.
   c. **pLayer:** Los 64 bits del **State** se permutan según la **P_LAYER**.
4) **Ronda Final (Post-blanqueo):** Se aplica la última clave de ronda:
   a. **AddRoundKey:** Se aplica XOR al **State** con el $\mathbf{RoundKey}_{32}$ final.
   $$\mathbf{State} \leftarrow \mathbf{State} \oplus \mathbf{RoundKey}_{32}$$
5) **Salida:** El registro **State** final es el mensaje cifrado C.

Al igual que otros estándares de cifrado, los componentes fundamentales de PRESENT, tales como la SBOX, el algoritmo de expansión de clave (que genera 32 claves de ronda o *Round Keys*) y la capa de permutación ($P\text{-}LAYER$), están todos definidos en su especificación oficial [2]. PRESENT soporta claves de 128 bits; esta comparativa se centra en su versión de 80 bits.

PRESENT, un estándar desde 2007, fue durante una década uno de los algoritmos más influyentes en el desarrollo de la *Criptografía Ligera*. A diferencia de AES, no fue diseñado para un uso universal, sino para aplicaciones específicas. Esto se evidencia en su uso de operaciones nativas de 4 bits, las cuales, si bien son lentas en procesadores de 32 o 64 bits, son extremadamente eficientes en hardware con recursos muy limitados.

Este enfoque optimizado implica ciertos sacrificios de diseño, entre los que se incluyen:

- Dificultad para realizar un refresco de clave (*re-keying*) de forma regular, dependiendo de las limitaciones del dispositivo.
- No está optimizado para procesar grandes volúmenes de mensajes a alta velocidad (bajo *throughput*).
- Requeriría más espacio (hardware) del especificado en el estándar si se necesita gestionar un alto rendimiento.
- Menor velocidad de ejecución en comparación con AES en plataformas de 32 o 64 bits.

A cambio, PRESENT ofrece un estándar robusto y probado para dispositivos de baja potencia sin debilidades criptográficas significativas conocidas, siendo ideal para implementaciones en sensores y dispositivos de comunicación con recursos restringidos.

PRESENT, al igual que AES, reemplaza la clásica estructura de Feistel 1 por una red de sustitución-permutación (SPN) 4. Esta elección de diseño le confiere ventajas significativas en paralelismo. Sin embargo, existe una contrapartida: dado que sus operaciones internas manipulan datos en fragmentos de 4 bits, su rendimiento en implementaciones por software es probablemente inferior al de DES

4 Análisis comparativo

Para obtener métricas de rendimiento relevantes para dispositivos de recursos limitados, las pruebas se ejecutarán en una máquina de escritorio estándar x86-64, utilizando implementaciones en Python. Esto servirá para validar la lentitud teórica de PRESENT en software de propósito general, y la capacidad de adaptación de DES y AES a hardware moderno. Se medirán las siguientes métricas:

- **Rendimiento:** Se medirá la velocidad de cifrado y descifrado en Megabytes por segundo (MB/s) o ciclos de CPU por byte. Se utilizarán conjuntos de datos de varios tamaños (ej. 1KB, 128KB, 1MB) para evaluar la escalabilidad.
- **Uso de Memoria:** Se medirá el tamaño del código compilado (en Kilobytes) para cada algoritmo. Esta es una métrica crítica para LWC, ya que el almacenamiento en dispositivos IoT es muy limitado.
- **Uso de Memoria:** Se medirá el uso pico de RAM durante la ejecución del cifrado y descifrado para determinar la viabilidad del algoritmo en dispositivos con memoria volátil escasa (ej. $<$ 64KB RAM).

4.1 Tabla Comparativa de Resultados

Finalmente, los datos teóricos y empíricos recopilados se sintetizarán en una tabla comparativa la siguiente manera

Table 1: Ejemplo de Matriz de Comparación de Algoritmos

Métrica	**DES**	**AES-128**	**PRESENT-80**
Tipo de Estructura	Feistel	SPN	SPN
Tamaño de Bloque (bits)	64	128	64
Tamaño de Clave (bits)	56	128	80
Seguridad Teórica (bits)	56	128	80
Rondas	16	10	31
Rendimiento (Software) [MB/s]	*(experimental)*	*(experimental)*	*(experimental)*
Uso de ROM (Flash) [KB]	*(experimental)*	*(experimental)*	*(experimental)*
Uso de RAM (Pico) [KB]	*(experimental)*	*(experimental)*	*(experimental)*
Área en Hardware (GE)	~3200	~3600	**~2280**
Resistencia SCA	Baja	Media	Media

Para esta comparativa, las implementaciones se basarán estrictamente en los estándares publicados de cada algoritmo. Se ha decidido no utilizar bibliotecas criptográficas nativas (ej. OpenSSL) para evitar que optimizaciones de plataforma no especificadas sesguen los resultados del rendimiento. Además, las pruebas de rendimiento se ejecutarán utilizando el sistema de emulación QEMU. Esto permitirá evaluar los algoritmos en diversos entornos controlados que simulan fielmente las máquinas de bajos recursos.

References

[1] Jesús Soto Cruz et al. "A Survey of Efficient Lightweight Cryptography for Power-Constrained Microcontrollers". In: *Technologies* 13 (Dec. 2024). DOI: 10.3390/technologies13010003.

[2] A. Bogdanov et al. "PRESENT: An Ultra-Lightweight Block Cipher". In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466. ISBN: 978-3-540-74735-2.

[3] Alex Biryukov. "Block Ciphers and Stream Ciphers: The State of the Art". In: (May 2004).

[4] Orlin Grabbe. "The DES Algorithm Illustrated". In: 2006. URL: https://api.semanticscholar.org/CorpusID:63508688.

[5] Electronic Frontier Foundation (EFF). *EFF's DES Cracker and Distributed.Net Win DES Challenge III in 22 Hours and 15 Minutes*. https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html. Comunicado de prensa. Jan. 1999. (Visited on 11/01/2025).

[6] A J Menezes, Paul C, and Scott A Vanstone. *Handbook of applied cryptography*. Crc Press, 2001.

[7] National Institute of Standards and Technology. *Data Encryption Standard (DES)*. Federal Information Processing Standards Publication 46-3. Supersedes FIPS PUB 46-2. Gaithersburg, MD: U.S. Department of Commerce, Oct. 1999. URL: https://csrc.nist.gov/pubs/fips/46-3/final.

[8] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Berlin, Heidelberg: Springer-Verlag, 2002. ISBN: 3-540-42580-2.

[9] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. Tech. rep. 197. Updated version accessed 2025-11-02. Gaithersburg, MD: Federal Information Processing Standards Publication (FIPS), Nov. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.

[10] Douglas R. Stinson and Maura B. Paterson. *Cryptography: Theory and Practice*. 4th. See Section 4.2 "Substitution-Permutation Networks". Boca Raton, FL: CRC Press, Taylor & Francis Group, 2018. ISBN: 978-1-1381-9701-5.

[11] G. Jacinto, A. Montiel, and Fernando Martinez. "Implementation of the cryptographic algorithm "present" in different microcontroller type embedded software platforms". In: *International Journal of Applied Engineering Research* 12 (Jan. 2017), pp. 8092–8096.