Avaliação de Algoritmos de Criptografia e Implementação de um Protocolo Leve para Comunicação entre Dispositivos IoT

Rafael H. Albarello*, Marcio S. Oyamada†, Edson T. de Camargo*†

*Universidade Tecnológica Federal do Paraná (UTFPR) – Toledo – PR – Brasil
†Programa de Pós-Graduação em Ciência da Computação (PPGComp) – UNIOESTE – Cascavel – PR – Brasil
Email: *rafael_albarello@hotmail.com, marcio.oyamada@unioeste.br, edson@utfpr.edu.br*

***Resumo*—A criptografia tem papel fundamental na segurança em Internet das Coisas (IoT). No entanto, algoritmos e protocolos tradicionais de criptografia podem não ser adequados para IoT devido a heterogeneidade, baixa capacidade computacional e consumo energético dos seus dispositivos. O objetivo deste trabalho é avaliar diferentes algoritmos de criptografia e implementar um protocolo leve para comunicação entre dispositivos IoT. A avaliação dos algoritmos considera tempo de execução, uso de CPU, memória e consumo energético. O protocolo emprega os algoritmos SHA-3, AES e ECDH e é projetado a partir dos algoritmos avaliados. Resultados demonstram que o protocolo leva menos de 0,8 segundos para garantir confidencialidade e integridade na troca de mensagens usando uma placa Raspberry Pi como gateway e um ESP32 como dispositivo final.**

***Index Terms*—segurança da informação, internet das coisas, criptografia, ECDH**

I. Introdução

O mercado global de soluções para usuários finais da Internet das Coisas (IoT) ultrapassou 100 bilhões de dólares em receita de mercado pela primeira vez em 2017 [1]. As previsões sugerem que esse número crescerá para cerca de 1,6 trilhão de dólares em 2025 [2]. São dezenas de bilhões de objetos conectados, número que deve somente crescer a medida que a conectividade com a Internet se torna uma característica padrão para um grande número de dispositivos eletrônicos. Com a integração desses dispositivos à Internet, torná-los seguros é uma prioridade. Se os usuários não confiarem nas informações dos dispositivos, haverá relutância em utilizá-los [3]. Empresas estariam dispostas a gastar até 22% mais em dispositivos IoT, caso tenham garantias de que a segurança de informação é incluída [1].

A segurança dos dispositivos IoT se baseia nos mesmos princípios da segurança de informação convencional, ou seja, a confidencialidade, integridade e disponibilidade. Porém, entre os fatores que dificultam garantir a segurança em IoT, estão a heterogeneidade dos equipamentos, a escalabilidade, a limitação do poder computacional e a capacidade energética limitada [4], [5]. A heterogeneidade dificulta o desenvolvimento de uma solução única e definitiva, pois vários modelos de microprocessadores e microcontroladores são utilizados em soluções IoT. A escalabilidade da solução é necessária pois a grande variedade de dispositivos tende a aumentar com a invenção de cada vez mais "coisas". Poucos recursos computacionais geralmente são alocados aos dispositivos IoT para barateá-los e algumas soluções realmente apenas precisam enviar um pequena quantidade de dados. Como se não bastasse, os dispositivos geralmente são construídos e programados para consumirem o mínimo de energia.

Consequentemente, algoritmos e mecanismos de segurança normalmente presentes no dia a dia podem não ser adequados ao poder computacional disponível em plataformas IoT. Assim, são necessários algoritmos de criptografia confiáveis, mas que empreguem pouca capacidade de processamento do dispositivo [6]. A criptografia tem papel fundamental na segurança dos dispositivos IoT pois garante o sigilo e integridade das informações coletadas e movimentadas pela rede, protegendo o dado contra invasores que queiram obter ou alterar a informação. A criptografia também oferece a garantia de que um dispositivo não se passe por outro, por meio da autenticação. Visto que a essência da IoT está na geração e transmissão de informação, sem a criptografia esses dispositivos se tornam vulneráveis e não confiáveis [7]. Sendo assim é necessária a avaliação de diversos algoritmos de criptografia, sejam esses algoritmos já existentes ou especialmente implementados para garantir as propriedades de segurança.

O objetivo deste trabalho é avaliar o desempenho de diferentes algoritmos de criptografia e implementar um protocolo leve para comunicação segura entre dispositivos IoT. São avaliados os algoritmos AES (*Advanced Encryption Standard*), SHA (*Secure Hash Algorithm*), RSA (*Rivest-Shamir-Adleman*) e a troca de chaves ECDH (*Elliptic Curve Diffie-Hellman*) usando o algoritmo curve25519, com diferentes tamanhos de chaves nos dispositivos Arduino Uno, ESP32, Raspberry PI 3 e em um computador de uso pessoal. A avaliação dos algoritmos considera o seu tempo de execução, o consumo energético, o uso de CPU e memória RAM.

A partir dos algoritmos avaliados, um protocolo foi implementado para permitir a comunicação segura entre os dispositivos avaliados. O protocolo emprega a troca de chaves ECDH, o algoritmo SHA-3 e o padrão AES para comunicação entre os dispositivos. Destaca-se que os algoritmos de curva

elíptica, como a troca de chaves ECDH em conjunto com o AES, vêm ganhando o mercado de criptografia assimétrica para IoT por fornecerem o mesmo nível de segurança que algoritmos tradicionais de chave pública, como o RSA, porém utilizando chaves consideravelmente menores, o que os tornam mais leves e rápidos e assim adequados para IoT [8].

Este trabalho segue organizado da seguinte forma. A Seção II descreve a fundamentação teórica e os trabalhos relacionados. A Seção III apresenta a metodologia de avaliação. A Seção IV apresenta os resultados de avaliação dos algoritmos e a Seção V detalha o projeto e a implementação do protocolo. Por fim, a conclusão é apresentada na Seção VI.

II. Internet das Coisas e Criptografia

Esta seção apresenta os conceitos fundamentais sobre IoT e Criptografia e discute ainda alguns trabalhos relacionados.

A. Conceitos Fundamentais

Conforme descrito em [9], a Internet das Coisas (IoT) permite a objetos físicos enxergar, ouvir, pensar e realizar tarefas. Através de uma rede de comunicação de dados, os objetos podem ainda "conversar" uns com os outros para compartilhar informações e coordenar suas ações. Uma das questões fundamentais para o sucesso de IoT é a segurança da informação que, por sua vez, tem entre seus pilares a criptografia [10].

Os algoritmos de criptografia são classificados em cifras simétricas e assimétricas. Na criptografia simétrica, o texto plano é encriptado e decriptado utilizando a mesma chave (ou senha) e algoritmo, normalmente executado na ordem reversa para decriptar. O AES é um algoritmo de cifra de bloco simétrico, utiliza blocos de 128 bits, e suas chaves podem ser de 128, 192 ou 256 bits [10]. Diferentemente da simétrica, a criptografia assimétrica não baseia sua encriptação em apenas uma chave, mas sim em duas, chamadas de chave pública e chave privada. Nesse sistema, uma mensagem encriptada com a chave pública, só pode ser descriptografada com a chave privada correspondente. Um dos pioneiros na criptografia assimétrica é o algoritmo RSA, de 1978 [10].

Em 1976, Whitfield Diffie e Martin Hellman [11] propuseram um protocolo chamado de Protocolo de Troca de Chaves Diffie-Hellman [12]. O objetivo do protocolo é realizar a troca de chaves entre duas partes utilizando chaves públicas e privadas em um canal aberto, e não a criptografia em si. Com essa chave, deve-se utilizar um algoritmo de criptografia simétrica para criptografar as mensagens trocadas pelo meio inseguro. A troca de chaves ECDH utiliza o mesmo princípio da troca de chaves definida por Diffie-Hellman, porém agrega a criptografia de curva elíptica (ECC). Uma curva elíptica [13] consiste de pontos x e y, tal que $x, y \in \mathbb{F}_p$, sendo p um número primo e $\mathbb{F}_p$ um corpo finito de inteiros módulo p, que satisfaçam a equação

$$y^2 = x^3 + ax + b \tag{1}$$

onde $a, b \in \mathbb{F}_p$ satisfazem a equação $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. O conjunto de todos os pontos pertencentes a uma curva E é denotado como $E(\mathbb{F}_p)$, incluindo o ∞ como elemento identidade. Para se gerar uma chave sobre uma curva elíptica, assume-se um ponto P tal que $P \in E(\mathbb{F}_p)$ e P tem ordem prima n. O subgrupo cíclico gerado por P em $E(\mathbb{F}_p)$ é

$$\langle P \rangle = \{\infty,\ P,\ 2P,\ 3P\,,\ ...\,,\ (n-1)P\} \tag{2}$$

Sendo p, E, P e n de conhecimento público, também chamados de características do domínio da curva elíptica, é preciso escolher um d aleatório dentro do intervalo $[1, n-1]$ para ser a chave privada, e encontrar a chave pública Q tal que $Q = dP$. A segurança da chave assimétrica criada sobre uma curva elíptica reside na dificuldade em encontrar a chave privada d sabendo apenas as características do domínio da curva e a chave pública Q. Esse é o *Problema do Logaritmo Discreto da Curva Elíptica* (ECDLP).

A ECC leva vantagem sobre a RSA devido a utilizar uma chave consideravelmente menor para fornecer a mesma segurança [10]. Ao comparar o tamanho de chave necessário ao ECC e ao RSA para garantir um mesmo nível de segurança, o ECC necessita de chaves menores para o mesmo nível de segurança. Uma chave de 224 bits em ECC tem a mesma força de uma chave de 2048 bits no RSA e a diferença entre os dois aumenta exponencialmente conforme aumenta-se o tamanho das chaves [13]. Assim, ECC normalmente é escolhida para ser utilizada em dispositivos com capacidades reduzidas, se tornando uma alternativa para a realidade dos dispositivos IoT.

A utilização do ECDH está descrita no Algoritmo 1 [14]:

Algoritmo 1 Utilização do ECDH

1) A e B definem os parâmetros da curva elíptica que irão utilizar. Essa comunicação pode acontecer em meio inseguro
2) A escolhe S_a sendo um número aleatório no intervalo $[1, n-1]$
3) B escolhe S_b, utilizando o mesmo procedimento anterior
4) A calcula $P_a = S_a * P$, e envia P_a para B
5) B também calcula $P_b = S_b * P$, e envia para A
6) Em posse da chave de B, A calcula $S = S_a * P_b$. Analogamente, B calcula $S = S_b * P_a$
7) Ambos possuem a chave compartilhada S

Um atacante não consegue descobrir a chave compartilhada S sabendo apenas os parâmetros da curva e P_a e P_b, a menos que resolva o problema ECDLP. Algumas implementações do ECDH já definem previamente os parâmetros da curva que irão utilizar, desse modo economizam algumas mensagens para efetuarem a troca de chaves. Esse algoritmo, porém, é suscetível a ataques *Man-In-The-Middle*, e necessitam de algum método de autenticação para prevenir tais ataques.

Criptografia *Hash* é um algoritmo que recebe uma entrada de qualquer tamanho, e retorna uma mensagem de tamanho fixo, esta chamada de *hash*. As funções *Hash* podem ser classificadas em dois grupos: com senha e sem senha. Os sem senhas recebem apenas o texto que se deseja transformar em *hash*, enquanto os com senha (também chamados de *message authentication codes* ou MACs) recebem, além do texto claro, uma senha conhecida apenas pelo remetente e destinatário da mensagem [15]. Os principais algoritmos de criptografia *hash* são os algoritmos da família *Secure Hash Algorithm* (SHA),

publicados pela NIST. Atualmente, o NIST especifica como seguros os algoritmos da família SHA-2, como SHA-256 e SHA-512, e os algoritmos da família SHA-3, como SHA3-256 e SHA3-512 [16].

B. Trabalhos Relacionados

A avaliação de algoritmos de criptografia sobre plataformas *Intel Edison* e a *TelosB* e sistemas operacionais IoT *ContikiOS*[1], *Yocto*[2] e *TinyOS*[3] é realizada em [17]. São avaliados algoritmos de criptografia simétrica, criptografia *hash*, MAC e AEAD (*Authenticated Encryption with Associated Data*). AEAD é um método para criptografar parte de uma mensagem e autenticá-la, garantindo que nem a parte criptografada, nem a parte em texto claro foram alteradas. O trabalho também propõe uma metodologia para realizar as análises de desempenho e análise do consumo energético, que é obtido utilizando um multímetro digital em conjunto com o software *LabView*.

Em [6], os autores apresentam uma arquitetura baseada em *blockchain* para realizar o controle de acesso em IoT. O trabalho também avalia o desempenho dos algoritmos RSA, SHA-256 e AES para os dispositivos Arduino UNO, Raspberry Pi 2, Orange Pi e PC. Com os resultados de desempenho dos algoritmos, os autores definiram que a Raspberry Pi 2 e a Orange Pi poderiam atuar como *gateways* na arquitetura proposta. Já o Arduino UNO seria empregado para controlar sensores e atuadores.

A criptoanálise dos algoritmos ECC e sua comparação com o RSA é realizada em [8]. Foram avaliados o consumo de energia e o tempo da troca de chaves Diffie-Hellman, do algoritmo RSA e do ECDH utilizando o simulador PrimeTime. Os algoritmos analisados foram implementados em *hardware* e *software* pelos autores, que também propuseram uma melhoria na computação de módulo de números fracionais usados na no algoritmo ECC. Também foi feita uma análise da segurança entre esses algoritmos, e o tamanho de chaves para atingirem o mesmo nível de segurança.

Uma investigação de aplicações de algoritmos de ECC e sua comparação com o RSA também foi realizada em [18]. Os autores mencionam as vantagens de se utilizar ECDH para troca de chaves efêmeras em dispositivos IoT, pois é mais leve e consome menos recursos que os algoritmos tradicionais. São ainda examinados vários usos do ECC em um contexto de computação móvel, incluindo telefones celulares.

Um *Internet-draft* descreve como utilizar chaves assimétricas pré-compartilhadas para comunicação de dispositivos IoT na camada de transporte, utilizando ECDH duplos ou triplos para gerar chaves simétricas e garantir a autenticidade das mensagens [19].

O trabalho em [20] apresenta um pesquisa aprofundada do estado da arte de primitivas criptográficas leves disponíveis até 2019. São discutidas 21 cifras de bloco, 19 cifras de fluxo, 9 funções *hash* e 5 variantes de criptografia de curva elíptica. Ou seja, no total 54 primitivas são comparadas em suas respectivas classes. A comparação das cifras é realizada em termos de área de *chip*, energia e potência, eficiência de hardware e software, taxa de transferência, latência e figura de mérito. Com base nos resultados, observou-se que AES e ECC são os mais adequados para primitivas criptográficas leves. Vários problemas de pesquisa em aberto no campo da criptografia leve também são identificados no trabalho, como reduzir o consumo de memória e energia dos algoritmos ECC. Já em [21] é proposto um sistema de comunicação segura para comunicação entre dispositivos em uma rede 5G. A comunicação segura proposta foi projetada com base em ECC e criptografia AEAD para dispositivos IoT com recursos limitados.

Em [22], os autores elencam os requisitos para segurança da comunicação IoT utilizando uma abordagem de 3 camadas IoT. Os autores também criaram uma extensa lista de esquemas de autenticação para IoT, classificando-as com base nas camadas que elas atuam, seus pontos fortes e fracos, se utilizam *tokens*, entre outros critérios.

Assim como em [6], [17], este trabalho avalia algoritmos de criptografia em plataformas de desenvolvimento IoT. No entanto, este trabalho difere de ambos pelas plataformas e algoritmos avaliados. Destaca-se neste trabalho a placa ESP32 e a curva elíptica ECDH x25519. Em [6], a avaliação do consumo de energia não é realizada e, diferente de [17], a avaliação de consumo de energia neste trabalho não conta com o software Labview. Os trabalhos [8], [18]–[20] investigam direta ou indiretamente os algoritmos RSA, ECC e ECDH, SHA-1, como neste trabalho. Porém, nenhum dos trabalhos relacionados implementa um protocolo baseado em ECDH, SHA-3 e AES para troca de mensagens entre dispositivos IoT com base nas avaliações realizadas. Por fim, diferente de [22], este trabalho não avalia e não inclui esquemas de autenticação.

III. Metodologia de Avaliação

Nesta seção, primeiramente, são apresentados os materiais e algoritmos de criptografia avaliados. Posteriormente, descreve-se como as medidas de avaliação são obtidas.

A. Materiais e Algoritmos de Criptografia

Os componentes e equipamentos empregados na avaliação são os seguintes:

- Raspberry Pi3 Model B+, escolhido por ser um computador de baixo custo que funciona com sistemas operacionais como Debian e Ubuntu;
- Arduino Uno R3, escolhido por ser a placa de referência quando se fala em prototipação IoT;
- ESP32, cada vez mais empregado em projetos de IoT, integra interfaces Wi-Fi e Bluetooth;
- Computador pessoal (PC) para comparação;
- Osciloscópio, para medições de consumo de potência.

As especificações dos dispositivos utilizados estão na Tabela I. Dentre os dispositivos escolhidos, o Arduino Uno R3 é o que apresenta menor poder computacional. Suporta um conjunto de instruções de 8 bits e possui somente 2 KB de memória. Por sua vez, o ESP32 possui mais de 10 vezes o poder de processamento do Arduino. Acompanha um *hardware* de aceleração

[1] www.contiki-os.org/
[2] www.yoctoproject.org/
[3] www.tinyos.net/

Tabela I
ESPECIFICAÇÕES DOS DISPOSITIVOS UTILIZADOS.

Componente	Arduino Uno R3	ESP32	Raspberry Pi 3 B+	PC
CPU	ATmega328	Xtensa LX6 Dual-Core	Broadcom Quad-Core	Intel Core i3 Quad-Core
Freq. CPU	20MHz	240MHz	1,4GHz	3,6 GHz
Intruções	8-bit	32-bit	64-bit	64-bit
Memória	2KB SRAM	520KB SRAM	1GB SDRAM	4GB SDRAM

de criptografia e um processador com 2 núcleos. O Raspberry Pi 3 B+ (RPi-3) já suporta um conjunto de instruções de 64 bits, contém um processador com 4 núcleos de alta frequência e suporta sistemas operacionais como Debian e Ubuntu. O PC utilizado para comparação também tem 4 núcleos, porém em uma frequência maior que a da Raspberry, com 4 vezes mais memória. Todos esses dispositivos contam com um contador de tempo interno com precisão de microssegundos, utilizado na medição de tempo de execução.

O algoritmo escolhido para criptografia simétrica de bloco foi o AES, devido a sua relevância e dominância. Para algoritmos de criptografia *hash*, foram testados os algoritmos SHA-2 e SHA-3, escolhidos por serem os recomendados pela NIST. E para criptografia assimétrica, foi utilizado o algoritmo curve25519 para realizar a troca de chaves Diffie-Hellman sobre uma curva elíptica, escolhido por ser um algoritmo ECDH sobre uma curva não patentada. Também foi utilizado o algoritmo RSA em criptografia assimétrica, escolhido por sua relevância. Esses algoritmos de criptografia foram executados nos dispositivos apresentados na Tabela I.

Os algoritmos utilizados para Arduino foram implementados por Rhys Weatherley, e estão presentes na biblioteca Crypto. Os códigos são *open source* e podem ser acessados online [23]. Os algoritmos utilizados para ESP32 estão presentes do *framework* ESP-IDF, fornecido pelo Espressif, fabricante do ESP32. A escolha da biblioteca para o ESP32 se deu pois a mesma faz uso do hardware dedicado à criptografia. Os resultados obtidos com o ESP32 sem utilizar o hardware acelerador criptográfico também foram obtidos utilizando a biblioteca Crypto. Na RPi-3 e no PC foi utilizada a biblioteca Cryptography do Python[4]. A biblioteca utiliza o OpenSSL em segundo plano, executando nativamente os algoritmos de criptografia.

Os parâmetros utilizados para cada teste foram:

- Criptografia simétrica de bloco
  - bloco de entrada de 16 bytes.
  - chaves de 16, 24 e 32 bytes para os algoritmos AES128, AES192 e AES256, respectivamente.
  - modo de criptografia ECB.
- Criptografia *Hash*
  - bloco de entrada de 32 bytes.
- Criptografia assimétrica
  - curve25519: chaves de 32 bytes.
  - RSA: chave de 256 bytes e bloco de 32 bytes.

[4]https://cryptography.io/en/latest/

O tamanho de 16 bytes para o bloco de entrada do algoritmo AES foi utilizado pois esse é o tamanho do bloco real utilizado pelo algoritmo. O modo de criptografia *Eletronic Codebook* (ECB) foi escolhido por ser o mais simples de implementar, apesar de ser o menos seguro dos modos de operação. Portanto, para descobrir a duração do processo de criptografia para uma mensagem maior que 16 bytes, basta aplicar um cálculo de proporção simples. O tamanho do bloco utilizado na criptografia *hash* foi escolhido em 32 bytes pois é o tamanho da chave gerada pela troca de chaves Diffie-Hellman no algoritmo curve25519.

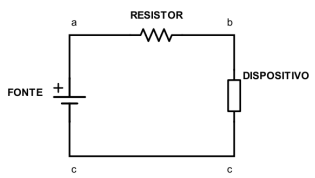
B. Medidas de Avaliação

Os algoritmos serão avaliados de acordo com os seguintes quesitos: tempo de execução do algoritmo, energia consumida durante a execução deste, uso de CPU, e uso de memória RAM. A partir dos dados coletados, é realizada uma análise e são escolhidos os algoritmos para compor o protocolo implementado. Os resultados apresentados são uma média simples de 10 medições independentes. A seguir, descreve-se como as avaliações são realizadas.

Para realizar a avaliação do desempenho de tempo de processamento dos algoritmos são realizadas leituras do *clock* interno do dispositivo exatamente no início e fim da execução do algoritmo a ser testado. Para o arduino foi usado a função *micros()*, que retorna o tempo passado desde o *boot* em microssegundos. Para o ESP32 foi utilizado o hardware de timer disponível pela API do ESP-IDF. Na Raspberry e PC foi utilizada a biblioteca datetime do Python.

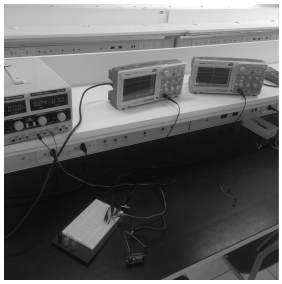
Para mensurar o consumo de energia dos dispositivos foram utilizados dois osciloscópios para medir a corrente sendo drenada da fonte e a tensão fornecida. Para obter a tensão, colocou-se o osciloscópio na saída e na entrada da fonte (pontos *a* e *c* da Figura 1). Para medir a corrente, foi utilizado um *shunt* resistor de 1Ω em série com o dispositivo a ser mensurado e então coletou-se a queda de tensão nesse resistor (pontos *a* e *b*). Como a resistência de entrada do dispositivo é muito maior que 1Ω, a adição do resistor em série não influencia nos valores medidos. Foram utilizados dois osciloscópios sem o terra da tomada, pois nas duas medições as referências são diferentes e o osciloscópio faz uma ligação direta entre a referência da ponteira de prova e o pino Terra da tomada. A Figura 2 é uma foto da medição de energia utilizando os osciloscópios.

Figura 1. Esquemático do circuito de medição



Pela Lei de Ohm, a corrente é igual à tensão dividida pela resistência. Como a resistência utilizada é igual a 1Ω, a queda de tensão medida sobre esse resistor é igual a corrente que

Figura 2. Foto dos osciloscópios e do circuito utilizados nas medições de energia



passa por ele. Conhecendo a corrente, a tensão, e o tempo de duração do algoritmo, é possível calcular o consumo de energia com a equação

$$\omega = \int_{t_0}^{t} vi \; dt \tag{3}$$

que resulta na energia consumida por um componente de t_0 a t, onde v é a tensão e i a corrente [24]. A tensão média foi medida utilizando a função de média do osciloscópio.

As avaliações do consumo de memória RAM para o segmento de dados (variáveis globais) e do segmento de código (instruções), foram obtidas utilizando as informações disponibilizadas pelo compilador nos dispositivos Arduino e ESP32. O código em ambos os dispositivos contém apenas as bibliotecas necessárias para o funcionamento do algoritmo. Nessa análise, não é considerada a memória RAM utilizada para a pilha e o *heap*. Na placa Arduino UNO R3, o compilador escolhido foi o AVR-GCC, utilizado por padrão no ambiente de desenvolvimento Arduino IDE [25]. No ESP32, o compilador utilizado foi GCC, utilizado por padrão pelo ESP-IDF, *framework* criado pela fabricante do dispositivo.

Na avaliação do uso de CPU, foi utilizado o monitor de recursos do sistema do *freeRTOS*, sistema operacional de tempo real do ESP-IDF. Já o Arduino Uno não conta com gerenciador de recursos, o que leva a alocar toda a sua CPU na execução dos algoritmos. Os compiladores foram utilizados em sua configuração padrão.

IV. Resultados da Avaliação dos Algoritmos

A seguir serão apresentados os resultados de tempo de execução, consumo energético, uso de CPU e memória RAM.

A. Tempo de execução

Conforme a Tabela II, o Arduino Uno apresentou resultados extremamente satisfatórios para a criptografia simétrica e *hash*, apesar das suas características de hardware. Decriptar usando o AES com chaves de 128 bits e 256 bits levou cerca de 1 milissegundos (ms) e 1,5 ms, respectivamente. O SHA3-512 levou cerca de 8,3 ms e o SHA512 17,1 ms, ou seja, mais que o dobro. Já o processo de criação e troca de chaves com o algoritmo curve25519 apresentou um tempo maior de execução, com quase 4 segundos para cada etapa. A primitiva dh1 gera as chaves pública e privada e a primitiva dh2 gera a chave compartilhada. Porém, como essas etapas são executadas apenas uma vez por conexão, esse é um valor aceitável.

Os resultados obtidos no ESP32 foram, com exceção do RSA, melhores que na RPi-3 devido à presença de hardware dedicado à aceleração de criptografia que conseguiu superar até o computador nos algoritmos de *hash* criptográfico [26]. A vantagem do ESP32 sobre a RPi-3 e o PC também pode ser explicada pela otimização dos códigos, visto que a biblioteca utilizada para criptografia na RPi3 e no PC não são otimizadas para os devidos equipamentos. Para comparação, os algoritmos da biblioteca Crypto também foram testados no ESP32 e, como esperado, por não utilizarem o acelerador de criptografia, o tempo de execução foi superior.

Na RPi-3, as operações no AES executaram em menos de 0,1 ms. As operações de *hash* levaram entre 10 ms e 9 ms. O RSA executou entre 15 ms e 26 ms. Já as operações dh1 e dh2 executaram em aproximadamente 648 ms e 1,6 ms, respectivamente. Destaca-se que o RPi-3, assim como o PC, foi o único a executar todos os algoritmos. Os resultados obtidos em um computador convencional estão na tabela para efeitos de comparação. Não há resultados do algoritmo RSA para Arduino e ESP32 sem acelerador pois a biblioteca Crypto não contém esse algoritmo. Também não há resultado de SHA-3 e de x25519 para ESP32 com acelerador, pois o acelerador não suporta tais algoritmos.

Comparando os resultados obtidos com o trabalho de Lunardi et. al. [6], no Arduino o AES256 levou 6,5 ms para criptografar e 25,9 ms para decriptar um bloco de mensagem. Em contraponto, neste trabalho o resultado obtido foi 0,79 ms para encriptar e 1,53 ms para decriptar, resultando em um algoritmo 8 vezes mais rápido para encriptar e 16 vezes mais rápido para decriptar. No algoritmo de *hash* criptográfico SHA2-256, o valor obtido na pesquisa de Lunardi et. al. foi de 22,3 ms, enquanto nesse trabalho o resultado foi de 10,76 ms para o microcontrolador calcular o *hash* de um bloco, sendo então mais de 2 vezes mais rápido. O resultado obtido indica que a biblioteca Crypto está otimizada para execução no processador alvo.

Ao comparar os resultados obtidos com os valores fornecidos pela biblioteca Crypto para Arduino [23], observa-se que foram muito próximos para o AES: neste trabalho o AES128 levou 0,56 ms e o autor da biblioteca mediu 0,53 ms. O mesmo não ocorreu para para os algoritmos SHA3-512 e curve25519: neste trabalho o tempo de execução foi 8,3 ms para calcular o *hash* de um bloco e o autor da Crypto anotou 3,64 ms. Também no algoritmo curve25519, tanto na criação quanto na troca de chaves, o tempo de execução foi superior cerca de 1,2 e 1 segundos, respectivamente. Observa-se que o autor apresenta seus resultados por byte, enquanto este trabalho apresenta o tempo total de execução do algoritmo em ms.

B. Consumo energético

Os resultados a seguir não foram testados para a RPi-3 pois não são recursos limitados do dispositivo, como energia e memória RAM, assim como o PC, que foi utilizado na comparação com os outros dispositivos.

Tabela II
TEMPO DE EXECUÇÃO EM MILISSEGUNDOS E CONSUMO DE ENERGIA EM nWh.

Algoritmo	**Tempo**					**Energia**		
	Arduino	ESP32 com acelerador	ESP32 sem acelerador	RPi-3	PC	Arduino	ESP32 com acelerador	ESP32 sem acelerador
AES128: Encrypt	0,564	0,009	0,044	0,077	0,008	0,783	0,226	0,103
AES128: Decrypt	1,080	0,009	0,044	0,049	0,005	1,500	0,226	0,103
AES192: Encrypt	0,676	0,010	0,041	0,091	0,008	0,939	0,251	0,096
AES192: Decrypt	1,308	0,010	0,044	0,106	0,005	1,817	0,251	0,103
AES256: Encrypt	0,792	0,011	0,041	0,093	0,010	1,100	0,276	0,096
AES256: Decrypt	1,532	0,011	0,045	0,044	0,007	2,128	0,276	0,105
SHA256	10,760	0,054	0,082	10	5	11,533	1,323	0,191
SHA512	17,136	0,069	0,203	11	6	11,533	1,691	0,474
SHA3-256	8,304	-	0,319	9	3	14,944	-	0,744
SHA3-512	8,304	-	0,327	9	4	23,800	-	0,763
RSA: Gen Key	-	44545,260	-	21	4	-	1524437,787	-
RSA: Encrypt	-	32,931	-	15	4	-	1639,232	-
RSA: Decrypt	-	1049,159	-	26	7	-	52224,804	-
x25519:dh1	3938,072	-	42,526	648,817	114,151	5469,544	-	99,227
x25519:dh2	3758,036	-	26,256	1,672	0,101	5219,494	-	61,264
Biblioteca	Arduino Crypto	MbedTLS	Arduino Crypto	Cryptography	Cryptography	Arduino Crypto	MbedTLS	Arduino Crypto

Os resultados do consumo de energia dos algoritmos em nWh estão na Tabela II. Para o Arduino, os algoritmos da família AES consomem cerca de $1nWh$, enquanto os da família SHA consomem de 10 a $20nWh$. As duas etapas da ECDH x25519 consomem somadas $10\mu Wh$. No ESP32, é possível perceber que o acelerador de hardware, apesar de tornar a execução do algoritmo mais rápida, faz com que a placa consuma mais energia do que quando o algoritmo é processado na CPU. No caso do SHA-256 o consumo é quase 7 vezes maior. Isso pode ser explicado pois a utilização de outro componente aumenta o consumo geral da placa, fazendo com que a corrente drenada aumente consideravelmente. Os valores da Tabela II referem-se apenas ao consumo dos algoritmos, o consumo das placas no modo *idle* foi subtraído do valor apresentado.

C. Uso de CPU

Os resultados de uso de CPU dos algoritmos para o Arduino UNO R3 foram todos 100%, pois ele não possui um sistema operacional para gerenciar seus recursos. Consequentemente, emprega toda a CPU para executar um algoritmo. Para o ESP32 utilizando o acelerador de hardware, o consumo de CPU sempre foi menor que 1%. Já nos algoritmos que não utilizam o acelerador, o consumo é de 100% de um dos dois processadores do dispositivo, o que equivale a 50% da sua capacidade de CPU. Portanto, além da execução dos algoritmos ser mais rápida com o acelerador, o uso de CPU é mínimo, pois o processamento está no acelerador.

D. Uso de memória RAM

Os algoritmos também foram analisados no consumo de memória RAM do programa para Arduino e ESP32. O Arduino, por não possuir um sistema operacional, utiliza 9 bytes de RAM para o segmento de dados e 444 bytes para o segmento de código quando carregado apenas o necessário para iniciar o Arduino, que são as funções vazias *setup()* e *loop()*. Já o ESP32 utiliza o *freeRTOS*, sistema operacional de tempo real *open-source*, no seu *framework* ESP-IDF, por isso o mínimo necessário para iniciar o ESP32, que é a função *app_main()*, consome 12024 bytes de memória RAM para segmento de dados e 150 kilobytes para o segmento de código. Esses valores foram considerados nas Tabelas III e IV, que representam apenas o incremento nesses valores quando o algoritmo é compilado.

Tabela III
TAMANHO DA IMAGEM DOS ALGORITMOS EM BYTES.

Algoritmo	**Arduino**	**ESP32 com acelerador**	**ESP32 sem acelerador**
AES128: Encrypt	3712	656	944
AES128: Decrypt	3712	652	944
AES192: Encrypt	3718	684	944
AES192: Decrypt	3718	660	944
AES256: Encrypt	3786	672	944
AES256: Decrypt	3786	668	944
SHA256	5444	21204	1964
SHA512	11800	21204	3504
SHA3-256	5866	-	2412
SHA3-512	5874	-	2416
RSA: Gen Key	-	21788	-
RSA: Encrypt	-	33664	-
RSA: Decrypt	-	33380	-
x25519:dh1	7272	-	10652
x25519:dh2	3708	-	2276

A partir da análise dos resultados obtidos, um protocolo para troca de chaves foi construído. O protocolo emprega os algoritmos AES256, SHA3-256 e curve25519 devido aos bons resultados obtidos na avaliação.

V. PROTOCOLO PROPOSTO

O protocolo proposto foi avaliado de acordo com o seu tempo de execução nas trocas de mensagens e processamento. O RPi-3 foi empregado como *gateway*. Conforme a Tabela II, o RPi-3 possui poder computacional e portanto consegue realizar a troca de chaves e manter a comunicação com diversos *end devices* ao mesmo tempo. Como *end devices* estão o ESP32 e o Arduino Uno, pois estes são os dispositivos que estão conectados aos sensores e atuadores e, portanto, precisam se conectar a um servidor.

Tabela IV
USO DE MEMÓRIA RAM DOS ALGORITMOS EM BYTES.

Algoritmo	Arduino	ESP32 com acelerador	ESP32 sem acelerador
AES128: Encrypt	372	12	48
AES128: Decrypt	372	12	48
AES192: Encrypt	404	12	48
AES192: Decrypt	404	12	48
AES256: Encrypt	436	12	48
AES256: Decrypt	436	12	48
SHA256	328	44	120
SHA512	432	44	224
SHA3-256	426	-	224
SHA3-512	426	-	224
RSA: Gen Key	-	-	-
RSA: Encrypt	-	52	-
RSA: Decrypt	-	52	-
x25519:dh1	237	-	80
x25519:dh2	217	-	64

O objetivo do protocolo aqui proposto é permitir a comunicação segura após o ingresso de dispositivos IoT em uma rede, utilizando o algoritmo curve25519, que é a troca de chaves Diffie-Helman utilizando algoritmo de curva elíptica sobre a curva $2^{255} - 19$. Esse algoritmo foi escolhido pois utiliza chaves de 32 bytes, muito menores que as utilizadas por algoritmos de criptografia assimétrica como o RSA, que utiliza chaves de 256 bytes. Conta ainda a seu favor o fato de ser de código aberto sobre uma curva não patenteada [27].

O objetivo do experimento é realizar a troca de chaves entre um *end device* e um *gateway*. Espera-se primeiro que os dispositivos tenham poder computacional suficiente para executar o protocolo e que o tempo de execução seja mínimo. O início da comunicação se dá com o *end device* enviando uma mensagem de *join* para o *gateway*. Esse pedido representa a tentativa do *end device* entrar na rede. Após esse pedido, o *gateway* responde com sua chave pública, criada a partir do curve25519. Em sequência, o *end device* calcula a chave compartilhada utilizando a chave pública recém recebida do *gateway* e sua chave privada.

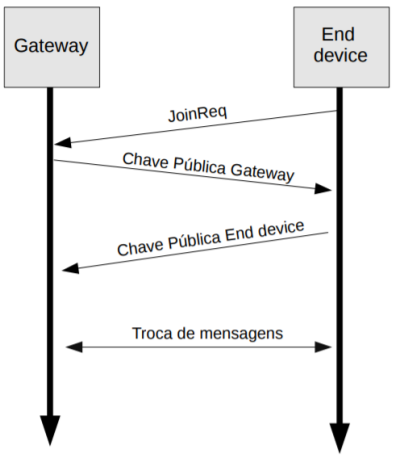


Figura 3. Protocolo de troca de chaves utilizando Curve25519.

Com o cálculo completado, o *end device* envia uma mensagem ao *gateway* com sua chave pública. Ao receber a chave pública do *end device*, o *gateway* também calcula a chave compartilhada utilizando sua chave privada e a chave pública recebida. Com a chave compartilhada em mãos, ambos dispositivos usam o *hash* criptográfico SHA3-256 na chave compartilhada. O resultado desse *hash* é a chave de criptografia simétrica do algoritmo AES256, que ambos os dispositivos utilizarão nessa sessão. A ordem das mensagens está ilustrada na Figura 3.

O Algoritmo 2 descreve um passo a passo detalhado das duas partes na troca de chaves e de mensagens.

Algoritmo 2 Passos do protocolo proposto

1) O *end device* envia o pedido de *join* ao *gateway*
2) O *gateway*, ao receber o pedido de join, calcula suas chaves públicas e privadas utilizando o algoritmo x25519, e envia a chave pública ao *end device* em resposta ao *join*
3) Ao receber a resposta do *join* com a chave pública do *gateway*, o *end device* calcula as suas chaves públicas e privadas utilizando o algoritmo x25519
4) O *end device* calcula a chave compartilhada utilizando sua chave privada e a chave pública do *gateway*
5) Ao calcular a chave compartilhada, o *end device* calcula o *hash* SHA3-256 da chave compartilhada, e envia sua chave pública ao *gateway*
6) Ao receber a chave pública do *end device*, e *gateway* calcula a chave compartilhada utilizando a chave recém recebida e sua chave privada
7) O *gateway* também calcula o *hash* SHA3-256 da chave compartilhada
8) A partir de agora, ambos os dispositivos utilizam o *hash* da chave compartilhada como chave do algoritmo AES256 para troca de mensagens

O tempo de execução do protocolo está na Tabela V. Como esperado, o tempo de execução para o Arduino foi muito maior em comparação ao ESP32. O ESP32 leva cerca de 808 ms para executar o protocolo tendo um RPi-3 como *gateway* e cerca de 252 ms quando o *gateway* é um PC.

Na execução do protocolo, a contagem do tempo inicia quando o *end device* começa a calcular sua chave privada e termina quando calcula a chave compartilhada e envia sua chave pública ao *gateway*, de acordo com os passos 1 a 5 do Algoritmo 2. O principal motivo para não empregar o RSA é a diferença de velocidade entre o AES e o RSA. Conforme visto na Tabela II, o AES pode ser mais de 3.000 vezes mais rápido que o RSA ao encriptar uma mensagem. Após a troca de chaves, as trocas de mensagens serão realizadas utilizando o AES. Como visto na avaliação dos algoritmos na Tabela II, o AES adiciona menos de 1,5 ms na torna a troca de mensagens usando o Arduino e menos de 0,04 ms usando um ESP32. Em termos de consumo de energia, o AES consome no máximo 2,1 nWh e 0,1 nWh no Arduino e ESP32, respectivamente.

A troca de chaves e mensagens propostas pelo protocolo foram implementadas em uma rede local, utilizando pacotes UDP. No código do *gateway* para a Raspberry Pi, foi utilizada uma implementação em Python, utilizando as bibliotecas "Cryptography"[5] para o x25519 e AES, "socket"[6] para comunicação UDP, e "hashlib"[7] para o SHA3-256. As chaves foram geradas e transmitidas no formato *raw*, ou seja, em bytes, sem nenhuma formatação. Foi escolhido esse formato

[5]https://cryptography.io/en/latest/
[6]https://docs.python.org/3/library/socket.html
[7]https://docs.python.org/3/library/hashlib.html

por ser o que gera menor tamanho de mensagem, além de estarem no formato utilizado pelas bibliotecas (bytes).

Para o Arduino e ESP32, foram utilizadas as biblioteca "Crypto"[8] para o x25519, AES e SHA3-256, e "Ethernet"[9] para o protocolo UDP. Assim como no *gateway*, as chaves foram geradas e transmitidas no formato *raw*, evitando assim haver conversões de tipo no *end-device*. Para economizar espaço de memória dinâmica no *end-device*, o vetor de bytes que armazena a chave pública do *gateway* pode ser liberado após a geração da chave compartilhada.

Tabela V
TEMPO DE EXECUÇÃO DO PROTOCOLO EM MILISSEGUNDOS.

Conjunto	Tempo
PC + Arduino	7878,36
RPi + Arduino	8462,19
PC + ESP32	252,39
RPi + ESP32	808,49

VI. CONCLUSÃO

A ampla adoção do conceito de IoT depende de garantir a segurança das informações que transitam por esses dispositivos. Este trabalho avaliou o tempo de execução, o consumo energético, o uso de memória e CPU de importantes algoritmos de criptografia, usados tradicionalmente em computação, em dispositivos Arduino Uno R3 e ESP32 que contam com limitações de hardware. Os resultados obtidos servem de base para um projetista IoT decidir qual algoritmo empregar em sua aplicação. Este trabalho também implementou um protocolo para que um dispositivo IoT entrar em uma rede e trocar mensagens de forma segura, sem que haja a necessidade de inserir chaves no dispositivo previamente. Nota-se que por empregar o algoritmo ECDH e AES, o protocolo é mais eficiente em termos de tempo de execução e segurança do que o algoritmo RSA.

Como trabalhos futuros, tem-se a otimização dos códigos e algoritmos de criptografia para IoT, a realização de testes de desempenho dos algoritmos em mais dispositivos IoT, especialmente dispositivos que utilizam os processadores ARM Cortex-M. Outra possibilidade é aumentar a lista de algoritmos testados nesses dispositivos. Com relação ao protocolo, uma importante etapa a ser desenvolvida ainda é a autenticação dos dispositivos, mitigando o impacto de um ataque *Man-In-The-Middle* durante o processo de troca de chaves.

REFERÊNCIAS

[1] Bain, "Unlocking opportunities in the internet of things," Online, 2018, acessado em 23/06/2019. [Online]. Available: https://www.bain.com/insights/unlocking-opportunities-in-the-internet-of-things/

[2] Statista, "Forecast end-user spending on IoT solutions worldwide from 2017 to 2025," 2020, acesso em 30/03/2020. [Online]. Available: https://www.statista.com/statistics/976313/global-iot-market-size/

[3] S. Upadhyay, "Ongoing challenges and research opportunities in internet of things (IoT)," in *International Journal of Engineering Technologies and Management Research*, 2018, pp. 216–222.

[4] N. Sklavos and I. D. Zaharakis, "Cryptography and security in internet of things (IoTs): Models, schemes, and implementations," *8th IFIP Int. Conference on New Technologies, Mobility and Security (NTMS)*, 2016.

[5] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," in *10th Int. Conf. for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 336–341.

[6] R. C. Lunardi, R. A. Michelin, C. V. Neu, and A. F. Zorzo, "Distributed access control on IoT ledger-based architecture," *Network Operations and Management Symposium (NOMS)*, 2018.

[7] D. Kim and M. G. Solomon, *Fundamentos de segurança de sistemas de informação*. Rio de Janeiro, RJ: LTC, 2014.

[8] T. K. Goyal and V. Sahula, "Lightweight security algorithm for low power IoT devices," in *Int. Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, 2016, pp. 1725–1729.

[9] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[10] W. Stallings, *Criptografia e segurança de redes*, 6th ed. São Paulo: Pearson, 2015.

[11] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[12] U. M. Maurer and S. Wolf, "The diffie–hellman protocol," *Designs, Codes and Cryptography*, vol. 19, no. 2-3, pp. 147–171, 2000.

[13] D. Hankerson, A. J. Menezes, and S. Vanstone, "Guide to elliptic curve cryptography," *Computing Reviews*, vol. 46, no. 1, p. 13, 2005.

[14] C. Lederer, R. Mader, M. Koschuch, J. Großschädl, A. Szekely, and S. Tillich, "Energy-efficient implementation of ecdh key exchange for wireless sensor networks," in *IFIP International Workshop on Information Security Theory and Practices*. Springer, 2009, pp. 112–127.

[15] M. A. AlAhmad and I. F. Alshaikhli, "Broad view of cryptographic hash functions," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 4, p. 239, 2013.

[16] NIST. (2015) Nist releases sha-3 cryptographic hash standard. [Online]. Available: https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard

[17] G. C. C. F. Pereira, R. C. A. Alves, F. L. da Silva, R. M. Azevedo, B. C. Albertini, and C. B. Margi, "Performance evaluation of cryptographic algorithms over IoT platforms and operating systems," *Security and Communication Networks*, vol. 2017, no. 2046735, 2017.

[18] R. Harkanson and Y. Kim, "Applications of elliptic curve cryptography: A light introduction to elliptic curves and a survey of their applications," in *Conference on Cyber and Information Security Research*, ser. CISRC '17. New York, NY, USA: ACM, 2017, pp. 6:1–6:7.

[19] T. Putman. (2017) Ecdh-based authentication using pre-shared asymmetric keypairs for (datagram) transport layer security ((d)tls) protocol version 1.2. Acessado em 09/10/2019. [Online]. Available: https://tools.ietf.org/id/draft-putman-tls-preshared-ecdh-00.html

[20] S. Dhanda, B. Singh, and P. Jindal, "Lightweight cryptography: A solution to secure IoT," *Wireless Personal Communications*, pp. 1–34, 2020.

[21] B. Seok, J. C. S. Sicato, T. Erzhena, C. Xuan, Y. Pan, and J. H. Park, "Secure d2d communication for 5g IoT network based on lightweight cryptography," *Applied Sciences*, vol. 10, no. 1, p. 217, 2019.

[22] M. El-hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A survey of internet of things (IoT) authentication schemes," *Sensors*, vol. 19, no. 5, p. 1141, 2019.

[23] R. Weatherley. (2018) Arduino cryptography library. Acessado em 01/07/2019. [Online]. Available: https://rweather.github.io/arduinolibs/crypto.html

[24] C. K. Alexander and M. N. Sadiku, *Fundamentos de circuitos elétricos*. AMGH Editora, 2013.

[25] Arduino. (2018) Build process. Acessado em 30/09/2019. [Online]. Available: https://github.com/arduino/Arduino/wiki/Build-Process

[26] Espressif, "Esp32 series datasheet," 2019, acessado em 13/08/2020. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[27] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.

[8] https://rweather.github.io/arduinolibs/crypto.html

[9] https://www.arduino.cc/en/reference/ethernet