Análise e comparação de algoritmos criptográficos simétricos embarcados na plataforma Arduino

João Victor Guinelli, Octavio Vieira Aguiar, Nilson Mori Lazarin

[1]Bacharelado em Sistemas de Informação – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ) – Nova Friburgo, RJ – Brazil

jvguinelli@gmail.com, octaviowork@hotmail.com, nilson.lazarin@cefet-rj.br

Abstract. *A wide number of devices collect sensitive data through sensors that are integrated to our homes, cars and clothes. As they have limited resources, there is a complexity for the secure storage and transmission of this data using more robust cryptographic algorithms. This paper presents a comparison between Rijndael and Serpent, finalists of the AES contest, embedded in the Arduino platform. Experiments were performed considering performance and resource consumption. According to the results, Rijndael proved to be more suitable when there is demand for less consumption of time and energy, while Serpent is more appropriate when the demand is for less use of storage space and memory.*

Resumo. *Inúmeros dispositivos colhem dados sensíveis através de sensores integrados em nossas casas, carros e roupas. Por possuírem recursos limitados, há uma complexidade para o armazenamento e transmissão segura destes dados utilizando algoritmos criptográficos mais robustos. Este artigo apresenta um comparativo entre o Rijndael e o Serpent, finalistas do concurso AES, embarcados na plataforma Arduino. Foram realizados experimentos considerando desempenho e consumo de recursos. Conforme os resultados obtidos, o Rijndael mostrou-se mais indicado quando há demanda por menor consumo de tempo e energia, enquanto o Serpent é mais apropriado quando a demanda é de menor uso de espaço de armazenamento e de memória.*

1. Introdução

O paradigma da IoT (*Internet of Things*) faz referência à grande presença de dispositivos ao nosso redor que são capazes de interagir uns com os outros para atingirem objetivos em comum de forma colaborativa [Atzori et al. 2010]. Muitas vezes esses dispositivos necessitam armazenar ou trocar informações sensíveis com seus pares, dando assim, origem a diversos problemas de segurança [Roman et al. 2013]. Uma forma de prover a segurança necessária a essas aplicações é utilizar algoritmos criptográficos para cifrar os dados a serem armazenados ou transmitidos [Zhang et al. 2014]. No entanto, é preciso ter em mente que dispositivos embarcados possuem certas limitações e, por isso, a utilização de algoritmos que consomem muitos recursos pode não ser viável [Jing et al. 2014]. Ao mesmo tempo, utilizar algoritmos que possuem bom desempenho, mas que não proveem um bom nível de segurança para as aplicações também não é uma opção interessante.

Este trabalho apresenta um comparativo entre os algoritmos de criptografia simétrica Rijndael [Daemen and Rijmen 1999] e Serpent [Knudsen 1998] quando execu-

tados em Arduino, plataforma possui recursos limitados e é amplamente utilizada no contexto da IoT. O comparativo tem como principal objetivo avaliar em quais circunstâncias é mais indicado utilizar o Serpent e em quais seria melhor utilizar o Rijndael. Na avaliação foram considerados indicadores como: tempo de execução, quantidade de ciclos de *clock*, consumo de memória SRAM e Flash (armazenamento), além do consumo de energia. A escolha do Rijndael e do Serpent se deu pelo fato dos dois algoritmos serem reconhecidamente seguros, tendo ambos chegado à final do concurso realizado pelo *National Institute of Standards and Technology* (NIST) para a escolha de um novo padrão criptográfico a ser utilizado pelo governo americano, o *Advanced Encryption Stantard* (AES). Além disso, no relatório acerca do processo de escolha do AES [Nechvatal et al. 2001], tanto o Rijndael quanto o Serpent recebem destaque entre os finalistas do concurso quando se considera a execução em ambientes com restrições de memória.

Este trabalho está estruturado da seguinte maneira: na seção 2 são apresentados alguns conceitos importantes sobre o tema abordado; na seção 3 é apresentada uma discussão sobre alguns trabalhos relacionados; na seção 4 é apresentada a metodologia utilizada no desenvolvimento do trabalho e descrição sobre os experimentos realizados; na seção 5, por sua vez, estão os resultados obtidos e uma análise sobre os mesmos; por fim, na seção 6 é apresentada a conclusão do trabalho e uma discussão sobre possíveis trabalhos futuros.

2. Fundamentação Teórica

Os critérios de avaliação considerados pelo NIST no concurso AES, em ordem de importância, foram: segurança, custo computacional, e características do algoritmo e de sua implementação [Nechvatal et al. 2001]. Com base nestes critérios foram anunciados cinco algoritmos finalistas: MARS, RC6, Rijndael, Serpent e Twofish. Após levar em consideração todas as análises realizadas e comentários feitos pelo público ao longo de um ano de testes, o algoritmo Rijndael foi escolhido como vencedor do concurso. Segundo o NIST, o vencedor apresentava boa performance de hardware e software em uma grande variedade de plataformas computacionais, além de se apresentar como uma boa escolha para ser executado em ambientes com restrições de espaço [Nechvatal et al. 2001].

2.1. Rijndael

Rijndael é uma cifra de bloco iterada na qual o tamanho do bloco e da chave criptográfica são variáveis, podendo ser independentemente especificadas para 128, 192 ou 256 bits. As operações internas do algoritmo são realizadas sobre uma matriz de bytes chamada de *estado* que possui sempre quatro linhas. Já o número de colunas dessa matriz é variável, sendo obtido ao dividir o tamanho do bloco utilizado por 32 bits [Daemen and Rijmen 1999]. A sequência de operações realizadas sobre o *estado* no processo de cifrar e decifrar é apresentada na Figura 1(a). O número de rodadas (Nr) a serem executadas pelo algoritmo é dado como uma função do tamanho do bloco e do tamanho da chave, conforme apresentado na Tabela 1. Abaixo estão descritas as etapas do Rijndael:

ByteSub: esta operação consiste em substituir cada byte do *estado* por um byte existente numa *S-box* (tabela de substituição) composta por 16 linhas e 16 colunas.

ShiftRow: esta operação consiste em rotacionar os bytes presentes nas linhas (C0, C1, C2 e C3) do *estado*, em função do tamanho do bloco. Sendo que: C0 não rotaciona

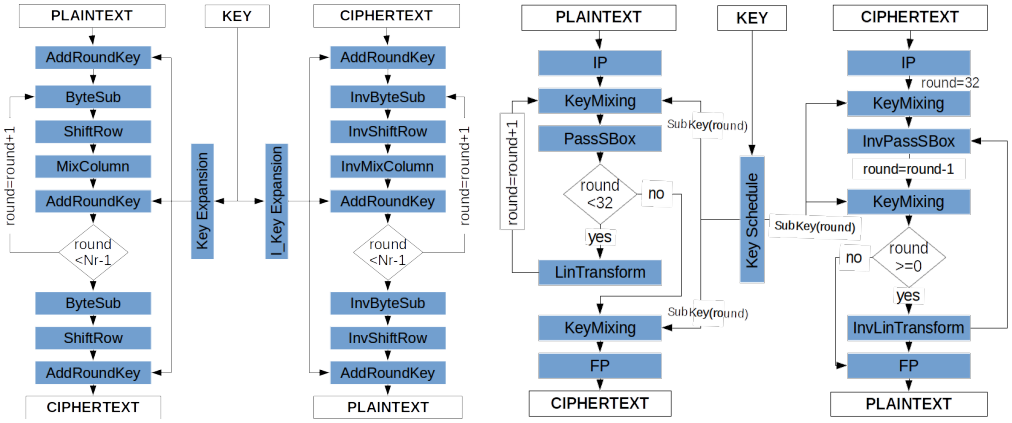
Tabela 1. Número de rodadas (Nr) em função de chave e bloco [Daemen and Rijmen 1999].

Tamanho da chave	Tamanho do bloco		
	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

em nenhum caso; C1 rotaciona uma coluna à esquerda em todos os casos; C2 rotaciona duas colunas à esquerda para blocos de 128 e 192 bits e três colunas à esquerda para blocos de 256 bits; C3 rotaciona três colunas à esquerda para blocos de 128 e 192 bits e rotaciona quatro colunas à esquerda para blocos de 256 bits.

MixColumns: esta operação trata cada coluna do *estado* como um polinômio em $GF(8^2)$, que é então multiplicado em módulo $x^4 + 1$ pelo polinômio sempre fixo $c(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02$.

AddRoundKey: nesta operação, cada byte do *estado* é combinado com o byte correspondente da sub-chave da rodada através da operação Ou-Exclusivo. A sub-chave de cada rodada é sempre uma matriz do mesmo tamanho do *estado*, essas sub-chaves são geradas pelo escalonador de chaves do algoritmo a partir da chave original. Ao todo são geradas uma sub-chave para cada rodada do algoritmo e uma sub-chave extra que é utilizada na rodada inicial, onde apenas a operação *AddRoundKey* é executada.



(a) Processo de cifrar e decifrar do Rijndael (b) Processo de cifrar e decifrar do Serpent

Figura 1. Estrutura dos algoritmos Rijndael e Serpent

2.2. Serpent

Serpent [Knudsen 1998], segundo colocado no concurso AES, é uma cifra de bloco de tamanho fixo (128 bits) e de tamanho da chave variável. Internamente, a cifra completa a chave inserida, através da adição de um bit "1" seguido por uma sequência de bits "0" até que atinja 256 bits necessários para o processo criptográfico seja obtido. No Serpent o texto em claro é tratado como 4 palavras (X_0, X_1, X_2, X_3) de 32 bits sendo submetido a uma rede de substituição-permutação de 32 rodadas. Para cada uma dessas rodadas, o escalonador de chaves do algoritmo gera uma sub-chave de 128 bits, derivada da chave informada pelo usuário. A única exceção é a última iteração, para a qual duas sub-chaves

são geradas. Na Figura 1(b) é possível observar o funcionamento da cifra. Cabe ressaltar que todas as rodadas do Serpent possuem o mesmo conjunto de operações, com exceção da última, onde a operação *Linear Transformation* é substituída por *Key Mixing*, o que explica a necessidade da utilização de uma sub-chave extra durante a última iteração. Abaixo são descritas as etapas do Serpent:

Initial Permutation (IP): utiliza um vetor de 128 posições. O valor de cada elemento do vetor indica a posição do bit no bloco submetido a IP e a posição deste elemento no vetor indica a nova posição do bit no bloco de saída.

Key Mixing: a sub-chave da rodada é combinada com a saída da rodada anterior através da operação Ou-Exclusivo.

Pass Through S-Box: o Serpent utiliza um conjunto de 8 *S-Boxes*, sendo cada uma delas utilizada em uma rodada. A cada rodada, 32 cópias da *S-Box* são aplicadas paralelamente a conjuntos de quatro *bits* vindos da operação anterior de *Key Mixing*.

Linear Transformation: nesta operação, cada uma das 4 palavras (X_0, X_1, X_2, X_3) de 32 bits geradas na operação anterior passam pela seguinte transformação:

$$\begin{aligned}
X_0; X_1; X_2; X_3 &:= S_i(B_i \oplus K_i) \\
X_0 &:= X_0 <<< 13 \\
X_2 &:= X_2 <<< 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\
X_1 &:= X_1 <<< 1 \\
X_3 &:= X_3 <<< 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\
X_0 &:= X_0 <<< 5 \\
X_2 &:= X2 <<< 22 \\
B_{i+1} &:= X_0; X_1; X_2; X_3
\end{aligned}$$

Onde:

- $<<<$ é uma operação de rotação à esquerda realizada sobre os bits de X_n.
- $<<$ é uma operação de mudança (*shift*) similar à de rotação, onde os bits mais significativos são excluídos durante a rotação e zeros são inseridos nos bits menos significativos.

Final Permutation (FP): esta operação é a operação inversa da Permutação Inicial (IP).

3. Trabalhos Relacionados

Um dos questionamentos levantados por [Johannesen 2014] é se algum *kit* de desenvolvimento de microcontrolador, como o Arduino, seria poderoso o suficiente para atuar como um módulo criptográfico. Para responder a esta pergunta, o autor procurou testar a performance do AES quando executado em uma placa Arduino Due. Os resultados encontrados levaram a conclusão de que o sistema é capaz de prover segurança suficiente para o problema em questão, mas que, no entanto, as limitações de hardware do Arduino tornam o sistema proposto inviável para encriptação de grande quantidade de dados.

O estudo realizado por [Frt'ala and Hromada 2014], por sua vez, compara 6 diferentes algoritmos de chave simétrica com o objetivo de determinar qual seria o melhor

deles para uso em um microprocessador AVR de 8 bits. Neste estudo é utilizado uma placa Arduino Nano e são testados 6 algoritmos, sendo 3 cifras de bloco: AES, DES e Present; e 3 cifras de fluxo: Grain, Trivium e Mickey. Os critérios levados em consideração para a escolha do melhor deles foram: velocidade de encriptação, tempo de inicialização da cifra, volume de texto cifrado transmitido com *tag* e consumo de SRAM. Os resultados obtidos mostraram que o AES era o algoritmo mais rápido, o que surpreendeu os autores, já que cifras de fluxo, são geralmente mais rápidas que cifras de bloco. Os autores acreditam que isto pode ter ocorrido por conta da biblioteca utilizada nos testes, a AVR Crypto Library[1], que talvez possuísse uma melhor implementação do AES, já que ele é o mais difundido dos algoritmos testados.

Já em [Hyncica et al. 2011] são testados quinze algoritmos de chave simétrica em três diferentes microcontroladores, dentre eles um Atmel ATmega de 8 bits. Todos os testes foram realizados utilizando o modo ECB, pelo fato dele não trazer nenhum *overhead* adicional ao algoritmo. Foram considerados o tempo para cifrar, o *throughput*, e os requisitos de memória do algoritmo. Os resultados dos testes mostraram que os algoritmos AES, Twofish e SAFER possuíam o melhor *throughput* de encriptação. Os autores concluem, ainda, que pelo fato da biblioteca de algoritmos utilizada ser de propósito geral e destinada a plataformas de 32 bits, ela não oferece o mesmo desempenho que é entregue por bibliotecas especializadas, mas, mesmo assim, é possível utilizá-la em plataformas de 16 ou 8 *bits*.

Para a realização deste trabalho havia a possibilidade de se utilizar alguma biblioteca de algoritmos criptográficos criada especificamente para microcontroladores AVR de 8 bits, que são os utilizados em placas Arduino, tal como a bibliotecas AVR Crypto Library, que foi usada por [Frt'ala and Hromada 2014] em seu estudo. Como foi destacado por [Hyncica et al. 2011], devido ao fato de serem desenvolvidas especificamente para uma determinada plataforma, essas bibliotecas tendem a obter um desempenho superior ao de bibliotecas de propósito geral para plataformas de 32 bits.

Por esse motivo, optamos pela implementação original do Rijndael[2] e do Serpent[3] fornecidas pelos próprios autores à época da realização do concurso do AES, uma vez que essas implementações, embora não ofereçam o melhor desempenho possível para a plataforma onde os testes foram realizados, apresentam como ponto favorável o fato de que nenhum dos algoritmos será prejudicado na análise por conta do uso de implementação otimizada, visto que não encontrou-se na literatura bibliotecas otimizadas do Serpent para o Arduino. Além disso, optou-se pela utilização do modo ECB por não adicionar nenhum *overhead* ao algoritmo, conforme observado por [Hyncica et al. 2011].

4. Metodologia

Nos experimentos foi utilizada a placa Arduino Mega 2560 (baseada no microcontrolador ATmega2560) conectada a um Módulo Cartão SD e alimentada via USB, através de um medidor USB de consumo de energia. Seu poder computacional é de 16MHz de Clock; 8KBytes de SRAM; 256KBytes de espaço de armazenamento. A escolha do Arduino Mega deu-se pela insuficiência de memória SRAM na placa Arduino Uno (2KBytes)

[1]https://wiki.das-labor.org/w/AVR-Crypto-Lib/en

[2]http://www.efgh.com/software/rijndael.htm

[3]http://www.cl.cam.ac.uk/ rja14/serpent.html

para suportar os algoritmos originais Rijndael e Serpent propostos por seus autores ao concurso AES. A placa Arduino Uno teve sua capacidade excedida em 88% para carregar o algoritmo Serpent e em 165% para carregar o Rijndael.

A avaliação de desempenho entre os algoritmos considerou as seguintes métricas: tempo de processamento, ciclos de clock, uso de memória SRAM, uso de memória de armazenamento, e consumo de energia. Buscou-se comparar os algoritmos e identificar em quais tipos de aplicação, levando em consideração requisitos e limitações, cada algoritmo é mais indicado. Os experimentos consistiram na utilização dos algoritmos sobre diferentes massas de dados em diferentes locais de armazenamento.

A primeira fase dos experimentos submeteu aos algoritmos conjuntos de dados carregados em memória SRAM e a segunda fase submeteu aos algoritmos conjuntos de dados armazenados em um SD Card. Em ambas as fases, cada um dos indicadores foi analisado tanto para cifrar, quanto para decifrar. Abaixo segue a descrição dos experimentos realizados:

Armazenamento: avalia o consumo de espaço de armazenamento, leva em consideração as informações retornadas pelo compilador avr-gcc ao término da compilação para a placa.

Ciclos: avalia os ciclos de *clock* necessários, foi utilizado o contador de *clocks* que se inicia quando a placa é ativada. A medição foi feita calculando a diferença entre a leitura feita no início e no final da execução do algoritmo.

Energia: avalia o consumo de energia através de um dispositivo externo que mensura corrente (amperes), tensão (volts), tempo (minutos) e consumo durante o tempo decorrido (mAh).

Memória: avalia o consumo de memória SRAM considerando-se a soma da memória consumida na *stack* e a *heap* durante a execução ininterrupta do algoritmo em questão.

Tempo: avalia o tempo gasto através da diferença entre o *timestamp* registrado no fim da execução do algoritmo e o *timestamp* registrado no início da execução.

5. Resultados

No que se refere aos resultados dos testes para analisar a quantidade de ciclos de clock necessários para a execução dos algoritmos, observou-se que a quantidade de ciclos gastos pelo Rijndael para criptografar foi no mínimo 103 vezes menor em SRAM e 5701 vezes menor em SD que o Serpent. Já para decriptografar ele foi 92 vezes menor em SRAM e 11395 vezes menor em SD. Esse resultado nos permite inferir que o Rijndael se mostra mais viável para aplicações que demandam um menor consumo de processamento. A Tabela 2 apresenta os resultados obtidos neste experimento.

No experimento de tempo de processamento o Rijndael gastou um tempo pelo menos 95 vezes menor em SRAM e 36329 vezes menor em SD que o Serpent para criptografar. Enquanto para decriptografar ele levou um tempo 82 vezes menor em SRAM e 34115 vezes menor em SD. O que demonstra que o Rijndael é mais apropriado para projetos que necessitem de maior performance e desempenho. A Tabela 3 apresenta os resultados das aferições dos tempos de execução dos algoritmos em cada cenário proposto.

Tabela 2. Ciclos de Clock gastos para processamento dos dados

MANIPULAÇÃO DE DADOS EM SRAM					
Massa de dados	**16Bytes**	**32Bytes**	**64Bytes**	**128Bytes**	**256Bytes**
RIJNDAEL - Cifrar	57.984	65.920	81.728	113.472	176.448
RIJNDAEL - Decifrar	65.472	73.216	89.472	121.472	184.512
SERPENT - Cifrar	18.197.760	18.191.360	18.193.280	18.197.184	18.204.992
SERPENT - Decifrar	17.091.328	17.084.608	17.086.656	17.090.624	17.098.560

MANIPULAÇÃO DE DADOS EM SD		
Massa de dados	**1024Bytes**	**2048Bytes**
RIJNDAEL - Cifrar	189.760	191.488
RIJNDAEL - Decifrar	190.272	191.616
SERPENT - Cifrar	1.162.666.368	1.091.857.664
SERPENT - Decifrar	2.325.140.288	2.183.513.536

Tabela 3. Milisegundos gastos para o processamento dos dados

MANIPULAÇÃO DE DADOS EM SRAM					
Massa de dados	**16Bytes**	**32Bytes**	**64Bytes**	**128Bytes**	**256Bytes**
RIJNDAEL - Cifrar	4	5	6	8	12
RIJNDAEL - Decifrar	5	6	7	8	13
SERPENT - Cifrar	1.138	1.136	1.137	1.138	1.139
SERPENT - Decifrar	1.068	1.068	1.068	1.068	1.069

MANIPULAÇÃO DE DADOS EM SD				
Massa de dados	**1024Bytes**	**2048Bytes**	**4096Bytes**	**8192Bytes**
RIJNDAEL - Cifrar	12	13	13	16
RIJNDAEL - Decifrar	12	13	13	16
SERPENT - Cifrar	72.667	145.320	290.638	581.267
SERPENT - Decifrar	68.241	136.470	272.933	545.855

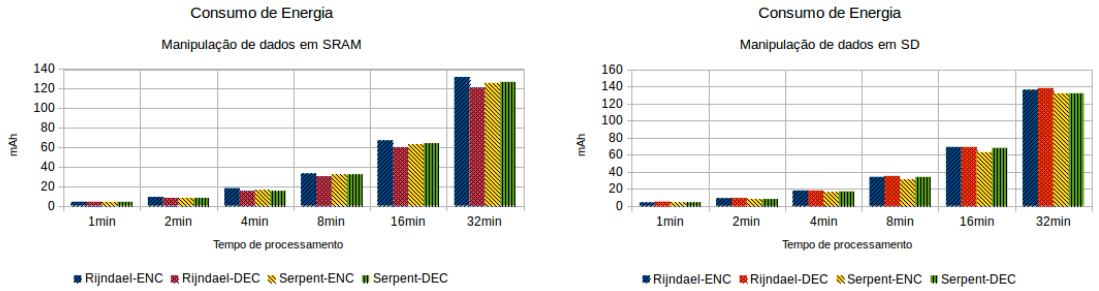
No que diz respeito aos ensaios para verificação do consumo de memória SRAM pelos algoritmos durante sua execução, foi observado que o Rijndael chega a consumir até 93,16% da memória diponível, enquanto no pior dos casos o Serpent consome 61,42% quando os dados são alocados em SRAM. Já quando os dados estão em SD, enquanto o Serpent consome 57,73% da memória no pior dos casos, o Rijndael consome até 89,44%. Esse fato torna o Serpent mais apropriado em projetos que trabalhem com quantidade limitada de memória. Na Tabela 4 são apresentados os resultados do experimento.

Tabela 4. Consumo de bytes da memória SRAM para processar os dados

MANIPULAÇÃO DE DADOS EM SRAM						
Massa de dados	**16Bytes**	**32Bytes**	**64Bytes**	**128Bytes**	**256Bytes**	**512Bytes**
RIJNDAEL - Cifrar	5.616	5.648	5.712	5.840	6.096	6.608
RIJNDAEL - Decifrar	6.640	6.672	6.736	6.864	7.120	7.632
SERPENT - Cifrar	4.040	4.072	4.136	4.264	4.520	5.032
SERPENT - Decifrar	4.040	4.072	4.136	4.264	4.520	5.032

MANIPULAÇÃO DE DADOS EM SD						
Massa de dados	**1KBytes**	**2KBytes**	**4KBytes**	**8KBytes**	**16KBytes**	**32KBytes**
RIJNDAEL - Cifrar	6.303	6.303	6.303	6.303	6.303	6.303
RIJNDAEL - Decifrar	7.327	7.327	7.327	7.327	7.327	7.327
SERPENT - Cifrar	4.727	4.727	4.727	4.727	4.729	4.729
SERPENT - Decifrar	4.727	4.727	4.727	4.727	4.729	4.729

Analisando-se os resultados dos testes do consumo energético da placa Arduino Mega 2560 durante a execução dos algoritmos, verificou-se que o Serpent apresentou



(a) Consumo de energia durante manipulação de dados em SRAM.

(b) Consumo de energia durante manipulação de dados em SD.

Figura 2. Consumo de Energia

um consumo de energia inferior ao Rijndael no mesmo período de tempo. A Figura 2 apresenta os resultados das observações nos intervalos propostos.

No entanto, ao considerarmos a quantidade de blocos processados pelos algoritmos no mesmo período de tempo, podemos chegar a uma conclusão diferente da que considera apenas o tempo de execução. Para obtermos o valor efetivo do gasto energético de cada algoritmo foram utilizadas as Equações 1 e 2:

$$bloco_{mAh} = \frac{\frac{1000}{bloco_{ms}} \bullet 60}{mAh_{min}} \qquad (1)$$

$$mAh_{bloco} = \frac{mAh_{min}}{\frac{1000}{bloco_{ms}} \bullet 60} \qquad (2)$$

Sendo:

$bloco_{mAh}$: quantidade de blocos (128 bits) processados por mAh;
mAh_{bloco} : quantidade de mAh consumidos pela operação sobre 1 bloco;
$bloco_{ms}$: quantidade de milissegundos que o algoritmo leva para operar sobre 1 bloco;
mAh_{min} : quantidade de mAh consumidos pelo algoritmo em 1min de execução.

Para aplicar a equação, utilizou-se os dados obtidos nos experimentos de tempo associados as observações de consumo energético. A Tabela 5 apresenta os resultados da aplicação da fórmula sobre as informações referente a um bloco. Com esses dados é possível verificar que o Rijndael consegue realizar 288 vezes mais operações por *mAh* que o Serpent durante a criptografia, e 214 vezes mais durante a decriptografia. Com isso, pode-se concluir que o Rijndael é o mais adequado quando se possui limitações de energia ou se deseja ter um baixo consumo energético.

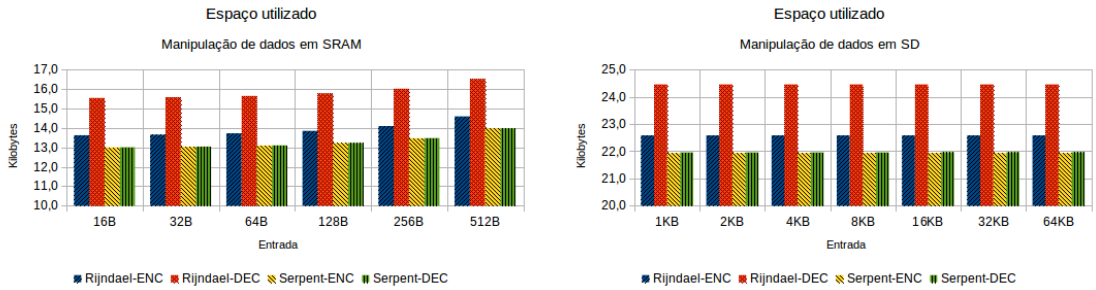
Tabela 5. Consumo de energia em relação a quantidade de blocos operados.

Algoritmo	Operação	mAh por bloco	blocos por mAh
Rijndael	Cifrar	0,0002666667	3750
	Decifrar	0,0003333333	3000
Serpent	Cifrar	0,0758666667	13
	Decifrar	0,0712000000	14

Com relação aos experimentos para medir o consumo de memória de armazenamento Flash para embarcar os algoritmos, o Serpent mostrou-se mais otimizado, com o

algoritmo de criptografia sendo aproximadamente 5% menor que o Rijndael e o de decriptografia em torno de 16% menor. Isso confere ao Serpent maior viabilidade em projetos que necessitam de espaço de armazenamento para embarcar outros algoritmos que consomem um espaço significativo. Nas figuras 3(a) 3(b) são apresentados os resultados obtidos pelo experimento nas configurações propostas por este trabalho.

Em todos os experimentos, foi implementada cada uma das operações (cifrar e decifrar) individualmente, uma vez que, não há memória SRAM suficiente na placa Arduino Mega 2560 para embarcar simultaneamente as operações de cifrar e decifrar do Rijndael, uma vez que as variáveis globais utilizadas pelo algoritmos excedem o limite de 8 Kbytes da placa. No caso do Serpent, isso foi possível, mas, para manter o padrão, somente uma operação foi mantida durante a realização do experimento.



(a) Espaço de armazenamento utilizado durante manipulação de dados em SRAM.

(b) Espaço de armazenamento utilizado durante manipulação de dados em SD.

Figura 3. Espaço de armazenamento utilizado

6. Conclusão

Este trabalho apresentou uma comparação entre dois algoritmos robustos e reconhecidamente seguros de criptografia simétrica, que permitem o uso de chave de 256 bits, em suas versões originais submetidas ao concurso AES, embarcados em um Arduino Mega 2560, disponível no GitHub[4]. Essa placa foi escolhida dado que o Arduino Uno teve sua capacidade excedida em mais de 80% para carregar o Serpent e mais de 160% para carregar o Rijndael. É importante ressaltar que o espaço de memória SRAM ocupado pelas variáveis globais utilizadas pelo Rijndael impede que as operações de cifrar e decifrar sejam embarcadas simultaneamente no Arduino Mega 2560 utilizado nos experimentos. Além disso, mesmo quando apenas uma das operações é embarcada o espaço restante pode não comportar outros algoritmos relacionados à aplicação em questão.

Esta comparação pode ser utilizada como apoio à tomada de decisão na escolha do algoritmo que melhor se aplica aos cenários em que é necessário armazenar ou trocar informações sensíveis entre dispositivos de IoT. Vale ressaltar que optamos pela implementação original do Rijndael e do Serpent fornecidas pelos próprios autores à época da realização do concurso AES para que nenhum dos algoritmos fosse prejudicado na análise por conta do uso de implementação otimizada, visto que não se encontrou na literatura bibliotecas otimizadas do Serpent para o Arduino.

[4]https://github.com/octaviovieira/Ptolemy-XV

O comparativo realizado levou em consideração aspectos relacionados ao desempenho e ao consumo de recursos por parte desses algoritmos, sendo eles: 1) tempo necessário para cifrar/decifrar uma mensagem; 2) quantidade de ciclos de *clock* utilizados para cifrar/decifrar; 3) consumo de memória SRAM; 4) consumo de energia; 5) espaço de armazenamento necessário para embarcar os algoritmos. Com os resultados obtidos, conclui-se que: o Rijndael é mais apropriado quando há demanda por um algoritmo que seja rápido e possua baixo consumo energético; já o Serpent é indicado para aplicações onde há restrições relacionadas ao uso de memória e espaço de armazenamento.

Como trabalhos futuros pode-se comparar os demais finalistas do concurso AES: RC6, Twofish e MARS. É possível também realizar a otimização dos códigos originais utilizados, a fim de obter-se um melhor desempenho dos algoritmos em plataformas para IoT, possibilizando inclusive que os mesmos possam ser embarcados em plataformas que possuam ainda mais restrições de recursos do que a utilizada neste trabalho. Além disso, existe a possibilidade de se realizar os experimentos utilizando outros dispositivos além da plataforma Arduino, como, por exemplo, microcontroladores PIC.

Referências

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805.

Daemen, J. and Rijmen, V. (1999). AES proposal: Rijndael.

Frt'ala, T. and Hromada, V. (2014). Lightweight cryptography with AVR 8-bit microprocessor in remote controlling. *16 th ELITECH '14 Conference of Doctoral Students Faculty of Electrical Engineering and Information Technology*.

Hyncica, O., Kucera, P., Honzik, P., and Fiedler, P. (2011). Performance evaluation of symmetric cryptography in embedded systems. In *Intelligent data acquisition and advanced computing systems (IDAACS), 2011 IEEE 6th international conference on*, volume 1, pages 277–282. IEEE.

Jing, Q., Vasilakos, A. V., Wan, J., Lu, J., and Qiu, D. (2014). Security of the internet of things: Perspectives and challenges. *Wireless Networks*, 20(8):2481–2501.

Johannesen, S. T. (2014). Cryptoprocessing on the Arduino-Protecting user data using affordable microcontroller development kits. Master's thesis, NTNU.

Knudsen, R. A. E. B. L. (1998). Serpent: A proposal for the advanced encryption standard. In *First Advanced Encryption Standard (AES) Conference, Ventura, CA*.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., and Roback, E. (2001). Report on the development of the Advanced Encryption Standard (AES). *Journal of Research of the National Institute of Standards and Technology*, 106(3):511.

Roman, R., Zhou, J., and Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266 – 2279.

Zhang, Z.-K., Cho, M. C. Y., Wang, C.-W., Hsu, C.-W., Chen, C.-K., and Shieh, S. (2014). IoT security: ongoing challenges and research opportunities. In *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pages 230–234. IEEE.