AES Otimizado para Uso em Aplicações IoT

Yuri Silva Vaz, Júlio C. B. Mattos, Rafael Iankowski Soares
Programa de Pós-Graduação em Computação
Centro de Desenvolvimento Tecnológico
Universidade Federal de Pelotas (UFPel)
Pelotas, Brasil
{ysvaz, julius, rafael.soares}@inf.ufpel.edu.br

***Abstract*—A segurança das informações trocadas diariamente entre dispositivos IoT é uma preocupação real, visto que o número de dispositivos está crescendo exponencialmente, produzindo uma numerosa quantidade de dados que trafegam pela Internet. Uma possível solução para proteger estas informações é o uso de algoritmos criptográficos. No entanto, os algoritmos de criptografia clássicos são custosos computacionalmente não sendo adequados para execução em dispositivos IoT de baixo poder computacional e com restrições de armazenamento de dados. Neste contexto, este trabalho propõe otimizações no algoritmo criptográfico AES, de forma que sua implementação seja viável para execução em dispositivos com recursos computacionais restritos, assim, possibilitando a proteção de dados em aplicações IoT. As otimizações propostas são dedicadas especificamente aos estágios SubBytes e MixColumns. Os resultados obtidos permitem bons resultados em desempenho e ainda garantem boa qualidade da criptografia. Para validar a segurança do algoritmo com as modificações propostas é utilizado o teste de efeito avalanche, além de verificar sua melhor distribuição de 0's e 1's no texto cifrado comparado ao algoritmo original, e também é aprovado nos testes do NIST. Em termos de desempenho, é obtida uma redução média de 86,71% em tempo de execução, comparado à versão original do AES, o que torna esta versão otimizada atrativa para aplicações IoT.**

***Index Terms*—Internet das Coisas, Segurança, Criptografia Leve, Otimização**

I. Introdução

Internet das Coisas, do inglês *Internet of Things* (IoT), pode ser definida como uma variedade de dispositivos, tais como sensores e atuadores, que têm a capacidade de interação entre si, com a motivação de atingirem um objetivo comum [1]. Estas aplicações IoT vêm crescendo em número anualmente. No ano de 2022, já haviam 14,3 bilhões de dispositivos IoT ativos e, em 2027, espera-se que esse número atinja, aproximadamente, 29 bilhões de dispositivos [2]. Visto que há uma forte projeção de crescimento em número destas aplicações, é necessária a preocupação com a segurança dos dados que trafegam entre os dispositivos, pois uma aplicação IoT vulnerável pode acarretar em sérios problemas com vazamentos e manipulações dos dados [3].

Uma possível solução para proteção dos dados que trafegam pelas aplicações é o uso de algoritmos criptográficos. Contudo, estes algoritmos podem apresentar um impacto em desempenho [4], o que é um grande problema devido a grande maioria das aplicações IoT serem desenvolvidas em dispositivos com recursos computacionais limitados, além da maior parte dos dispositivos serem alimentados por baterias [5].

Visto que a criptografia pode ser prejudicial em termos de desempenho e consumo energético, surge então uma classe de algoritmos criptográficos focados em aplicações IoT e baixo consumo. Estes são chamados de *lightweight cryptography* (LWC), que em tradução direta seriam algoritmos de criptografia leve. Eles são um método de encriptação que prezam por ocupar pouco espaço de armazenamento e também por utilizar pouco recurso computacional [6].

De fato, a melhor solução encontrada pelos pesquisadores para enfrentar o problema de segurança em dispositivos com recursos computacionais limitados é utilizando LWC. Há uma imensa demanda por algoritmos de criptografia leves que sejam confiáveis e que tenham um bom custo-benefício. É muito mais eficiente aprimorar a segurança através de algoritmo criptográficos de baixo consumo do que substituir os dispositivos por algo com maior poder computacional [7].

Dos algoritmos criptográficos de chave simétrica, o AES é até hoje uma interessante solução de segurança, já tendo sido amplamente testado por agências governamentais e pela comunidade acadêmica. O AES possui um bom compromisso entre tempo de execução, simplicidade de implementação e nível de segurança comparado a outros algoritmos, apresentando uma estrutura de funções facilmente decompostas, o que o torna um dos algoritmos mais usados na literatura para as análises e avaliações [13].

Este artigo propõe uma versão otimizada do algoritmo AES, tanto em termos de desempenho como em termos de consumo de memória. Para isto, foram realizadas otimizações nas funções *SubBytes*, utilizando uma s-box menor, e *MixColumns*, utilizando operações menos custosas em termos de recursos computacionais. O diferencial deste artigo comparado aos demais é a preocupação em mais de uma otimização, sendo elas tanto em desempenho, melhorando o tempo de execução, quanto em consumo de memória, obtendo uma versão bem mais leve do algoritmo. Todas as otimizações foram validadas, aprovadas em requisitos de segurança e comparadas ao algoritmo AES versão original, além de comparar os resultados obtidos com outros artigos relacionados. Foi possível verificar uma redução considerável, tanto em termos de tempo de execução quanto em consumo de memória, também garantindo um ótimo nível de segurança, assim, sendo viável sua utilização em aplicações IoT.

A estrutura deste artigo é a seguinte: Capítulo II traz os conceitos do AES; Capítulo III apresenta as otimizações

propostas; Capítulo IV detalha os resultados obtidos; Capítulo V compara os resultados com outros trabalhos e, por último, Capítulo VI apresenta as conclusões e trabalhos futuros.

II. Algoritmo AES

O algoritmo *Advanced Encryption Standard* (AES) foi desenvolvido em 2001 por Vincent Rijmen e Joan Daemen e foi selecionado pelo NIST (*National Institute of Standards and Technology)* como o algoritmo padrão de criptografia. Ele é um algoritmo baseado em uma rede de substituição e permutação, que utiliza o conceito de chave simétrica, ou seja, a mesma chave é usada para encriptar e decriptar a informação.

A entrada do algoritmo é um bloco de 16 bytes, chamado de *plaintext*. Este bloco vai passar pelas 10 rodadas do algoritmo, onde cada rodada consiste em 4 estágios (com exceção da última rodada, que são apenas 3 estágios) que são: *Substitute Bytes* (SubBytes), *Shift Rows*, *Mix Columns* e *Add Round Key*. Ao final das 10 rodadas, o algoritmo tem como saída um bloco encriptado, chamado de *cipher text*.

O AES é um algoritmo de cifra de bloco, onde cada bloco possui um tamanho fixo de 128 bits (16 bytes). Estes 16 bytes de entrada são sempre mapeados em uma matriz 4x4, ordenados por colunas. A Fig. 1 exemplifica o processo de mapeamento de uma string de entrada, em hexadecimal, para a matriz 4x4. Esta matriz percorre por todos os estágios e rodadas do algoritmo, sendo nela realizadas todas as modificações que ocorrem em cada um dos estágios. Ao final das 10 rodadas, esta matriz de estado estará contendo o texto final encriptado.

Plaintext: 66630efac432c45664098000bdebe9a7

Matriz de estado:

66	c4	64	bd
63	32	09	eb
0e	c4	80	e9
fa	56	00	a7

Fig. 1. Matriz de estado.

O algoritmo opera com um número determinado de rodadas, no qual este número é estabelecido de acordo com o tamanho da chave, que podem ser de 128, 192 ou 256 bits, gerando um total de 10, 12 e 14 rodadas, respectivamente.

A. Operação SubBytes

O estágio de *SubBytes* consiste numa etapa de substituição, onde cada um dos valores armazenados nas 16 posições da matriz 4x4 será substituído por um novo valor com base em uma tabela de substituição, que é chamada de s-box e contêm 256 posições, normalmente expressos em hexadecimal. Esta s-box consiste em uma matriz 16x16 e pode ser visualizada na Fig. 2.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Fig. 2. Matriz 16x16 que representa a S-Box [15].

B. Operação ShiftRows

O estágio de *ShiftRows* consiste em deslocamentos circulares na matriz 4x4, onde cada uma das quatro linhas da matriz se deslocam para a esquerda com um determinado valor. As linhas 0, 1, 2 e 3 da matriz de entrada se deslocam para a esquerda 0, 1, 2 e 3 vezes, respectivamente. A Fig. 3 exemplifica este processo, mostrando a matriz antes e depois da operação de *ShiftRow*.

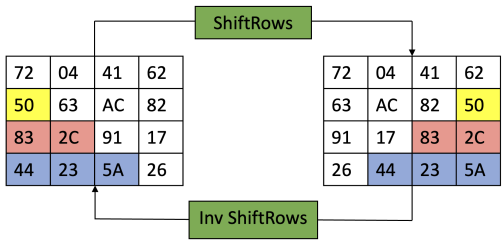

Fig. 3. Exemplo da operação ShiftRow [16].

C. Operação MixColumns

O estágio de *MixColumns* utiliza cada uma das colunas da matriz de estado individualmente, realizando multiplicação de matrizes para gerar os novos resultados. A Fig. 4 exemplifica como a matriz final é gerada, através da multiplicação de matrizes.

$$\begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \begin{vmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{vmatrix} = \begin{vmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{vmatrix}$$

Fig. 4. Operação *MixColumns* [17].

Cada posição da matriz resultante é a soma dos produtos dos elementos de uma linha e uma coluna. Estas somas e multiplicações utilizam o conceito de Campos de Galois, especificamente em GF(2^8).

D. Operação AddRoundKey

O estágio de *AddRoundKey* é uma operação XOR entre cada posição da matriz de estado com a chave da rodada. A Fig.

5 demonstra uma operação de *AddRoundKey*, onde vemos a matriz de entrada à esquerda realizando uma operação XOR com matriz que contém chave secreta correspondente a rodada do algoritmo, resultando em uma nova matriz de estado.

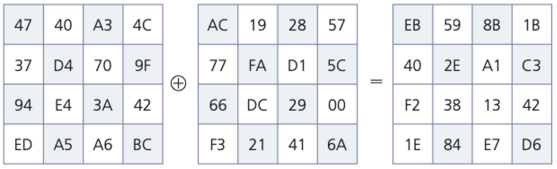

Fig. 5. Exemplo de operação *AddRoundKey* [17].

O algoritmo AES-128, originalmente, é composto por 10 rodadas, onde da 1ª a 9ª rodada são executadas as 4 operações acima descritas exatamente nesta ordem e, por último, na 10ª rodada, são executadas somente 3 destas operações, sendo a operação de *MixColumns* não utilizada nesta rodada.

III. Otimizações Propostas

As otimizações propostas neste artigo estão concentradas nas operações de *SubBytes* e *MixColumns*. A modificação proposta no estágio de *SubBytes* acarreta numa redução em consumo de memória. Por outro lado, a modificação implementada no estágio de *MixColumns* traz uma redução em tempo de execução, logo, a combinação destas duas otimizações trazem ótimos resultados para o algoritmo em duas vertentes.

A. Otimização no estágio SubBytes

A otimização proposta para o estágio de *SubBytes* se baseia na utilização de uma s-box menor, contendo apenas 16 entradas. Esta s-box é modelada como um vetor com índice de 0 a 15 e seu conteúdo indo de F a 0. A Fig. 6 exemplifica a ideia da tabela desenvolvida.

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valor	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Fig. 6. S-Box desenvolvida com 16 bytes.

A matriz de estado é composta por 16 entradas, dispostas entre as 4 linhas e 4 colunas da matriz. Cada byte, representado em hexadecimal, é divido em dois dígitos onde estes dois dígitos são utilizados como índice da s-box proposta. Quando há um dígito de A a F, em hexadecimal, este é utilizado como decimal, variando de 10 a 15, respectivamente. A Fig. 7 exemplifica o processo de aplicação da operação *SubBytes*, utilizando a nova s-box, mostrando a matriz de estado e, posteriormente, a matriz atualizada pós modificação.

Utilizando este método, não há necessidade de uma s-box para encriptação e uma s-box para decriptação (s-box reversa), pois utilizamos a mesma tabela para ambas operações. Utilizando o hexadecimal 3B como exemplo, ao passar pelo estágio de *SubBytes*, ele se tornaria C4 e, ao passar pelo processo de decriptação, utilizando a mesma s-box, ele voltaria ao seu valor 3B original.

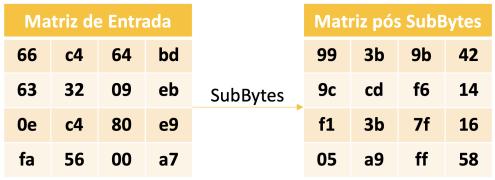

Fig. 7. Exemplo do estágio *SubBytes* utilizando a nova s-box.

O algoritmo original do AES armazena duas s-box de 256 bytes cada, uma para o processo de encriptação e outra para o processo de decriptação, totalizando 512 bytes de consumo em armazenamento. Já a alternativa proposta utiliza apenas 16 bytes, pois utiliza uma única tabela tanto para encriptar quanto para decriptar a informação do bloco, resultando numa redução de 96,87% em bytes.

B. Otimização no estágio MixColumns

A otimização sugerida para o estágio de *MixColumns* se baseia na utilização de operações mais simples e menos custosas em termos de recursos computacionais. Para isto, foi desenvolvida uma função que realiza multiplicação em campos de Galois pela constante 2, exemplificado pelo Algoritmo 1, que serve de base para todas as outras multiplicações, ou seja, todas as outras multiplicações são feitas utilizando esta função e utilizando funções xor para somas. As multiplicações são sempre pelas constantes 2 e 3, no processo de encriptação, e pelas constantes 9, 11, 13 e 14 no processo de decriptação.

```
unsigned char gmult2(unsigned char value){
    if(value & 0x80)
        return (value << 1) ^ 0x1B;
    else
        return value << 1;
}
```

Algoritmo 1. Função que realiza multiplicação por 2

O Algoritmo 2 apresenta a operação *MixColumns* proposta, onde originalmente era um laço de repetição agora são expandidas as operações de soma e multiplicação, que vão compondo as novas colunas da matriz de estado.

```
void MixColumnsLite(unsigned char *state){
    unsigned char tmp[4][4];
    tmp[0][0] = (unsigned char)(gmul2(state
        [0]) ^ gmul3(state[4]) ^ state[8] ^
        state[12]);
    tmp[1][0] = (unsigned char)(state[0] ^
        gmul2(state[4]) ^ gmul3(state[8]) ^
        state[12]);
    tmp[2][0] = (unsigned char)(state[0] ^
        state[4] ^ gmul2(state[8]) ^ gmul3(
        state[12]));
    tmp[3][0] = (unsigned char)(gmul3(state
        [0]) ^ state[4] ^ state[8] ^ gmul2(
        state[12]));
    tmp[0][1] = (unsigned char)(gmul2(state
        [1]) ^ gmul3(state[5]) ^ state[9] ^
        state[13]);
    tmp[1][1] = (unsigned char)(state[1] ^
        gmul2(state[5]) ^ gmul3(state[9]) ^
        state[13]);
```

```
    tmp[2][1] = (unsigned char)(state[1] ^
        state[5] ^ gmul2(state[9]) ^ gmul3(
        state[13]));
    tmp[3][1] = (unsigned char)(gmul3(state
        [1]) ^ state[5] ^ state[9] ^ gmul2(
        state[13]));
    tmp[0][2] = (unsigned char)(gmul2(state
        [2]) ^ gmul3(state[6]) ^ state[10] ^
        state[14]);
    tmp[1][2] = (unsigned char)(state[2] ^
        gmul2(state[6]) ^ gmul3(state[10]) ^
        state[14]);
    tmp[2][2] = (unsigned char)(state[2] ^
        state[6] ^ gmul2(state[10]) ^ gmul3(
        state[14]));
    tmp[3][2] = (unsigned char)(gmul3(state
        [2]) ^ state[6] ^ state[10] ^ gmul2(
        state[14]));
    tmp[0][3] = (unsigned char)(gmul2(state
        [3]) ^ gmul3(state[7]) ^ state[11] ^
        state[15]);
    tmp[1][3] = (unsigned char)(state[3] ^
        gmul2(state[7]) ^ gmul3(state[11]) ^
        state[15]);
    tmp[2][3] = (unsigned char)(state[3] ^
        state[7] ^ gmul2(state[11]) ^ gmul3(
        state[15]));
    tmp[3][3] = (unsigned char)(gmul3(state
        [3]) ^ state[7] ^ state[11] ^ gmul2(
        state[15]));
}
```

Algoritmo 2. Função referente ao estágio *MixColumns*

Esta otimização proposta para o estágio de *MixColumns* apresentou bons ganhos em termos de tempo de execução, tornando o algoritmo AES bem mais rápido comparado a sua versão original.

IV. Resultados

As otimizações implementadas foram avaliadas em termos de desempenho, utilizando como métricas o tempo de encriptação e o consumo de memória, e em termos de segurança, avaliando efeito avalanche e critério de balanceamento, além de serem submetidas aos testes do NIST. Todas avaliações foram conduzidas em um MacBook Air com processador Apple M2, 8GB de memória RAM, sistema operacional macOS Ventura 13.4.1 e SSD de 256GB, além do auxílio da IDE do Arduino na versão 2.1.0 e compilador GCC, executado diretamente via terminal.

A. Análise de Desempenho

1) Tempo de Encriptação: Para obtenção dos valores dos tempos de execução, cada teste foi executado 1000 vezes e foi extraída a média aritmética destes valores. Como entrada do algoritmo foram utilizadas strings de tamanho 10, 25, 70, 100, 1000, 2000 e 10000 bytes. A Tabela I traz o comparativo dos tempos de execução entre o algoritmo AES original e o algoritmo proposto.

Pode-se observar uma redução significativa em termos de tempo de encriptação, atingindo uma redução de aproximadamente 90% com o uso da otimização proposta.

TABELA I
Tempos de encriptação do AES original e Modificado

Entrada(B)	AES Original(µs)	AES Modif.(µs)	Redução(%)
10	40,96	8,23	79,92
25	74,60	12,92	82,68
70	172,95	21,96	87,30
100	243,57	29,25	87,99
1000	2130,79	220,75	89,64
2000	4222,14	434,43	89,71
10000	21091,54	2167,31	89,72

2) Consumo de Memória: A Figura 8 mostra o consumo de memória, em bytes, das duas versões do algoritmo, tanto a memória dinâmica quanto a memória de programas.

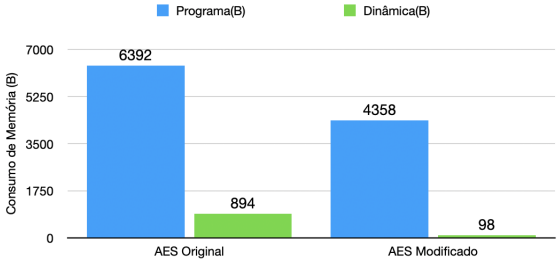

Fig. 8. Consumo de memória do AES original e Modificado.

Para extração destes resultados, foram compiladas ambas versões dos algoritmos na IDE do Arduino, setando o Arduino Uno como processador alvo. Em questão da memória de programas foi possível obter uma redução de 31,82%. Grande parte desta redução se deve ao fato da versão otimizada de *MixColumns* ser mais simples, utilizando um código bem menor para realizar as operações necessárias, também contribuindo para uma diminuição no tempo de execução. Já em memória dinâmica foi possível obter uma redução de 89,04%, o que é um número bem expressivo, e se deve ao fato da modificação implementada no estágio *SubBytes*, onde foi reduzido de duas s-box que totalizavam 512 bytes para uma única s-box que ocupa apenas 16 bytes.

B. Análise de Segurança

1) Efeito Avalanche: A análise de efeito avalanche é primordial para avaliar a segurança de um algoritmo criptográfico. Se houver uma pequena mudança no *plaintext* ou na chave, mesma que de apenas 1 bit, isto deve causar uma mudança significativa no *ciphertext* [8]. Nos testes realizados neste artigo, as mudanças ocorreram na chave, utilizando o conceito de distância de Hamming (HD) entre duas strings, cuja distância corresponde ao número de caracteres diferentes entre elas. O cálculo efetivo do efeito avalanche se deu de acordo com a equação apresentada na Eq. 1.

$$\text{Efeito Avalanche} = \left(\frac{\text{nº de bits alterados no ciphertext}}{\text{nº de bits no ciphertext}}\right) * 100 \quad (1)$$

A análise utiliza HDs variando de 1 a 5, e em seguida, é realizada uma média aritmética destes 5 resultados. O algoritmo AES original e modificado apresentaram um efeito avalanche de 50,55% e 50,41%, respectivamente, o que está de acordo com o valor esperado para este teste, onde devem-se atingir valores próximo de 50%.

2) Critério de Balanceamento: Um dos principais critérios para testar se uma s-box está bem planejada é o cálculo de balanceamento. Uma s-box deve distribuir de forma balanceada a quantidade de 0's e 1's no *ciphertext*. A Figura 9 apresenta a distribuição de 0's e 1's de ambas versões do AES, quando testados com uma entrada de 1000 bytes.

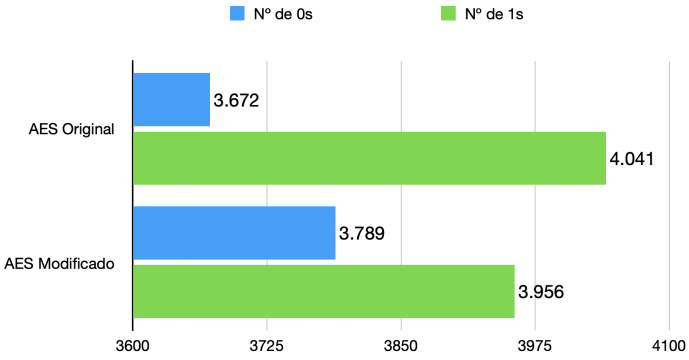

Fig. 9. Critério de Balanceamento.

Como se pode observar na Figura 9, o algoritmo modificado apresentou uma melhor distribuição na quantidade de 0's e 1's.

3) NIST Randomness Test: O teste do NIST é utilizado para verificar a aleatoriedade do *ciphertext* gerado por um algoritmo criptográfico [18]. A saída destes testes para cada sequência binária é sempre aleatória ou não aleatória, o que significa na prática aprovado ou reprovado, respectivamente. Foram geradas 6 strings de forma aleatória, cada uma com 1000 bytes, e logo foram analisadas as sequências binárias geradas pelo algoritmo proposto. Ao executar o teste nas 6 sequências, foi possível verificar que todas foram aprovadas em cada um dos testes disponíveis.

V. Comparação dos Resultados

Na literatura existem soluções propostas com o objetivo de tornar o algoritmo AES mais leve. A busca pelas otimizações já desenvolvidas foi feita através de um mapeamento sistemático da literatura, nas bibliotecas IEEE Xplore e ACM Digital Library, onde inicialmente foi obtido um total de 3.664 artigos e, passando por 5 etapas de filtragem, foram selecionados 5 artigos que tinham grande afinidade ao tema deste artigo e possibilitavam comparação com este trabalho. Esta seção apresenta os principais trabalhos encontrados visando estabelecer uma comparação das estratégias de otimização. Em [8] é proposto a substituição do estágio *MixColumns* por uma função que utiliza o conceito matemático de frações contínuas, onde de mesmo modo gera confusão no *ciphertext*. Com isto, [8] conseguiu reduzir o tempo de execução do algoritmo, ainda garantindo segurança para a informação. É possível avaliar as proposta com uma entrada de tamanho 20 bytes, onde em [8] foi atingida uma redução de 41,38% comparado ao AES original, enquanto a redução atingida pela modificação proposta por este artigo foi de 81,63%. Ambas otimizações apresentaram bom resultado de efeito avalanche, porém, a otimização desenvolvida por [8] não foi submetida aos testes de aleatoriedade do NIST nem foi realizada avaliação de critério de balanceamento.

A modificação proposta por [9] consiste na utilização de sistemas caóticos em todos os estágios do AES. Para isto, foram utilizadas combinações 1D, 2D e 3D do mapa caótico do tipo *logistic*. Foi obtida redução no tempo de execução além de garantir segurança da aplicação. Há como avaliar o tempo de encriptação para entradas de tamanho 10000 bytes, onde eles conseguiram uma redução de 91,11% comparado ao AES original, enquanto a modificação proposta por este artigo atingiu uma redução de 89,72%. Apesar de ter uma redução ligeiramente menor, a modificação proposta por este artigo consome uma quantidade de memória bem menor, visto que a proposta de [9] utiliza 2 s-boxes de 256 bytes cada, enquanto a proposta deste artigo utiliza apenas 16 bytes na s-box. Ambas otimizações avaliaram critério de balanceamento, efeito avalanche e NIST.

Modificações em dois estágios do AES foram propostas por [10], no qual consistem em utilizar uma s-box de menor tamanho além de utilizar um sistema caótico *logistic* 1D no estágio de *ShiftRow*. Foram obtidos bons níveis de segurança com as modificações propostas, além de serem obtidas reduções de tempo de execução e consumo de memória. É possível comparar os tempos de execução, para entradas de tamanho 20 e 200 bytes, onde foi por eles obtido uma redução de 13,6% e 4,23%, respectivamente, comparados ao AES original, enquanto a mudança deste artigo atingiu uma redução de 81,63% e 88,7% para entradas de mesmo tamanho. Em questão de consumo de memória, [10] utiliza 64 bytes em suas s-box, enquanto a modificação proposta utiliza uma única s-box de apenas 16 bytes. Ambas modificações foram avaliadas no teste do NIST e também em efeito avalanche, porém, [10] não realizou avaliação de critério de balanceamento.

Em [11] é proposta uma versão do algoritmo AES com menor número de rodadas e que utiliza tabelas pré-calculadas para acelerar o tempo de execução do estágio *MixColumns*. Com isto, foi possível obter uma execução mais rápida do algoritmo. Em questões de tempo de encriptação, é possível comparar as proposta com a entrada de tamanho 1000 bytes, onde foi encontrado por [11] uma redução de 35% em comparação ao AES original, enquanto a proposta deste artigo atingiu uma redução de 88,93%. Ambos artigos avaliaram efeito avalanche, tendo resultados aceitáveis, porém, a proposta de [11] não foi submetida ao teste do NIST nem realizou avaliação em termos de consumo de memória.

Combinando os sistemas caóticos *logistic* e *Lorenz*, [12] propõe a utilização de um sistema 5D em todos os estágios do algoritmo AES. Foi verificado que o algoritmo proposto é robusto em termos de segurança, além de obter um melhor tempo de execução. Pode-se comparar o tempo de execução com

entradas de tamanho 1000 e 10000 bytes, onde a otimização proposta por [12] atingiu uma redução de 30,19% e 17,7%, respectivamente, comparado ao AES original, enquanto a proposta deste artigo atingiu uma redução de 88,93% e 89,72%, respectivamente. [12] validou sua otimização com os testes do NIST, porém, não realizou testes de efeito avalanche nem observou questões de consumo de memória. A Figura 10 apresenta um resumo das comparações realizadas entre os artigos.

Artigo	Tamanho da Entrada(B) Redução em %				Efeito Aval.	Critério de Balanc.	NIST
	20	200	1000	10000			
8	41.38%	-	-	-	✅	❌	❌
9	-	-	-	91.11%	✅	✅	✅
10	13.6%	4.23%	-	-	✅	❌	✅
11	-	-	35%	-	✅	❌	❌
12	-	-	30.19%	17.7%	❌	❌	✅
Este Artigo	81.63%	88.7%	88.93%	89.72%	✅	✅	✅

Fig. 10. Resumo das Comparações Entre os Artigos.

VI. Conclusões

Este artigo propôs otimizações em dois estágios do algoritmo AES, *SubBytes* e *MixColumns*, onde se buscou redução de consumo de memória e redução de tempo de execução, respectivamente. A modificação realizada em *SubBytes* consistiu em utilizar uma s-box menor, saindo de duas s-boxes de 256 bytes cada do algoritmo original, para apenas uma s-box de 16 bytes. Já no estágio *MixColumns*, a modificação se concentrou em utilizar operações de adição e multiplicação mais leves, fazendo com que o algoritmo tivesse uma execução mais rápida. A média de redução, entre tamanhos de entrada avaliados, quando comparados ao AES original, foi de 86,71%, levando o algoritmo a ter uma tempo de execução bem menor. Já em termos de consumo de memória apresentou bons resultados tornando o algoritmo bem mais leve, fazendo com que seja mais viável sua utilização em dispositivos com recursos computacionais limitados. Por último, o algoritmo proposto foi validado em termos de segurança com o testes de efeito avalanche e de balanceamento, além de ser aprovado no teste de aleatoriedade do NIST. Concluindo, é possível afirmar que este *lightweight* AES é apropriado para utilização em aplicações IoT, pois foram atingidas grandes reduções tanto em consumo de memória quanto em tempo de execução. Como trabalhos futuros, serão avaliadas ambas versões, original e proposta, do algoritmo AES em plataformas apropriadas para desenvolvimento de aplicações IoT, com o intuito de extrair métricas tanto desempenho quanto consumo energético.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Referências

[1] P. Marwedel, Embedded System Design, Dortmund, Germany: Springer, 2021.

[2] S. Sinha. “State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally.” IOT Analytics. https://iot-analytics.com/number-connected-iot-devices/ (accessed Jun. 30, 2023).

[3] N. Mishra and S. Pandya, ”Internet of Things Applications, Security Challenges, Attacks, Intrusion Detection, and Future Visions: A Systematic Review,” in IEEE Access, vol. 9, pp. 59353-59377, 2021.

[4] A. A. Ahmad, ”A New Security Method for the Internet of Things Based on Ciphering and Deciphering Algorithms,” Kirkuk University Journal/Scientific Studies(KUJSS), Vol. 13, No.3, 2018.

[5] K.L. Tsai, Y.L. Huang, F.Y. Leu, I. You, Y.L. Huang, and C.H. Tsai, ”AES-128 Based Secure Low Power Communication for LoRaWAN IoT Environments,” IEEE Access, Vol. 6, 2018.

[6] O. Toshihiko. “Lightweight Cryptography Applicable to Various IoT Devices.” NEC. https://www.nec.com/en/global/techrep/journal/g17/n01/170114.html (accessed Jun. 30, 2023).

[7] S. Ullah, R. Z. Radzi, T. M. Yazdani, A. Alshehri and I. Khan, ”Types of Lightweight Cryptographies in Current Developments for Resource Constrained Machine Type Communication Devices: Challenges and Opportunities,” in IEEE Access, vol. 10, pp. 35589-35604, 2022.

[8] H. M. Mohammad and A. A. Abdullah, ”Enhancement process of AES: a lightweight cryptography algorithm-AES for constrained devices” in TELKOMNIKA (Telecommunication Computing Electronics and Control), vol. 20, pp. 551-560, 2022.

[9] M. S. Fadhil, A. K. Farhan, M. N. Fadhil and N. M. G. Al-Saidi, ”A New Lightweight AES Using a Combination of Chaotic Systems,” 2020 1st. Information Technology To Enhance e-learning and Other Application (IT-ELA, Baghdad, Iraq, 2020, pp. 82-88.

[10] D. N. Hammod, ”Modified Lightweight AES based on Replacement Table and Chaotic System,” 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 2022, pp. 1-5.

[11] L. Shi, Y. Wang, R. Jia, T. Peng, J. Jiang, and S. Zhu, ”Research of Lightweight Encryption Algorithm Based on AES and Chaotic Sequences for Narrow-Band Internet of Things,” In: Machine Learning and Intelligent Communications, vol. 294, pp. 267–280, 2019.

[12] J. R. Naif, G. H. Abdul-Majeed and A. K. Farhan, ”Secure IOT System Based on Chaos-Modified Lightweight AES,” 2019 International Conference on Advanced Science and Engineering (ICOASE), Zakho - Duhok, Iraq, 2019, pp. 1-6.

[13] F. Medeleanu, C.Racuciu, M.Rogobete,”Considerations About The Possibilities To Improve Aes s-box Cryptographic Properties By Multiplication”,Proceedings Of The Romanian Academy, 2015; 16(Special Issue):339-344.

[14] T. Neha. “Advanced Encryption Standard (AES).” Binary Terms. https://binaryterms.com/advanced-encryption-standard-aes.html (acessed Jul. 01, 2023)

[15] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback and J. Dray, Advanced Encryption Standard (AES), Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.FIPS.197 (Accessed July 1, 2023)

[16] S. A. Mehdi, K. K. Jabbar and F. H. Abbood, ”Image Encryption Based On The Novel 5D Hyper-Chaotic System Via Improved AES Algorithm”, In: International Journal of Civil Engineering and Technology (IJCIET), vol. 9, pp. 1841-1855, 2018

[17] W. Stallings, Cryptography and Network Security: Principles and Practice. USA: Prentice Hall Press, 2013.

[18] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and SanVo,”A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Application”, NIST Special Publication 800-22, 2000.