UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RAFAEL HICKMANN ALBARELLO

AVALIAÇÃO DE ALGORITMOS DE CRIPTOGRAFIA E IMPLEMENTAÇÃO DE UM PROTOCOLO LEVE PARA TROCA DE CHAVES EM DISPOSITIVOS IOT

TRABALHO DE CONCLUSÃO DE CURSO

TOLEDO
2020

RAFAEL HICKMANN ALBARELLO

AVALIAÇÃO DE ALGORITMOS DE CRIPTOGRAFIA E IMPLEMENTAÇÃO DE UM PROTOCOLO LEVE PARA TROCA DE CHAVES EM DISPOSITIVOS IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná - UTFPR Campus Toledo, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Edson Tavares de Camargo
Universidade Tecnológica Federal do Paraná

TOLEDO
2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Toledo
Coordenação do Curso de Engenharia Eletrônica

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Título do Trabalho de Conclusão de Curso Nº 4

AVALIAÇÃO DE ALGORITMOS DE CRIPTOGRAFIA E IMPLEMENTAÇÃO DE UM PROTOCOLO LEVE PARA TROCA DE CHAVES EM DISPOSITIVOS IOT

por

Rafael Hickmann Albarello

Esse Trabalho de Conclusão de Curso foi apresentado às **16h do dia 25 de novembro de 2020** como **requisito parcial** para a obtenção do título de **Bacharel em Engenharia de Computação**. Após deliberação da Banca Examinadora, composta pelos professores abaixo assinados, o trabalho foi considerado **APROVADO**.

Prof. Me. Alexandre Augusto Giron
UTFPR

Prof. Dr. Álvaro Ricieri Castro e Souza
UTFPR

Prof. Dr. Marcio Seiji Oyamada
Unioeste

Prof. Dr. Edson Tavares de Camargo
UTFPR

O termo de aprovação assinado encontra-se na coordenação do curso

Toledo, 25 de novembro de 2020

AGRADECIMENTOS

Aos meu pais Neusa e Gilberto, e minha família, pelo amor, carinho e suporte que me permitiram chegar até aqui.

Aos meu amigos, pelas horas de descontração que tornaram a jornada até aqui muito mais divertida.

Ao professor e orientador Edson Tavares de Camargo, pela enorme ajuda em realizar esse trabalho, e pela oportunidade de aprender sobre o tema.

À todos os professores que fizeram parte da graduação, e tornaram esta uma enorme experiência de vida.

À Trinovati Tecnologia, pela disponibilização de conhecimento e equipamentos para tornar essa pesquisa possível.

Busquem conhecimento. (E.T.B., 2009).

RESUMO

Garantir a segurança das informações em Internet das Coisas (IoT) tem se mostrado como um grande desafio, principalmente pela heterogeneidade, baixa capacidade computacional e consumo energético dos seus dispositivos. No contexto da segurança da informação, a criptografia é responsável por garantir confidencialidade, integridade e autenticação. Este Trabalho de Conclusão de Curso trata sobre a segurança da informação, com enfoque em criptografia, no contexto de IoT. O objetivo é avaliar o desempenho de diferentes algoritmos de criptografia para dispositivos IoT, incluindo o estado da arte, e implementar um protocolo de troca de chaves. A avaliação dos algoritmos considera tempo de execução, uso de memória e consumo energético. O protocolo emprega o algoritmo de troca de chaves Diffie-Hellman sobre uma curva elíptica (ECDH) x25519 e foi projetado a partir dos algoritmos de criptografia avaliados. Os algoritmos são avaliados nos dispositivos Arduino Uno, ESP32 e Raspberry Pi 3. Os resultados obtidos demonstram que é possível garantir a confidencialidade e integridade em dispositivos IoT sem comprometer seu desempenho. O protocolo proposto foi aplicado com sucesso para trocas de chaves em dispositivos IoT com recursos computacionais mínimos.

Palavras-chave: Segurança de Informação. *Internet of Things*. Criptografia. ECDH.

ABSTRACT

Ensuring security in Internet of Things (IoT) has proved to be a great challenge, mainly due to the heterogeneity, low computational power and energy consumption of its devices. In this context, the cryptography is responsible for providing confidentiality, integrity and authentication. This work is about information security, focused on cryptography algorithms for the IoT domain. The goal is to evaluate the performance of different cryptography algorithms on IoT devices, including the state of art, and to implement a key exchange protocol. The algorithm evaluation measured execution time, memory usage and power consumption. The protocol employs the Elliptic Curve Diffie-Hellman (ECDH) key exchange algorithm x25519 and is designed from the cryptography algorithms evaluated. The algorithms are evaluated on the Arduino Uno, ESP32 and Raspberry Pi 3. Results show that it is possible to ensure confidentiality and integrity in IoT devices without compromising its performance. The proposed protocol was successfully applied for a key exchange in IoT devices with minimal computational power.

Keywords: Information Security. Internet of Things. Cryptography. ECDH.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3DES	*Triple DES*
ABINC	Associação Brasileira de Internet das Coisas
AEAD	*Authenticated Encryption with Associated Data*
AES	*Advanced Encryption Standard*
BLE	*Bluetooth Low Energy*
CID	Confidencialidade, Integridade e Disponibilidade
DDoS	*Distributed Denial of Service*
DES	*Data Encryption Standard*
ECB	*Eletronic Codebook*
ECC	Criptografia de Curva Elíptica
ECDH	Troca de chaves Diffie-Hellman sobre curva elíptica
ECDLP	Problema do Logaritmo Discreto da Curva Elíptica
ECIES	*Elliptic Curve Integrated Encryption* Scheme
IoT	*Internet of Things*
LPWAN	Low Power Wide Area Network
M2M	*Machine-to-Machine*
NFC	*Near Field Communication*
NIC.br	Núcleo de Informação e Coordenação do Ponto BR
NIST	*National Institute of Standards and Technology*
RFID	*Radio Frequency Identification*
RSA	Rivest-Shamir-Adleman
SHA	*Secure Hash Algorithm*

SUMÁRIO

1 INTRODUÇÃO

O termo Internet das Coisas (IoT) possui diversas definições. Para Santos et. al., IoT é uma extensão da Internet atual que proporciona aos objetos do dia a dia com capacidade de comunicação, quaisquer que sejam, conectarem-se à Internet. Ainda para Santos et. al., o surgimento do termo se deve aos avanços de várias áreas, como sistemas embarcados, micro-eletrônica, comunicação e sensoriamento (SANTOS et al., 2016). O Núcleo de Informação e Coordenação do Ponto BR (NIC.br) define IoT como uma nova fase da Internet onde a rede passa a interligar vários tipos de objetos e dispositivos inteligentes que vão interagir entre si e conosco, tornando o nosso dia a dia mais fácil (NIC.BR, 2014). O Decreto Nº 9854, de 25 de junho de 2019, que institui o Plano Nacional de IoT, define IoT como a infraestrutura que integra a prestação de serviços de valor adicionado com capacidades de conexão física ou virtual de coisas com dispositivos baseados em tecnologias da informação e comunicação existentes e nas suas evoluções, com interoperabilidade. No decreto, coisas são objetos do mundo físico ou digital, capazes de serem identificados e integrados pelas redes de comunicação (BRASIL, 2019). Já para a Associação Brasileira de Internet das Coisas (ABINC), IoT não é o nome de uma tecnologia mas sim um termo que abrange diferentes tecnologias com implicações profundas nos negócios, na cultura e na vida em sociedade em geral (ABINC, 2015).

De fato, independente das diversas definições apresentadas, o que está por trás do termo IoT é viabilizar o controle remoto de objetos e permitir que tais objetos sejam vistos como provedores de serviços. Ou seja, um sensor de temperatura passa a incorporar a capacidade de comunicação com uma rede e disponibilizar suas leituras à Internet como um serviço. Mais do que isso, os dados coletados desse sensor podem acionar um dispositivo em qualquer lugar do mundo usando os protocolos tradicionais de Internet.

Com a integração desses dispositivos à Internet, surge a necessidade de protegê-los. Garantir a segurança da informação desses dispositivos tem se provado um desafio. A segurança dos dispositivos IoT precisa lidar com questões como heterogeneidade e escalabilidade, visto que IoT engloba vários objetos, como sensores e outros itens que não se classificam necessariamente com computação, como geladeiras, torradeiras, etc. (SKLAVOS; ZAHARAKIS, 2016). Mahmoud et. al. considera que a segurança dos dispositivos IoT se baseia nos mesmos princípios da segurança de informação convencional, ou seja, a confidencialidade, integridade e disponibilidade. Porém, em IoT, existem algumas restrições e limitações, pois os equipamentos IoT são heterogêneos e possuem capacidades computacionais e energéticas limitadas (MAHMOUD et al., 2015). Para Upadhyay, a segurança de IoT está fundamentalmente ligada a confiança dos usuários em seus equipamentos, pois se as pessoas não confiarem nas informações dos dispositivos, haverá relutância em utilizá-los (UPADHYAY, 2018). O autor continua afirmando que por tal motivo, garantir a segurança em produtos e serviços IoT deve ser considerado como prioridade do setor. Para Xu, Wendt e Potkoniak, os principais requisitos da segurança de informação para IoT

incluem autenticação, integridade dos dados, confiança mútua e privacidade (XU; WENDT; POTKONJAK, 2014).

Entre os fatores que dificultam a garantia da segurança em IoT, estão a heterogeneidade dos equipamentos e a limitação do poder computacional. A heterogeneidade dificulta o desenvolvimento de uma solução única e definitiva, pois vários modelos de microprocessadores são utilizados nos diversos microcontroladores disponíveis, e devido a escalabilidade da IoT, essa grande variedade tende a aumentar com a invenção de cada vez mais "coisas". Poucos recursos computacionais geralmente são alocados aos dispositivos IoT para barateá-los. Consequentemente, algoritmos e mecanismos de segurança normalmente presentes no dia a dia podem não ser adequados ao poder computacional presente. Como afirma Lunardi et. al., são necessários algoritmos de criptografia confiáveis, mas que empreguem pouca capacidade de processamento do dispositivo (LUNARDI et al., 2018). Sendo assim é necessária a avaliação de diversos algoritmos de criptografia, sejam esses algoritmos já existentes ou especialmente implementados para garantir as propriedades desejadas de segurança.

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, agregando assim confiança aos dados transmitidos. Visto que a essência da IoT está na geração e transmissão de informação, sem a criptografia esses dispositivos se tornam vulneráveis e não confiáveis (KIM; SOLOMON, 2014). Para Goyal e Sahula, os algoritmos de curva elíptica, como a troca de chaves Diffie-Hellman sobre curva elíptica (ECDH) em conjunto com AES, vem 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 ideais para IoT (GOYAL; SAHULA, 2016). Os protocolos de troca de chaves possibilitam a geração de uma chave por sessão (efêmeras), fazendo com que um eventual vazamento de chaves de um dispositivo não comprometa a segurança da rede, nem de sessões passadas ou futuras desse dispositivo, além de reduzir a quantidade de vezes que uma chave de sessão é utilizada, tornando mais difícil a criptoanálise (RABIAH et al., 2018). Na área de *hash* criptográfico, o mercado de IoT vem utilizando os algoritmos SHA-256 e SHA-512, porém há a tendência de serem substituídos por seus sucessores SHA3-256 e SHA3-512 (RAO; NEWE; GROUT, 2014).

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é avaliar o desempenho de diferentes algoritmos de criptografia e implementar um protocolo leve para troca de chaves em dispositivos IoT.

1.1.2 Objetivos Específicos

Estes são os objetivos específicos deste trabalho:

- pesquisar diferentes algoritmos e bibliotecas de criptografia em uso em dispositivos IoT;
- avaliar o desempenho e o custo de processamento desses algoritmos nos dispositivos Arduino UNO, ESP32 e Rasberry PI 3;
- classificar os algoritmos, com base em tempo de execução, memória utilizada, consumo energético e método de criptografia;
- implementar um protocolo leve para troca de chaves baseado no algoritmo ECDH.

1.2 JUSTIFICATIVA

Segundo o CGI.br (2009), os princípios da Internet no Brasil incluem garantir a privacidade do indivíduo, assim como a segurança deste e de suas informações. Portanto, os equipamentos que estão conectados à rede devem garantir esses princípios aos seus usuários.

Para dispositivos IoT, essa regra não é diferente. Devido a sua natureza de estar em qualquer lugar, a não proteção desses dispositivos pode levar a vazamentos de informações em massa, permitindo atacantes monitorarem as vítimas, sem o conhecimento destas. A falta de segurança de IoT também pode conceder ao atacante controle sobre o dispositivo, abrindo a possibilidade para a alteração nos dados gerados e controle sobre os demais dispositivos, controlados pelo dispositivo vulnerável, ou até servir como ponto de retransmissão para ocultar outros ataques (NETWORKS, 2018).

Porém, esses aparelhos contam com uma dificuldade a mais do que computadores e celulares, pois os dispositivos IoT contam com limite de processamento e de energia, assim como protocolos específicos de comunicação. Outro grande problema é a dificuldade em atualizar o *firmware* desses aparelhos, dificultando e até inviabilizando as atualizações de segurança desses dispositivos. Por isso, os dispositivos IoT precisam garantir que a sua criptografia seja feita de maneira resiliente, porém consuma pouco tempo, processamento e energia (LUNARDI et al., 2018; PEREIRA et al., 2017).

Essa dificuldade em manter os aparelhos IoT seguros resultou em diversas falhas de segurança nos últimos anos. E essas falhas vão além de comprometer os dados do próprio IoT, pois um dispositivo comprometido pode fornecer ao atacante senhas, deixar vulnerável a rede local, e também pode ser utilizado pelo invasor para realizar um ataque *Distributed Denial of Service* (DDoS), por exemplo. É comum que dispositivos IoT permitam acesso externo via SSH, o que permite ao atacante explorar o acesso via porta 22 e, ao obter sucesso em um ataque, configurar remotamente o dispositivo (MOORE; BARNES; TSCHOFENIG, 2017).

Uma brecha de segurança também está na falta de atenção que as empresas desenvolvedoras de equipamentos IoT dão para a segurança destes. Segundo um relatório da Cisco (2018), apenas 13% das empresas acreditavam que *botnets* IoT seriam um grande problema para seus negócios. Porém dados da Kaspersky (2019) mostram que, apesar do número de ataques DDoS

cair de 2017 para 2018, no primeiro semestre de 2019 eles voltaram a crescer devido ao botnet Cayosin, que ataca, entre outros, dispositivos IoT. Botnet é o nome dado a rede de dispositivos comprometidos controlados por um atacante via código malicioso (*malware*). Botnets são utilizados para executar os ataques dos invasores, como ataques DDoS, spam de email, ou utilizados para roubar dados pessoais presentes nos dispositivos infectados (STONE-GROSS et al., 2009).

Na Internet das Coisas, muitos dispositivos são criados para serem *plug-and-play*, ou seja, basta apenas ligá-los que eles já estarão coletando dados e se comunicando com outros dispositivos ou servidores. Isso é possível devido aos avanços nos protocolos utilizados em IoT, principalmente os voltados para comunicação *Machine-to-Machine* (M2M), ou comunicação entre máquinas. Assim, a necessidade de protocolos que garantam a segurança na transmissão da informação desses dispositivos é fundamental para o futuro de IoT (SALMAN; JAIN, 2019).

Visto isso, este trabalho estuda os algoritmos do estado da arte para criptografia em dispositivos IoT, com o objetivo de avaliar o seu desempenho em diferentes aparelhos IoT, fornecendo assim um parâmetro para quem deseja escolher qual tecnologia de criptografia deseja usar em seus projetos IoT, assim como um protocolo de troca de chaves leve e seguro para IoT.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado na seguinte forma: o Capítulo 2 contém a revisão da literatura acerca dos temas abordados nesse trabalho, além de uma seção de trabalhos relacionados. Próximo, está o Capítulo 3, onde são apresentado os métodos utilizados na pesquisa, assim como os materiais empregados. No Capítulo 4, serão apresentados os resultados obtidos nesta pesquisa. Para encerrar, o Capítulo 5 apresenta a conclusão dessa pesquisa, e ideias para pesquisas futuras.

2 REFERENCIAL TEÓRICO

Este Capítulo descreve os conceitos de IoT e de segurança da informação. Especial atenção é dada à criptografia na seção 2.3, onde são apresentados a criptografia de curva elíptica e o protocolo Diffie-Hellman para troca de chaves. Bibliotecas que implementam algoritmos de criptografia para dispositivos IoT também são apresentadas, bem como os trabalhos relacionados encontrados na literatura.

2.1 INTERNET DAS COISAS

Para Al-Fuqaha, a Internet das Coisas (IoT) permite a objetos físicos enxergar, ouvir, pensar e realizar trabalhos pois os objetos "falam"uns com os outros, trocam informações e coordenam decisões. Desse modo, os objetos em IoT deixam de ser objetos tradicionais e se tornam objetos inteligentes, ou *smart objects*, utilizando tecnologias como sistemas embarcados, tecnologias de comunicação, rede de sensores e protocolos de internet (AL-FUQAHA et al., 2015). Aos objetos inteligentes que coletam dados e colaboram para a troca de informações e para a tomada de decisões inteligentes, é dado o nome de "coisas"ou "*things*"(ZORZO et al., 2018).

A possibilidade de monitorar e controlar os dispositivos IoT permite usá-los como geradores de informação, assim fornecendo dados importantes que podem ser utilizados para tomadas de decisões, incluindo enviar comandos ao próprio objeto e a outros. Um exemplo de grande importância da IoT é o conceito de rede elétrica inteligente (*smart grid*), onde o consumo da energia elétrica é monitorado constantemente e em diversos pontos da cidade. Assim, é possível prever a necessidade da energia, e gerar e gerir essa energia da melhor maneira possível, evitando desperdício. Outros benefícios da *smart grid* incluem a detecção de falhas mais rapidamente, manutenção preventiva, melhorar a capacidade e eficiência da rede, entre outros. Outro exemplo que vem fomentando o mercado de IoT são as casas inteligentes (*smart houses*), onde diversos dispositivos IoT controlam e monitoram as casas. Por exemplo: ligar ou desligar o ar-condicionado com base na hora que a pessoa chega em casa, abrir e fechar as portas através do celular, verificar se equipamentos ou luzes estão ligados ou não, e até mesmo exibir informações do dia no espelho do banheiro (FANG et al., 2011).

Segundo um estudo realizado pela Bain (2018), o mercado mundial de IoT movimentou 235 bilhões de dólares em 2017. Esse número crescerá para 520 bilhões de dólares em 2021. De acordo com o estudo, a área de maior crescimento dentro de IoT será a área de *data center* e estatística, com crescimento de cerca de 50% ao ano, seguida da área de integração de sistemas, crescendo 40% ao ano. Ainda segundo o estudo, a principal dificuldade que freia o avanço da IoT é a segurança dos dispositivos IoT. A pesquisa destaca que as empresas estariam dispostas a gastar até 22% mais em dispositivos IoT, caso tivessem garantias de que a segurança de

informação está sendo atendida pelo conceito de IoT.

Os principais protocolos de comunicação empregados em IoT visam oferecer conectividade sem fio e minimizar o consumo energético dos dispositivos. Nesse sentido, destacam-se duas categorias de redes de comunicação: Redes de Baixa Potência e Longo Alcance (LPWAN) e Redes de Baixa Potência e Curto Alcance. Na categoria de LPWAN, os principais protocolos são 3: SigFox e LoRaWAN, dois protocolos criados para utilizarem pouca energia em detrimento da velocidade de transmissão, ideais para comunicação M2M entre sensores e servidor, pois seu alcance pode chegar a mais de 50 quilômetros. O terceiro principal protocolo de LPWAN é a Rede Celular (3G/4G/5G), que diferentemente dos dois anteriores, prioriza velocidade de transmissão sobre consumo de energia, e por esse motivo esse protocolo é usado somente quando a largura de banda é crucial para a aplicação. (AL-SARAWI et al., 2017). Entre os protocolos de curto alcance, os principais são:

- 6LoWPAN - Compatível com o protocolo IP, os dispositivos se conectam via IPv6 nas redes IEEE 802.15.4
- ZigBee - Assim como a 6LoWPAN, utiliza as redes IEEE 802.15.4 e faixas de radiofrequência não licenciadas
- *Bluetooth Low Energy* (BLE) - Utiliza as mesmas frequências do *Bluetooth*, porém com otimizações para baixo consumo de energia e para interação M2M.
- *Radio Frequency Identification* (RFID) - É um método de identificação utilizando radiofrequência de curto alcance onde o dispositivo final não precisa de energia para ser identificado.
- *Near Field Communication* (NFC) - Utiliza a mesma técnica do RFID, porém para distâncias ainda menores (cerca de 10 cm), e permite a comunicação em duas vias e maior confiabilidade na transmissão.

Apesar de promissor, o conceito de IoT prevê dispositivos expostos via Internet e, portanto, a segurança é uma das questões fundamentais para o seu sucesso.

2.2 SEGURANÇA DE INFORMAÇÃO

Para Kim e Solomon (2014, p. 06), segurança de sistemas de informação são as atividades que protegem um sistema de informação e seu armazenamento de dados. Ainda segundo os autores, a segurança de informação é necessária para proteger a propriedade intelectual das empresas, assim como dados pessoais e privados dos utilizadores de um sistema de informação. O Instituto Nacional de Padrões e Tecnologia dos Estados Unidos define segurança de informação como a proteção da informação ou sistema de informação de acesso não autorizado, uso, divulgação, perturbação, modificação ou destruição a fim de garantir confidencialidade, integridade e disponibilidade (NIST, 2017, p. 02).

Kim e Solomon (2014) dizem que para garantir que uma informação seja considerada segura, é necessário atender a 3 objetivos de segurança de informação, a tríade (CID) que são:

- Confidencialidade: somente quem tem permissão pode visualizar informação.

- Integridade: somente quem tem permissão pode alterar informação.
- Disponibilidade: a informação deve estar acessível aos usuários autorizados sempre que a solicitarem.

Os objetivos de confidencialidade e integridade são garantidos através da criptografia.

2.3 CRIPTOGRAFIA

Segundo o dicionário Michaelis, criptografia é "arte ou processo de escrever em caracteres secretos ou em cifras" (WEISZFLOG, 2015). A criptografia é utilizada em sistemas de informação para atender a confidencialidade e integridade, além de oferecer autenticação para as mensagens(KIM; SOLOMON, 2014). Na criptografia, o termo **texto claro** ou *plaintext* é a mensagem original que se deseja transmitir, enquanto o termo **texto cifrado**, ou *cyphertext* é a mensagem já codificada. Transformar o texto claro em cifrado se chama **encriptação**, e transformar texto cifrado em claro, é a **decriptação**. Os algoritmos de encriptação e decriptação são chamados de **cifras** (STALLINGS, 2015).

Para uma cifra ser considerada segura, é necessário embaralhar suficientemente o texto, para que seja computacionalmente inviável descobrir o texto claro conhecendo o texto cifrado e a cifra. Também deve ser impossível descobrir a chave, sabendo o texto claro e cifrado, e a cifra.

Existem duas técnicas para tentar burlar a segurança da criptografia. Uma dessas técnicas é o **ataque por força bruta**, que consiste em testar todas as chaves possíveis, até se obter uma tradução inteligível. Para esse ataque ser possível, é necessário o atacante saber o formato da mensagem a ser descoberta, seja a mensagem um conjunto numérico, um texto em português, ou algum arquivo compactado. Sem conhecer previamente a mensagem é praticamente impossível automatizar a tarefa de distingui-las das mensagens aleatórias geradas a partir de chaves incorretas. A outra técnica de ataque à criptografia é a **criptoanálise**, onde o objetivo é encontrar e explorar falhas no funcionamento do algoritmo, tentando deduzir o texto claro ou a chave utilizada.

Existem várias cifras prontas para serem usadas, podendo estas serem públicas ou proprietárias. As proprietárias muitas vezes recorrem à segurança por obscuridade, onde assume-se que, sem saber como se dá o funcionamento do algoritmo, o atacante encontra dificuldades em encontrar as vulnerabilidades. Porém, as cifras públicas tendem a ser mais confiáveis que as proprietárias, visto que estas estão sujeitas a análises e testes heterogêneos (SILVA; CARVALHO; TORRES, 2003).

2.3.1 Cifras de fluxo e cifras de bloco

As cifras podem ser classificadas em cifras de fluxo ou de bloco. As cifras de fluxo foram criadas para encriptar um bit, ou um byte, por vez. Nesse modelo, a criptografia não depende apenas da chave e do texto claro, mas também dependem do estado atual da cifra. Por depender do estado, as cifras de fluxo também são chamadas de cifras de estado (MENEZES;

OORSCHOT; VANSTONE, 1996). As cifras de bloco encriptam trechos inteiros do texto claro, normalmente em blocos de 64 ou 128 bits. Seu funcionamento se dá numa entrada de tamanho *n*, e uma saída de mesmo tamanho *n*, utilizando técnicas como substituição e permutação. Para a decriptação ser possível, considerando que existem 2^n saídas possíveis, duas entradas diferentes não devem produzir a mesma saída, ou seja, a transformação deve ser não singular (STALLINGS, 2015).

As cifras de bloco são as mais utilizadas hoje, visto que o modelo da comunicação de redes também se dá por blocos (pacotes). Outra desvantagem para as cifras de fluxo é a necessidade frequente da renovação das chaves, pois, com uma quantidade suficiente de texto encriptado, é possível descobrir a chave utilizada. No entanto, segundo Menezes, Oorschot e Vanstone (1996, p.191) as cifras de fluxo são mais rápidas que cifras de bloco em hardware, e possuem um circuito elétrico menos complexo. Este trabalho avalia algoritmos de criptografia que aplicam a cifra de bloco, como o AES e o RSA, pois são mais rápidas em software e melhores para criptografar dados estáticos.

Outra classificação importante dos algoritmos de criptografia é sua divisão em cifras simétricas e assimétricas.

2.3.2 Criptografia Simétrica

Na criptografia simétrica, o texto plano é encriptado e decriptado utilizando a mesma chave (ou senha), através de um algoritmo que possa ser utilizado tanto para a encriptação, quanto para a decriptação, normalmente executado na ordem reversa para decriptar. Um exemplo de algoritmo de criptografia simétrica famoso é o AES, sucessor do DES (STALLINGS, 2015).

A criptografia simétrica requer que apenas as pessoas devidas possuam a chave, pois na chave está baseada toda a criptografia de uma mensagem. O grande problema da criptografia simétrica está em garantir a segurança do compartilhamento da chave. Existem técnicas para garantir a segurança nessa troca de chaves, mas dependem de outros tipos de criptografia para funcionarem (MENEZES; OORSCHOT; VANSTONE, 1996). A seguir os algoritmos AES e DES são apresentados.

O algoritmo Lúcifer surgiu na década de 70, em uma parceria entre uma equipe da IBM e da NSA, como um candidato para se tornar o padrão de encriptação de dados, ou *Data Encryption Standard* (DES). O algoritmo foi escolhido em 1976 pelo governo dos Estados Unidos como padrão para criptografia simétrica sendo renomeado para DES, apesar de algumas críticas de utilizar chaves não suficientemente grandes (BURNETT; PAINE, 2002).

Seu funcionamento é baseado na criptografia de bloco, onde blocos de 64 bits passam por 16 rodadas de permutações e substituições, cada rodada utilizando uma chave diferente, derivada da chave original de 64 bits. Porém, como 8 bits da chave são bits de paridade, o tamanho real da chave é de 56 bits, o que reduz a segurança desse algoritmo. Contudo, para os computadores da época, as mais de 72 quadrilhões de possibilidades mostravam ser suficientes para resistir a um ataque de força bruta. Porém, na década de 90, o DES sucumbiu ao poder dos

novos computadores, tanto que em 1999, na *RSA Conference*, sua senha foi quebrada em menos de 24 horas. Como alternativa, foi utilizado o *Triple DES* (3DES), que consiste em aplicar a criptografia DES três vezes com três chaves independentes, aumentando a segurança para 168 bits. Porém, como normalmente era utilizado a primeira e a terceira chaves iguais, o tamanho da chave cai para 112 bits. Em um ataque ao 3DES com 2 chaves iguais, segundo Barker et al. (2012), é o de combinação de texto claro e texto cifrado. Se o atacante possuir 2^{40} combinações, a segurança estimada é de 80 bits. Caso o atacante possua 2^{56} combinações, a segurança do 3DES é equivalente ao DES, ou seja, 56 bits.

Em 2001, o *National Institute of Standards and Technology* (NIST), órgão governamental americano responsável por fornecer padrões para a tecnologia, publicou o *Advanced Encryption Standard* (AES), um novo algoritmo de criptografia simétrica para substituir definitivamente o DES e 3DES. O AES, assim como o DES, é um algoritmo de cifra de bloco simétrico, utiliza blocos de 128 bits, e suas chaves podem ser de 128, 192 ou 256 bits. O Rijndael, algoritmo vencedor do concurso do AES, ainda aceita chaves de 160 e 224 bits, porém estas não fazem parte da publicação do AES.

Para garantir a segurança da criptografia, o AES utiliza quatro operações distintas, em 10, 12 ou 14 rodadas, dependendo do tamanho da chave, sobre uma matriz de 4 bytes por 4 bytes, totalizando a entrada total de 128 bits. Cada rodada utiliza uma chave diferente de 128 bits, todas derivadas da chave original. As quatro operações, segundo Stallings (2015), são:

- **Transformação de SubBytes** : substituição byte a byte utilizando S-box.
- **Transformação ShiftRows** : consiste no deslocamento/rotação de uma linha da matriz.
- **Transformação MixColumns** : embaralhamento de colunas, onde cada byte da coluna recebe um novo valor com base em sua posição, e nos valores dos outros 3 bytes de sua coluna.
- **Transformação AddRoundKey** : é realizado a operação XOR entre a matriz e a chave da rodada.

2.3.3 Criptografia Assimétrica

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. O contrário também é valido, uma mensagem encriptada com a chave privada, só pode ser descriptografada com a chave pública. Um dos algoritmos pioneiros na criptografia assimétrica é o algoritmo RSA, de 1978 (STALLINGS, 2015).

A criptografia de chave pública, como também é chamada, se tornou revolucionária pois oferece solução para dois grandes problemas da criptografia: a distribuição de chaves e a assinatura digital. O funcionamento dessa técnica assume que os envolvidos na troca de mensagens possuam as chaves públicas de todos. Assim, quando se quer mandar uma mensagem por um meio inseguro, criptografa-se a mensagem com a chave pública do destinatário. Desse modo, só

é possível descobrir a mensagem utilizando a chave privada, que está em posse do destinatário. Como o processo inverso também é possível, pode-se realizar uma assinatura digital, encriptando um trecho da mensagem com sua própria chave privada. Assim, o destinatário descriptografa esse trecho utilizando a chave pública do remetente. Se esse trecho for descriptografado com sucesso, é possível garantir a autoria da mensagem (MENEZES; OORSCHOT; VANSTONE, 1996).

2.3.3.1 RSA

O algoritmo mais famoso e utilizado de criptografia assimétrica é o Rivest-Shamir-Adleman (RSA). O nome do algoritmo é a junção dos sobrenomes dos três criadores desse algoritmo que foi criado em 1977 no MIT, e publicado em 1978. O algoritmo consiste em utilizar cifras de bloco com operações de exponenciação e módulo (STALLINGS, 2015).

As chaves do RSA são definidas como chave pública **PB** $= (e, N)$ e a chave privada **PV** $= (d, N)$, e são obtidas da seguinte maneira:

Algoritmo 1 Passos da geração de chaves do algoritmo RSA

1. Escolhe-se dois números primos p e q suficientemente grandes
2. $N = p * q$, N faz parte das chaves pública e privada
3. Escolhe-se e como parte da chave pública, que atenda as seguintes condições:
   - $1 < e < \phi(N)$, sendo $\phi(N) = (p-1)(q-1)$
   - e é coprimo[1] com $\phi(N)$
4. Calcula-se d para ser parte da chave privada, sendo $d \equiv e^{-1}(\mod \phi(N))$, onde e^{-1} é o inverso modular de $e \mod \phi(N)$
5. **PB** $= (e, N)$; **PV** $= (d, N)$

Com essas chaves, o processo de encriptação ($M \rightarrow M'$), onde M é a mensagem em texto plano e M' é a mensagem criptografada, e decodificação ($M' \rightarrow M$) ocorre da seguinte forma:

- **Encriptação**: $(M)^e \mod N = M'$
- **Decodificação**: $(M')^d \mod N = M$

Desse modo, ambas as partes precisam conhecer o valor de N, mas quem deseja criptografar a mensagem precisa apenas conhecer o valor de e, enquanto quem quer descriptografar a mensagem precisa do valor de d para conhecer o conteúdo da mensagem. A segurança desse algoritmo depende da dificuldade de determinar o valor d conhecendo apenas e e N. Para garantir essa segurança, o método de criação das chaves utiliza um conjunto de números primos grandes, que são difíceis de fatorar.

Assumindo que A tenha escolhido os primos $p = 11$ e $q = 3$, suas chaves públicas e privadas são:

Algoritmo 2 Exemplo de geração de chaves do algoritmo RSA

1. $N = p * q = 11 * 3 = 33$
2. $\phi(N) = (p-1)(q-1) = (10 * 2) = 20$
3. Escolhe-se e coprimo com 20, $e = 3$
4. Encontra-se d tal que $d \equiv e^{-1} \pmod{\phi(N)}$; $d = 7$ via Algoritmo Euclidiano Estendido
5. $PB = (3, 33)$ e $PV = (7, 33)$

2.3.3.2 Troca de Chaves *Diffie-Hellman*

Em 1976, Whitfield Diffie e Martin Hellman publicaram um artigo onde afirmavam que a criptografia se encontrava à beira de uma revolução (DIFFIE; HELLMAN, 1976, p. 664), ao inventarem um protocolo para troca de chaves em um canal aberto de uma forma segura, chamado de Protocolo Diffie-Hellman (MAURER; WOLF, 2000).

O Protocolo Diffie-Hellman foi o primeiro trabalho publicado que tratava de criptografia assimétrica, possibilitando a comunicação segura em um meio inseguro. O objetivo do protocolo é realizar a troca de chaves entre duas partes utilizando chaves públicas e privadas, 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.

O protocolo de troca de chaves funciona da seguinte maneira: escolhe-se um número primo p e um inteiro g que seja raiz primitiva de p. A e B, duas entidades que queiram criar um chave segura para realizarem sua troca de mensagens, devem escolher um número inteiro S_a e S_b tal que S_a,$S_b < p$, para serem suas chaves privadas (STALLINGS, 2015). A partir de suas chaves privadas, cada parte gera sua chave pública P, sendo a chave pública de $P_a = g^{S_a} \mod p$ e $P_b = g^{S_b} \mod p$. Para gerarem a chave compartilhada C, A deve receber a chave pública de B, e vice-versa. Com a chave pública da outra parte, A calcula $C = P_b^{S_a} \mod p$, e B calcula $C = P_a^{S_b} \mod p$,

Aproveitando-se das seguintes propriedades

$$(g^a)^b = g^{ab} = g^{ba} = (g^b)^a \tag{1}$$

e

$$(g^a \mod p) \equiv g^a \pmod{p} \tag{2}$$

temos que

$$(g^a \mod p)^b \equiv (g^a)^b \equiv (g^b)^a \equiv \left(g^b \mod p\right)^a \pmod{p} \tag{3}$$

Desse modo, é garantido que a chave C calculada por ambas as partes é igual, podendo ser utilizada como uma chave simétrica para uma comunicação segura entre A e B. A segurança de que um intruso no meio não consiga descobrir a chave C sabendo $\{P_a, P_b, p, g\}$, é que o atacante precisa calcular o logaritmo discreto para conseguir obter a chave, calculando $S_b = \text{dlog}_{p,g}(P_b)$. Como o logaritmo discreto é muito mais difícil de calcular quando comparado a

exponenciais módulo um primo, esse algoritmo se torna eficiente para quem o utiliza, e, caso o primo escolhido seja grande para a tecnologia atual, é inviável para o atacante descobrir C por força bruta.

Assumindo que A e B tenham escolhidos os parâmetros $p = 17$ e $g = 5$, e que A tenha escolhido sua chave privada $S_a = 10$ e B tenha escolhido $S_b = 3$, temos:

Algoritmo 3 Troca de Chaves Diffie-Hellman

1) $p = 17$; $g = 5$
2) $S_a = 10$; $S_b = 3$
3) $P_a = g^{S_a} \mod p = 5^{10} \mod 17 = 9$
4) $P_b = g^{S_b} \mod p = 5^{3} \mod 17 = 6$
5) $C = P_b^{S_a} \mod p = 6^{10} \mod 17 = \mathbf{15}$
6) $C = P_a^{S_b} \mod p = 9^{3} \mod 17 = \mathbf{15}$

Assim, A e B puderam encontrar uma senha compartilhada $C = 15$ de modo seguro em um meio inseguro. Nesse exemplo foram utilizados valores pequenos de p, S_a e S_b, numa aplicação real o NIST recomenda que se escolha p de 2048 bits e q de 224 bits (BARKER; ROGINSKY, 2018).

Existem também algoritmos de criptografia que não utilizam a operação de módulo ou exponenciação, como é o caso dos algoritmos de criptografia de curva elíptica.

2.3.4 Criptografia de Curva Elíptica

Segundo Hankerson, Menezes e Vanstone (2005, p. 13), uma curva elíptica 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 modulo p, que satisfaçam a equação

$$y^2 = x^3 + ax + b \tag{4}$$

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.

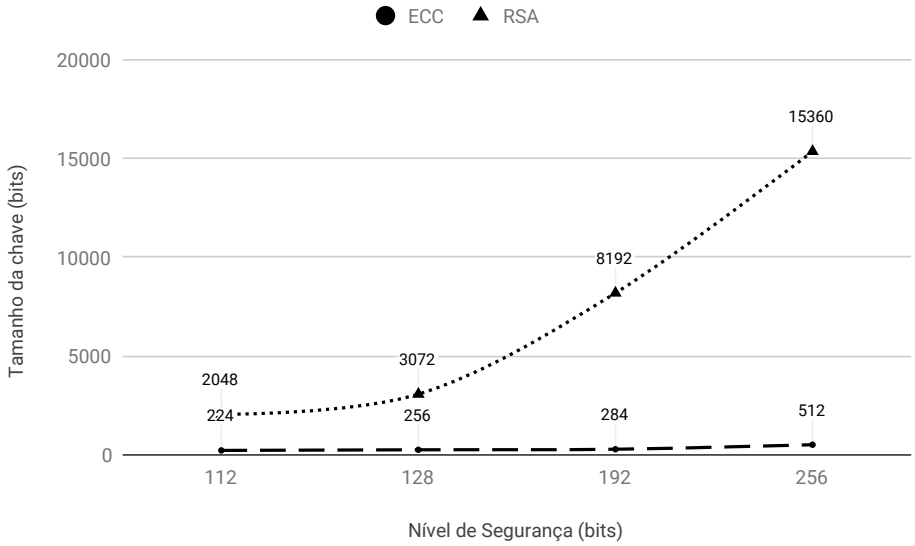
Para se gerar uma chave sobre uma curva elíptica, assume-se um ponto P tal que $P \in E(\mathbb{F}_p)$ e P tem ordem prima n. O subgrupo cíclico gerado por P em $E(\mathbb{F}_p)$ é

$$\langle P \rangle = \{\infty,\ P,\ 2P,\ 3P\ ,\ \ldots\ ,\ (n-1)P\} \tag{5}$$

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).

Figura 1 – Gráfico comparando ECC com RSA



Fonte: Hankerson, Menezes e Vanstone (2005, p. 19)

A técnica mais rápida para resolver problemas de logaritmo discreto é o algoritmo de Pollard rho. Porém, sua eficácia para resolver ECDLP é reduzida quando comparado a resolução de um problema de logaritmo discreto tradicional. Como o esforço computacional de uma Criptografia de Curva Elíptica (ECC) é comparável ao esforço exigido na RSA para um mesmo tamanho de chave. Porém, a ECC leva vantagem sobre a RSA devido a utilizar uma chave consideravelmente menor para fornecer a mesma segurança (STALLINGS, 2015, p. 293-241).

Na Figura 1 é comparado o tamanho de chave necessário ao ECC e ao RSA, para garantir um mesmo nível de segurança entre eles. Como ECC necessita de chaves menores para o mesmo nível de segurança, conforme visto na figura, 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 o nível de segurança empregado. Assim, ECC normalmente é escolhida para ser utilizada em dispositivos com capacidades reduzidas, se tornando uma alternativa para a realidade dos dispositivos IoT.

2.3.4.1 Troca de chaves Diffie-Hellman sobre uma curva elíptica - ECDH

A Troca de Chaves Diffie-Hellman sobre uma curva elíptica (ECDH), funciona sobre uma curva elíptica da mesma maneira que a original funciona sobre grupos cíclicos $\mathbb{Z}_p$, conforme descrito em 2.3.3.2. A vantagem de utilizar ECDH sobre o clássico é a mesma de usar ECC, as chaves são menores para o mesmo nível de segurança, o que faz com que os cálculos sejam menos morosos. A utilização do ECDH está descrita no Algoritmo 4: (LEDERER et al., 2009)

Algoritmo 4 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

Conforme mencionado em 2.3.4, 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.

2.3.5 Criptografia *Hash*

Criptografia *Hash* é um algoritmo que recebe uma entrada de qualquer tamanho, e retorne 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 (ALAHMAD; ALSHAIKHLI, 2013).

Para Stallings (2015), um algoritmo de criptografia *hash* deve garantir que:

- a entrada pode ter qualquer tamanho;
- a saída deve ter tamanho fixo;
- a saída deve ser relativamente fácil de calcular para qualquer entrada;
- deve ser computacionalmente impossível encontrar a mensagem original conhecendo-se o *hash* e o algoritmo;
- deve ser computacionalmente impossível encontrar duas entradas diferentes que gerem saídas iguais;
- a saída obedece os testes padrões de pseudoaleatoriedade.

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 (NIST, 2015).

Na Tabela 1 estão exemplos de algoritmos de hash criptográfico para entradas "UTFPR2018"e "UTFPR2019". Conforme os exemplos, é possível verificar que, apesar da entrada variar apenas o último caractere, o resultado é completamente diferente. Essa característica dos algoritmos de *hash* criptográfico impede a descoberta da mensagem inteira quando se conhece parte dela.

Tabela 1 – Exemplo de hash criptográfico.

Algoritmo	Entrada	Saída
SHA-256	UTFPR2018	de489b3982c46f45b5d029c5...
SHA-256	UTFPR2019	5be94503e2fefaaea4e9b4e7...
SHA3-256	UTFPR2018	1ec34707807e83bbf1b4fbb2...
SHA3-256	UTFPR2019	49f4d5efe5ab39596bc71f45...

2.4 BIBLIOTECAS DE CRIPTOGRAFIA EM IOT

Ao se falar de bibliotecas de criptografia para IoT, é necessário analisar a compatibilidade e otimização para diferentes processadores, visto a heterogeneidade dos dispositivos IoT. Algumas bibliotecas são escritas visando serem facilmente portadas, normalmente escritas em C, buscando facilidade e abrangência, enquanto outras tem como objetivo serem executadas em um processador específico, buscando eficiência (KUMAR; BORGOHAIN; SANYAL, 2015).

A biblioteca **mbedTLS** é uma biblioteca *open source*, escrita em C, com o objetivo de ser portátil para a maioria dos dispositivos embarcados. Antes conhecida por PolarSSL, a biblioteca passou-se a chamar mbedTLS após ser comprada pela empresa ARM. Por ser uma biblioteca SSL/TLS, seu objetivo é estabelecer a segurança da camada de transporte do modelo OSI. Portanto, a biblioteca oferece os protocolos SSL/TLS de cliente e servidor, assim como módulos independentes para diversos algoritmos de criptografia, como o AES, a família SHA-2, RSA, Troca de Chaves Diffie-Hellman, ECDH, ECDSA, gerador de números aleatórios, entre outros (ARM, 2019).

Outra biblioteca SSL escrita para ser compatível com a maioria dos dispositivos embarcados e de tempo real é a **WolfSSL**, biblioteca *open source* e escrita em C, oferece uma API para ser utilizada, e é compatível com a OpenSSL. Ela oferece diversos algoritmos de criptografia, tais como AES, ChaCha20, *hash* criptográfico da família SHA-1 e 2, RSA, ECDH incluindo x25519, ECDSA para autenticação usando curva elíptica, criptografia de curva elíptica com 5 curvas de tipos diferentes, geração de certificados, todos os algoritmos necessários para os protocolos SSL/TLS, além de suporte para criptografia em hardware em diversos *chips* e zlib para compressão de arquivos (WOLFSSL, 2019).

Já a biblioteca **Arduino Crypto**, escrita em C++, foi criada com o objetivo de ser utilizada nos processadores ATmega328 de 8-bits, apesar de ser compatível com todos os dispositivos Arduino e similares. É uma biblioteca *open source*, mantida por Rhys Weatherley, e contém um grande número de algoritmos de criptografia, como o AES e ChaCha para criptografia simétrica, SHA das famílias 1, 2 e 3 para *hash* criptográfico, e algoritmos para ECDH e ECDSA, como o x25519 e Ed25519. Na página de documentação da biblioteca, o autor também oferece os tempos de execução dos algoritmos nos dispositivos Arduino Uno e Arduino Due. (WEATHERLEY, 2018).

A biblioteca **TinyECC** é escrita em nesC e tem o objetivo de dar suporte para algoritmos de criptografia de curva elíptica no sistema operacional TinyOS, com otimizações para as placas

MICA2/MICAz, TelosB/Tmote Sky, BSNV3 e Imote2. A biblioteca oferece suporte para ECDSA, ECDH e *Elliptic Curve Integrated Encryption* (ECIES).O código é *open source* e mantido pela Universidade Pública da Carolina do Norte - EUA (LIU; NING, 2008).

Neste trabalho será empregada a biblioteca mbedTLS para a criptografia no ESP32 utilizando seu *hardware* acelerador de criptografia, pois é a biblioteca padrão do *framework* do dispositivo. Também será utilizada a biblioteca Arduino Crypto para o Arduino Uno e o ESP32. A seguir, são apresentados alguns trabalhos que lidam com análise de desempenho de algoritmos de criptografia em IoT, e também trabalhos sobre protocolos de comunicação para IoT.

2.5 TRABALHOS RELACIONADOS

Nos últimos anos, alguns trabalhos foram publicados abrangendo temas relacionados ao deste trabalho.

Em 2017, Pereira et. al. propuseram a avaliação de algoritmos de criptografia sobre plataformas *Intel Edison* e a *TelosB* e sistemas operacionais IoT *ContikiOS*, *Yokto* e *TinyOS*. Além de realizarem as avaliações de algoritmos de criptografia simétrica, criptografia *hash*, MAC e AEAD[2] sobre os dispositivos, os autores propõem uma metodologia para realizar as análises de desempenho, separando o algoritmo em etapas como *"Init"*, "Criptografia"e "Descriptografia", e análise do consumo energético nos dispositivos IoT, utilizando um multímetro digital em conjunto com o software *LabView* (PEREIRA et al., 2017).

Lunardi et. al. apresentam uma pesquisa sobre realizar o controle de acesso utilizando uma arquitetura baseada em *distributed ledger*. Nesse modelo, os dispositivos se conectam através de *gateways*, que são responsáveis por interconectar os diversos tipos de dispositivos, transformando essas informações no mesmo protocolo. Ao criar mensagens, o dispositivo a assina e envia para o *gateway*, que realiza a segunda assinatura da informação, e envia o bloco para o *distributed ledger*, para todos manterem uma cópia atualizada. Como cada dispositivo é mapeado como um bloco no *ledger*, essa arquitetura utiliza *blockchain*. 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 trabalhar como *gateways*, porém o Arduino UNO só seria suficiente para controlar sensores e atuadores (LUNARDI et al., 2018).

Goyal e Sahula realizaram a criptoanálise dos algoritmos de Criptografia e Curva Elíptica (ECC), para compará-los ao RSA. Nessa análise, os autores coletaram o consumo de energia e de 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 metodologia ECC. Também foi feita uma análise da segurança entre esses

[2] *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.

algoritmos, e o tamanho de chaves para atingirem o mesmo nível de segurança (GOYAL; SAHULA, 2016).

Harkanson e Kim investigaram as aplicações de algoritmos de criptografia de curva elíptica e os compararam com o RSA, realizando testes de tempo em um PC. Nas propostas de aplicação de ECC, 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. Os autores também realizam uma análise do futuro da ECC contra computadores quânticos, e preveem que mesmo a segurança adicional que a ECC oferece atualmente não será suficiente para proteger os ataques quânticos (HARKANSON; KIM, 2017).

Na área de protocolos relacionados, Putman propõe em um *Internet-draft* utilizar chaves assimétricas pré-compartilhadas para comunicação de dispositivos IoT na camada de transporte, utilizando ECDH duplos ou triplos para gerar chaves simétricas, para garantir a autenticidade das mensagens (PUTMAN, 2017). Outro *Internet-draft* proposto por Stebila fala sobre um protocolo de troca de chaves efêmeras utilizando ECDH para autenticação do protocolo SSH. Por ser escrito em 2004, o protocolo de Stebila utiliza SHA-1 para gerar o *hash* das chaves. Os protocolos dos dois autores trabalham sobre curvas elípticas genéricas.

El-hajj et. al. elencam os requisitos para segurança da comunicação IoT utilizando uma abordagem de 3 camadas IoT: *"Perception, Network, Application"*. 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 (EL-HAJJ et al., 2019).

3 MATERIAIS E MÉTODOS

Este Capítulo apresenta os componentes e equipamentos utilizados para avaliar os algoritmos criptográficos, bem como descreve como se dará a avaliação de acordo com os objetivos específicos definidos para o trabalho.

3.1 MATERIAIS

Os materiais utilizados nesse trabalho são:

- Raspberry PI3 Model B+, escolhido por ser um computador barato que funciona com sistemas operacionais como Debian e Ubuntu;
- Arduino Uno R3, escolhida por ser a placa de referência quando se fala em prototipação IoT;
- ESP32, escolhido por possuir processador com 2 núcleos, Wi-Fi e Bluetooth integrados, e conter um *hardware* acelerador de criptografia;
- PC para comparação;
- Osciloscópio, para medições de consumo de potência.

As especificações dos dispositivos utilizados estão na Tabela 2.

Tabela 2 – Especificações dos dispositivos utilizados.

Componente	Arduino Uno R3	ESP32	Raspberry Pi 3 B+	PC
CPU	ATmega328	Xtensa LX6 Dual-Core	Broadcom BCM2837B0 Quad-Core	Intel Core i3 Quad-Core
Frequência CPU	20MHz	240MHz	1,4GHz	3,6 GHz
Conjunto de Intruções	8-bit	32-bit	64-bit	64-bit
Memória RAM	2 KB SRAM	520 KB SRAM	1 GB SDRAM	4 GB SDRAM

Como é possível perceber, dentre os dispositivos escolhidos para avaliação, 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. O ESP32 possui mais de 10 vezes o poder de processamento do Arduino, além disso possui *hardware* de aceleração de criptografia, e um processador com 2 núcleos. A Raspberry Pi 3 B+ já suporta um conjunto de instruções de 64 bits, além de um processador com 4 núcleos e de alta frequência, que suporta sistemas operacionais como Debian e Ubuntu. O PC utilizado para comparação também tem 4 núcleos, porém numa frequência maior que a da Raspberry, com 4 vezes mais memória.

3.2 MÉTODOS

Segundo Marconi e Lakatos, "*não há ciência sem o emprego de métodos científicos*". Visto isso, esse trabalho utiliza o método dedutivo. Método dedutivo é um processo de análise da informação que utiliza o raciocínio lógico e a dedução para obter uma conclusão a respeito de um determinado assunto. Neste trabalho, os algoritmos serão avaliados nos seguintes quesitos: tempo de execução do algoritmo, energia consumida durante a execução deste, uso de CPU, tamanho da imagem e uso de memória RAM. Com os 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. 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. (MARCONI; LAKATOS, 2003).

Para avaliar a energia consumida pelo algoritmo, desejava-se empregar uma metodologia próxima à metodologia proposta por Pereira et al. (2017), que consiste em utilizar um multímetro digital, conectado à um computador com *LabView*, para monitorar a corrente sendo drenada pelo dispositivo. O *LabView* é um *software* para desenvolvimento, testes e medições de sistemas embarcados fornecido pela National Instruments. Antes de realizar a medição do consumo, todos os processos não essenciais ao funcionamento do dispositivo seriam encerrados para não interferirem na medição. Após isso, o valor do consumo médio seria medido executando apenas os processos essenciais. Após subtrair esse valor médio obtido pelo do valor medido com o algoritmo em execução, seria obtido o consumo de energia do algoritmo apenas. Porém, por falta de multímetro com amostragem suficiente para a leitura da execução do algoritmo, adquiriu-se um medidor de corrente por efeito HAL modelo ACS712, em conjunto com um conversor analógico-digital de 12 bits com amostragem de 3,3kS/s, conforme a Figura 2, que apresenta a medição de corrente de um Arduino, onde a saída I2C do ACS712 está sendo lida por uma Raspberry Pi. Porém o medidor se mostrou ineficiente para correntes abaixo de 150mA, que é o caso da maioria dos dispositivos aqui testados.

Como as tentativas anteriores de medir o consumo de corrente falharam, decidiu-se por empregar um osciloscópio 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 3). Para medir a corrente, colocou-se um resistor de 1Ω em série com o dispositivo a ser mensurado, conforme esquemático da Figura 3, e então foi obtida 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 5 é uma foto da medição de energia utilizando os osciloscópios.

Pela Lei de Ohm, a corrente é igual à tensão dividida pela resistência. Como a resistência

Figura 2 – Foto do teste de leitura de corrente do Arduino utilizando o ACS712

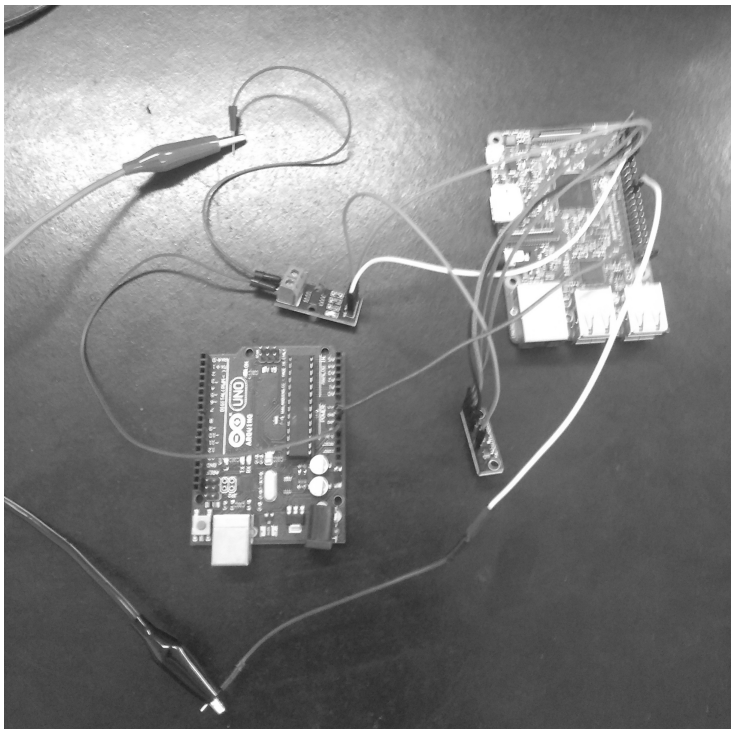
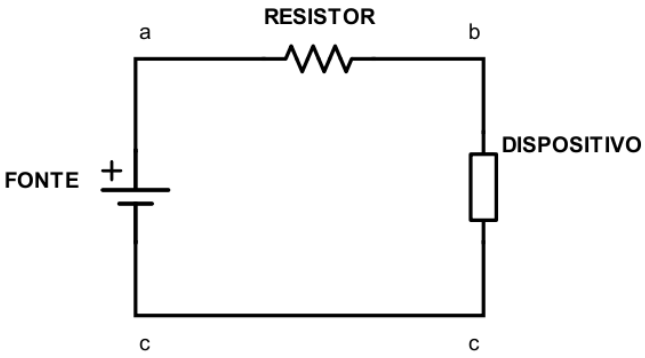


Figura 3 – Esquemático do circuito de medição



Fonte: o autor

utilizada é igual a 1Ω, a tensão medida sobre esse resistor é igual a corrente que passa por ele, Sabendo a corrente, a tensão, e o tempo de duração do algoritmo, é possível calcular o consumo de energia com a equação

$$\omega = \int_{t_0}^{t} vi \, dt \tag{6}$$

que resulta na energia absorvida por um componente de t_0 a t, onde v é a tensão e i a corrente (ALEXANDER; SADIKU, 2013). A tensão média foi medida utilizando a função de média do osciloscópio.

As avaliações do consumo de memória RAM estática e do tamanho da imagem do

Figura 4 – Foto dos osciloscópios e do circuito utilizados nas medições de energia

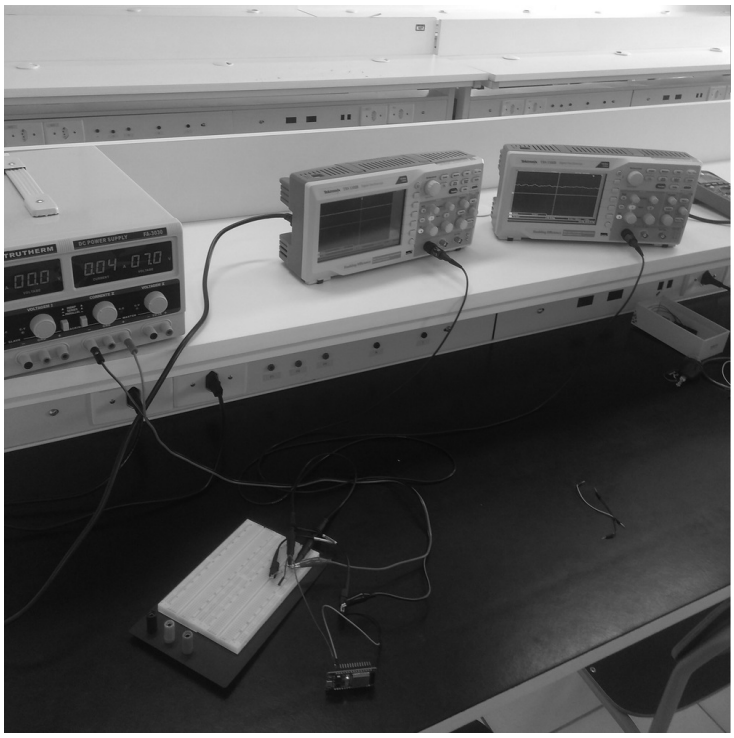
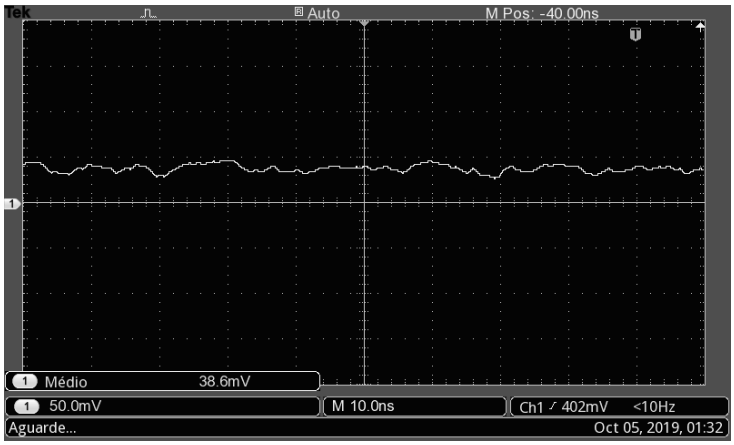


Figura 5 – Exemplo de leitura realizada no osciloscópio



programa foram feitas utilizando funções do compilador dos programas, onde foi carregado apenas o mínimo necessário para que o algoritmo funcionasse. Nessa análise, não é considerada a memória RAM dinâmica, usada para armazenar variáveis. Na placa Arduino UNO R3, o compilador escolhido foi o AVR-GCC, utilizado por padrão no ambiente de desenvolvimento Arduino IDE (ARDUINO, 2018). No ESP32, o compilador utilizado foi o ESP-IDF, *framework* criado pela fabricante do dispositivo. Na avaliação do consumo de CPU, foi utilizado o monitor de recursos do sistema do *freeRTOS*, sistema operacional de tempo real do ESP-IDF. Os compiladores foram utilizados em sua configuração padrão.

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 publicados pela NIST, SHA-2 e SHA-3, escolhidos por serem os recomentados 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.

O protocolo aqui proposto é avaliado conforme o tempo de execução das trocas de mensagem e processamento, e testado utilizando a Raspberry Pi 3 como *gateway*, por possuir um bom poder computacional e portanto conseguir 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.

4 RESULTADOS

Este Capítulo trata sobre os resultados obtidos no trabalho, sendo separado em duas seções: avaliação dos algoritmos e proposta do protocolo.

4.1 AVALIAÇÃO DOS ALGORITMOS

Para medir a eficiência dos protocolos de criptografia existentes, foram testados diferentes algoritmos de criptografia em Arduino Uno R3, em ESP32, Raspberry Pi 3 B+, e um PC. Todos esses dispositivos contam com um contador de tempo interno com precisão de microssegundos, utilizado na medição de tempo de execução.

Os algoritmos utilizados para Arduino foram implementados por Rhys Weatherley, e estão presentes na biblioteca Crypto disponível para Arduino. Os códigos são *open source* e podem ser acessados online[1]. 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 a criptografia. Os resultados obtidos com o ESP32 sem utilizar o hardware acelerador criptográfico também foram obtidos utilizando a biblioteca Crypto. Na Raspberry Pi e no PC foi utilizado a biblioteca Cryptography do Python.

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.

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 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.

[1] O repositório da biblioteca está disponível em https://rweather.github.io/arduinolibs/. Acessado em 23/06/2019

A seguir serão apresentados os resultados de tempo de execução, consumo energético, uso de memória RAM estática e tamanho da imagem.

4.1.1 Tempo de execução

Os resultados de tempo de execução obtidos estão contidos na Tabela 3. No Arduino UNO, conforme pode-se observar, foram obtidos resultados extremamente satisfatórios tanto para a criptografia simétrica, quanto para a *hash*. 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. Porém, como essas etapas são executadas apenas uma vez por conexão, esse é um valor aceitável.

Tabela 3 – Tempo de execução dos algoritmos em milissegundos.

Algoritmo	**Arduino**	**ESP32 com acelerador**	**ESP32 sem acelerador**	**RPi-3**	**PC**
AES128: Encrypt	0,564	0,009	0,044	0,077	0,008
AES128: Decrypt	1,08	0,009	0,044	0,049	0,005
AES192: Encrypt	0,676	0,010	0,041	0,091	0,008
AES192: Decrypt	1,308	0,010	0,044	0,106	0,005
AES256: Encrypt	0,792	0,011	0,041	0,093	0,01
AES256: Decrypt	1,532	0,011	0,045	0,044	0,007
SHA256	10,760	0,054	0,082	10	5
SHA512	17,136	0,069	0,203	11	6
SHA3-256	8,304	-	0,319	9	3
SHA3-512	8,304	-	0,327	9	4
RSA: Gen Key	-	44545,260	-	21	4
RSA: Encrypt	-	32,931	-	15	4
RSA: Decrypt	-	1049,159	-	26	7
x25519:dh1	3938,072	-	42,526	648,817	114,151
x25519:dh2	3758,036	-	26,256	1,672	0,101
Biblioteca	Arduino Crypto	MbedTLS	Arduino Crypto	Cryptography	Cryptography

Os resultados obtidos no ESP32 foram, com exceção do RSA, melhores que na Raspberry PI 3, devido à presença de hardware dedicado à aceleração de criptografia, que conseguiu superar até o computador nos algoritmos de *hash* criptográfico (ESPRESSIF, 2019). A vantagem do ESP32 sobre a RPi3 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, os resultados foram mais lentos do que quando usando o acelerador. Na Raspberry PI 3, todos os algoritmos, exceto a criação de chaves com o curve25519, demoraram alguns milissegundos para serem executados. A execução das variações do AES demoraram menos de 1 milissegundo. 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., no Arduino a encriptação AES256 levou 6,5 ms para criptografar e 25,9 ms para descriptografar um bloco de mensagem no trabalho de Lunardi et. al. Em contraponto, neste trabalho o resultado obtido para encriptação foi 0,79 ms para encriptação e 1,53 ms para decriptação, resultando em um algoritmo 8 vezes mais rápido para criptografia, e 16 vezes mais rápido para descriptografia. 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, devido a implementações diferentes dos algoritmos (LUNARDI et al., 2018).

Comparando com os valores fornecidos pelo desenvolvedor da biblioteca Crypto para Arduino, Rhys Weatherley, os valores obtidos foram muito próximos com os obtidos nesse trabalho, exceto nos algoritmos SHA3-512, onde levamos menos tempo para calcular o *hash* de um bloco, e o algoritmo curve25519, tanto na criação de chaves quanto na troca de chaves, variando cerca de 1,2 e 1 segundos, respectivamente, a mais para executarmos o algoritmo (WEATHERLEY, 2018).

4.1.2 Consumo energético

Os resultados a seguir não foram testados para a Raspberry Pi 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.

Os resultados do consumo de energia dos algoritmos em nWh estão na Tabela 4. Para o Arduino, os algoritmos da família AES consomem cerca de $1nWh$, enquanto os da família SHA consomem de 10 a $20nWh$, e 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 4 referem-se apenas ao consumo dos algoritmos, o consumo das placas no modo *idle* foi subtraído do valor apresentado.

4.1.3 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, então ao executar um algoritmo, todos os recursos são destinados a ele. Para o ESP32 utilizando o acelerador de hardware, o consumo de CPU sempre foi menor que 1%, enquanto nos algoritmos que não utilizam o acelerador, o consumo é de 100% de um dos dois processadores do dispositivo, o que

Tabela 4 – Consumo de energia dos algoritmos em nWh.

Algoritmo	**Arduino**	**ESP32 com acelerador**	**ESP32 sem acelerador**
AES128: Encrypt	0,783	0,226	0,103
AES128: Decrypt	1,500	0,226	0,103
AES192: Encrypt	0,939	0,251	0,096
AES192: Decrypt	1,817	0,251	0,103
AES256: Encrypt	1,100	0,276	0,096
AES256: Decrypt	2,128	0,276	0,105
SHA256	11,533	1,323	0,191
SHA512	11,533	1,691	0,474
SHA3-256	14,944	-	0,744
SHA3-512	23,800	-	0,763
RSA: Gen Key	-	1524437,787	-
RSA: Encrypt	-	1639,232	-
RSA: Decrypt	-	52224,804	-
x25519:dh1	5469,544	-	99,227
x25519:dh2	5219,494	-	61,264

equivale a 50% da sua capacidade de CPU, conforme Tabela 5. Portanto, além da execução dos algoritmos ser mais rápida com o acelerador, o uso de CPU é mínimo, pois o processamento está no acelerador.

Tabela 5 – Uso de CPU dos algoritmos no ESP32 em porcentagem.

Algoritmo	**Arduino**	**ESP32 com acelerador**	**ESP32 sem acelerador**
AES128: Encrypt	100%	<1%	50%
AES128: Decrypt	100%	<1%	50%
AES192: Encrypt	100%	<1%	50%
AES192: Decrypt	100%	<1%	50%
AES256: Encrypt	100%	<1%	50%
AES256: Decrypt	100%	<1%	50%
SHA256	100%	<1%	50%
SHA512	100%	<1%	50%
SHA3-256	100%	-	50%
SHA3-512	100%	-	50%
RSA: Gen Key	-	<1%	-
RSA: Encrypt	-	<1%	-
RSA: Decrypt	-	<1%	-
x25519:dh1	100%	-	50%
x25519:dh2	100%	-	50%

4.1.4 Uso de memória RAM estática e tamanho da imagem

Os algoritmos também foram analisados no consumo de memória RAM estática e o aumento no tamanho da imagem do programa para Arduino e ESP32. O Arduino, por não possuir um sistema operacional, utiliza 9 bytes de RAM estática e 444 bytes na imagem 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 estática e 150 kilobytes na imagem. Esses valores foram considerados nas tabelas abaixo, que representam apenas o incremento nesses valores quando o algoritmo é compilado.

Tabela 6 – 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

Com a análise dos resultados obtidos, foram escolhidos os algoritmos AES256, SHA3-256 e curve25519 para a implementação do protocolo de troca de chaves e posterior comunicação criptografada.

4.2 IMPLEMENTAÇÃO DO ALGORITMO ECDH

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. Outro motivo para utilizar esse algoritmo é ele ser de código aberto (*open source*) sobre uma curva não patenteada (BERNSTEIN, 2006).

Tabela 7 – 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 experimento é realizar a troca de chaves entre um *end point* ou *end device* e um *gateway*. Neste trabalho serão usados um Arduino UNO R3 e um ESP32 como *end device*, e uma Raspberry Pi 3 como *gateway*. A troca deve ser o mais rápida possível, utilizando o mínimo de memória possível. A Figura 6 mostra um exemplo de configuração utilizando a Raspberry Pi como *gateway* e o Arduino como *end device*, realizando a troca de mensagens por pacotes UDP em rede local.

O início da comunicação se dá com o *end point* 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 point* calcula a chave compartilhada utilizando a chave pública recém recebida do *gateway* e sua chave privada.

Com o cálculo completado, o *end point* envia uma mensagem ao *gateway* com sua chave pública. Ao receber a chave pública do *end point*, 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 7. O Algoritmo 5 descreve um passo a passo detalhado das duas partes na troca de chaves e de mensagens.

Figura 6 – Foto da configuração utilizando RPi como *gateway* e Arduino como *end device*

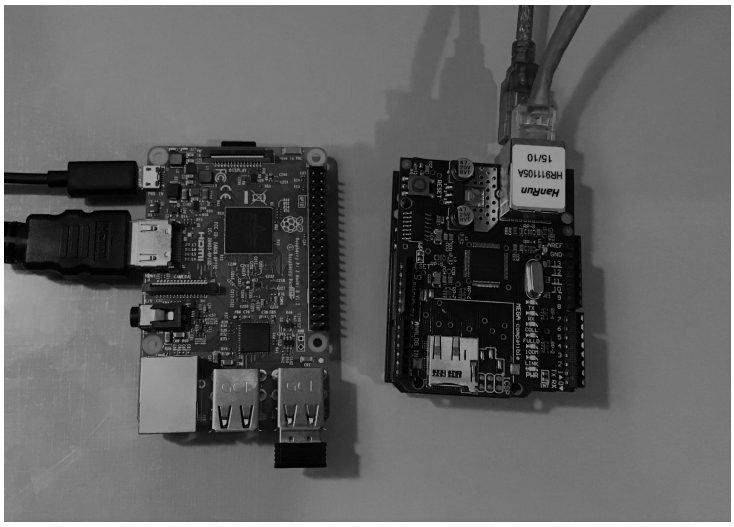
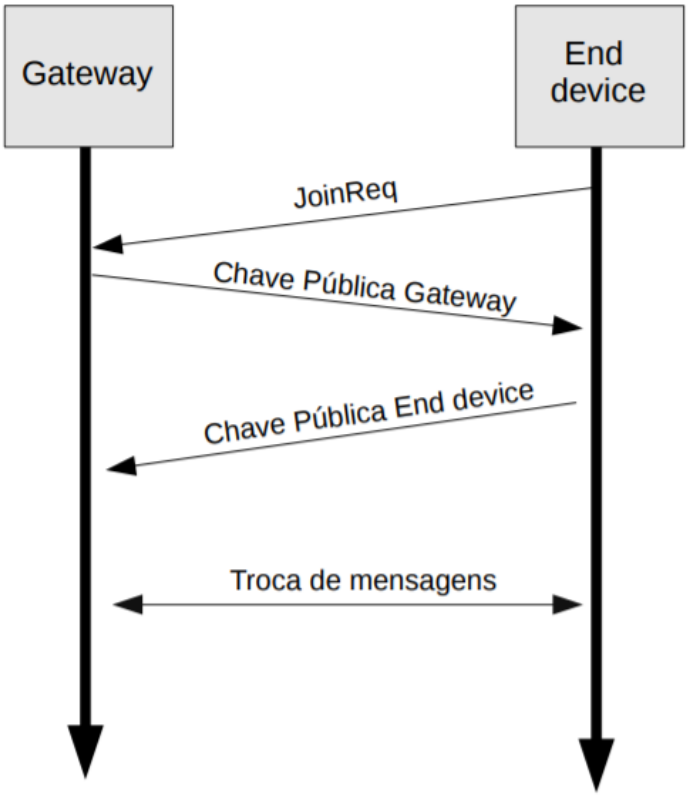


Figura 7 – Protocolo de troca de chaves utilizando Curve25519



Fonte: o autor

Algoritmo 5 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*, o *gateway* calcula a chave compartilhada

Os resultados da latência para realizar o protocolo estão na tabela 8. O início da contagem do tempo se deu ao *end device* começar a calcular sua chave privada, e terminou quando calculou a chave compartilhada e enviou sua chave pública ao *gateway*. Foram utilizados como *gateways* o PC e a Raspberry Pi 3, e como *end device* o Arduino R3 e o ESP32. Porém, o principal motivo para usar esse protocolo ao invés do RSA é a diferença de velocidade entre o AES e o RSA. Conforme visto na Tabela 3, o AES pode ser mais de 3.000 vezes mais rápido que o RSA ao encriptar uma mensagem. Como após a troca de chaves ser terminada, toda a troca de mensagens será realizada utilizando o AES, a maior velocidade do AES realmente faz a diferença.

Tabela 8 – 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

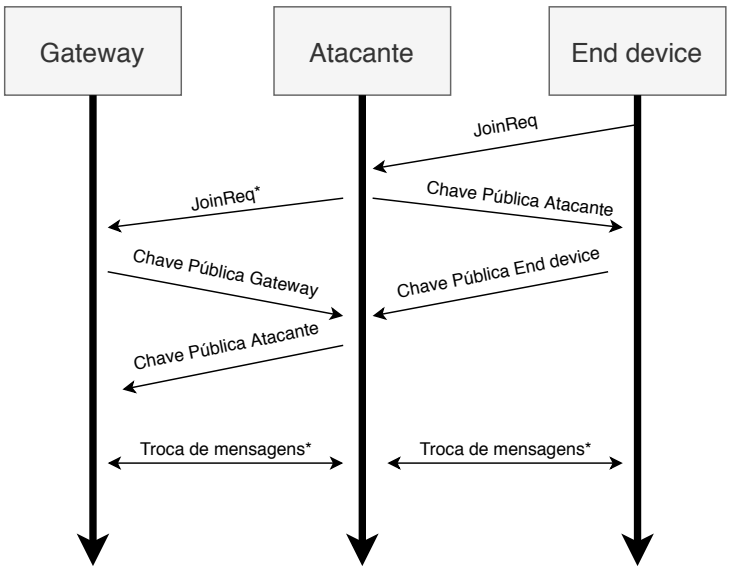
4.2.1 Discussão sobre autenticação

O protocolo acima proposto não oferece autenticação aos dispositivos que entram na rede. As dificuldades de autenticação são causadas pois esse protocolo assume que não há qualquer interação prévia entre *gateway* e *end device*. Desse modo, não há como o *end device* ter certeza que está se comunicando diretamente com o *gateway* e vice-versa. Ou seja, sempre há a possibilidade de um ataque *Man-In-The-Middle*, onde um atacante intercepta e altera as mensagens entre as duas partes.

Em um cenário onde o ataque *Man-In-The-Middle* é efetuado, o atacante realiza a troca de chaves com ambos *gateway* e *end device*. Desse modo, o atacante pode tanto criar e alterar mensagens, quanto apenas ouví-las, repassando a mensagem de uma parte a outra. Uma ilustração de um ataque *Man-In-The-Middle* está na Figura 8.

Um método de assinatura de mensagens não fornece a garantia de autenticação, pois como as partes não conhecem previamente uma a outra, o atacante sempre pode simular um *gateway* para o *end device* e vice-versa. Assim, mesmo assinando a mensagem, o atacante ainda possui o conteúdo dela em texto plano, então ele pode simplesmente remover a assinatura original e colocar a sua assinatura quando for repassar a mensagem.

Figura 8 – Protocolo de troca de chaves com ataque Man-In-The-Middle



Fonte: o autor

5 CONCLUSÃO

A Internet das Coisas tem um futuro promissor, e cada vez mais dispositivos e aplicações serão criados para melhorar a vida das pessoas. Seu futuro, porém, depende da garantia da segurança das informações que transitam por esses dispositivos. Este trabalho apresentou diversos desafios para se garantir a segurança nesses dispositivos, que contam com limitações de hardware, energia, tempo, etc.

Considerando essas limitações, nesse trabalho foram realizados testes e medições para verificar o consumo de tempo, energia, CPU e memória de diversos algoritmos de criptografia do estado da arte, para os dispositivos Arduino Uno R3 e ESP32. Esses resultados foram obtidos com a intenção de servirem para um projetista IoT poder decidir qual algoritmo ele deseja utilizar em sua aplicação, e com isso concluindo os 3 primeiros objetivos específicos desse trabalho, que são pesquisar, avaliar e classificar esses algoritmos para os dispositivos IoT.

Também nesse trabalho foi implementado um protocolo para que um dispositivo IoT possa entrar em uma rede nova e trocar mensagens de forma segura, sem que haja a necessidade de inserir chaves no dispositivo previamente, além do protocolo ser mais eficiente do que algoritmos de criptografia assimétrica como RSA. Nesse protocolo, são utilizados algoritmos de Troca de Chaves Diffie-Helman sobre uma Curva Elíptica, hash criptográfico e criptografia simétrica. Com esse resultado, o último objetivo específico do trabalho, de implementar e avaliar o protocolo para IoT, também é concluído.

Algumas dificuldades foram encontradas na realização deste trabalho. A principal delas foi na medição do consumo energético dos dispositivos, pois a ideia inicial de realizar as medições utilizando o *LabView* foi inviabilizada por falta de equipamentos e recursos. A segunda tentativa falhou pois as características do dispositivo de medição escolhido não atendiam as necessidades desse trabalho. Outra dificuldade foi encontrar bibliotecas de criptografia compatíveis para executarem o algoritmo x25519, pois nem todas as bibliotecas fornecem a chave pública em hexadecimal.

5.1 TRABALHOS FUTUROS

Como trabalhos futuros relacionados a este, tem-se a realização dos testes de desempenho dos algoritmos em mais dispositivos IoT, especialmente dispositivos que utilizam os processadores ARM Cortex-M. Outra sugestão é a de 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

ABINC. **ABINC**. 2015. Acessado em 26/06/2019. Disponível em: <https://abinc.org.br/abinc/>. Citado na página 1.

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE communications surveys & tutorials**, IEEE, v. 17, n. 4, p. 2347–2376, 2015. Citado na página 5.

AL-SARAWI, S. et al. Internet of things (iot) communication protocols. In: IEEE. **2017 8th International conference on information technology (ICIT)**. [S.l.], 2017. p. 685–690. Citado na página 6.

ALAHMAD, M. A.; ALSHAIKHLI, I. F. Broad view of cryptographic hash functions. **International Journal of Computer Science Issues (IJCSI)**, v. 10, n. 4, p. 239, 2013. Citado na página 14.

ALEXANDER, C. K.; SADIKU, M. N. **Fundamentos de circuitos elétricos**. [S.l.]: AMGH Editora, 2013. Citado na página 20.

ARDUINO. **Build Process**. 2018. Acessado em 30/09/2019. Disponível em: <https://github.com/arduino/Arduino/wiki/Build-Process>. Citado na página 21.

ARM. **SSL Library mbed TLS / PolarSSL**. 2019. Acessado em 11/11/2019. Disponível em: <https://tls.mbed.org/>. Citado na página 15.

BAIN. **Unlocking Opportunities in the Internet of Things**. 2018. Online. Acessado em 23/06/2019. Disponível em: <https://www.bain.com/insights/unlocking-opportunities-in-the-internet-of-things/>. Citado na página 5.

BARKER, E. et al. Recommendation for key management part 1: General (revision 3). **NIST special publication**, v. 800, n. 57, p. 1–147, 2012. Citado na página 9.

BARKER, E.; ROGINSKY, A. **Transitioning the Use of Cryptographic Algorithms and Key Lengths**. [S.l.], 2018. Citado na página 12.

BERNSTEIN, D. J. Curve25519: new diffie-hellman speed records. In: SPRINGER. **International Workshop on Public Key Cryptography**. [S.l.], 2006. p. 207–228. Citado na página 27.

BRASIL. **DECRETO Nº 9.854, DE 25 DE JUNHO DE 2019**. 2019. Acessado em 26/06/2019. Disponível em: <http://www.in.gov.br/en/web/dou/-/decreto-n-9.854-de-25-de-junho-de-2019-173021041>. Citado na página 1.

BURNETT, S.; PAINE, S. **Criptografia e segurança: o guia oficial RSA**. [S.l.]: Gulf Professional Publishing, 2002. Citado na página 8.

CGI.BR. **RES/2009/003/P - Princípios para a governança e uso da internet no Brasil**. 2009. Citado na página 3.

CISCO. **Cisco 2018 Annual Cybersecurity Report**. 2018. Online. Disponível em: <https://www.cisco.com/c/dam/m/digital/elq-cmcglobal/witb/acr2018/acr2018final.pdf>. Citado na página 3.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE transactions on Information Theory**, IEEE, v. 22, n. 6, p. 644–654, 1976. Citado na página 11.

EL-HAJJ, M. et al. A survey of internet of things (iot) authentication schemes. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 19, n. 5, p. 1141, 2019. Citado na página 17.

ESPRESSIF. **ESP32 Series Datasheet**. 2019. Online. Acessado em 25/06/2019. Citado na página 24.

FANG, X. et al. Smart grid—the new and improved power grid: A survey. **IEEE communications surveys & tutorials**, IEEE, v. 14, n. 4, p. 944–980, 2011. Citado na página 5.

GOYAL, T. K.; SAHULA, V. Lightweight security algorithm for low power iot devices. In: IEEE. **2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [S.l.], 2016. p. 1725–1729. Citado 2 vezes nas páginas 2 e 17.

HANKERSON, D.; MENEZES, A. J.; VANSTONE, S. Guide to elliptic curve cryptography. **Computing Reviews**, v. 46, n. 1, p. 13, 2005. Citado 2 vezes nas páginas 12 e 13.

HARKANSON, R.; KIM, Y. Applications of elliptic curve cryptography: A light introduction to elliptic curves and a survey of their applications. In: **Proceedings of the 12th Annual Conference on Cyber and Information Security Research**. New York, NY, USA: ACM, 2017. (CISRC '17), p. 6:1–6:7. ISBN 978-1-4503-4855-3. Citado na página 17.

KASPERSKY. **DDoS attacks in Q1 2019**. 2019. Online. Acessado em 23/06/2019. Disponível em: <https://securelist.com/ddos-report-q1-2019/90792>. Citado na página 3.

KIM, D.; SOLOMON, M. G. **Fundamentos de segurança de sistemas de informação**. Rio de Janeiro, RJ: LTC, 2014. ISBN 9788521625070. Citado 3 vezes nas páginas 2, 6 e 7.

KUMAR, U.; BORGOHAIN, T.; SANYAL, S. Comparative analysis of cryptography library in iot. **arXiv preprint arXiv:1504.04306**, 2015. Citado na página 15.

LEDERER, C. et al. Energy-efficient implementation of ecdh key exchange for wireless sensor networks. In: SPRINGER. **IFIP International Workshop on Information Security Theory and Practices**. [S.l.], 2009. p. 112–127. Citado na página 13.

LIU, A.; NING, P. A configurable library for elliptic curve cryptography in wireless sensor networks 2008, washington, dc, usa. **IEEE Computer Society**, 2008. Citado na página 16.

LUNARDI, R. C. et al. Distributed access control on iot ledger-based architecture. **IEEE Xplore**, 2018. Citado 4 vezes nas páginas 2, 3, 16 e 25.

MAHMOUD, R. et al. Internet of things (iot) security: Current status, challenges and prospective measures. In: IEEE. **2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)**. [S.l.], 2015. p. 336–341. Citado na página 1.

MARCONI, M. de A.; LAKATOS, E. M. **Fundamentos de Metodologia Científica**. 5. ed. São Paulo: Atlas, 2003. Citado na página 19.

MAURER, U. M.; WOLF, S. The diffie–hellman protocol. **Designs, Codes and Cryptography**, Springer, v. 19, n. 2-3, p. 147–171, 2000. Citado na página 11.

MENEZES, A. J.; OORSCHOT, P. C. V.; VANSTONE, S. A. **Handbook of applied cryptography**. Boca Raton: CRC press, 1996. Citado 2 vezes nas páginas 8 e 10.

MOORE, K.; BARNES, R.; TSCHOFENIG, H. **Best Current Practices for Securing Internet of Things (IoT) Devices**. 2017. Acessado em 23/10/2018. Disponível em: <https://tools.ietf.org/html/draft-moore-iot-security-bcp-01>. Citado na página 3.

NETWORKS, A. **Botnets IoT: o lado escuro do código aberto**. 2018. Acessado em 26/06/2019. Disponível em: <https://br.arbornetworks.com/asert-blog/botnets-iot-o-lado-escuro-do-codigo-aberto/>. Citado na página 3.

NIC.BR. **A Internet das coisas, explicada pelo NIC.br**. 2014. Acessado em 26/06/2019. Disponível em: <https://www.youtube.com/watch?v=jlkvzcG1UMk>. Citado na página 1.

NIST. **NIST Releases SHA-3 Cryptographic Hash Standard**. 2015. Disponível em: <https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard>. Citado na página 14.

NIST. **An Introduction to Information Security**. 2017. Citado na página 6.

PEREIRA, G. C. C. F. et al. Performance evaluation of cryptographic algorithms over iot platforms and operating systems. **Security and Communication Networks**, v. 2017, n. 2046735, 2017. Acessado em 23/10/2018. Disponível em: <https://www.hindawi.com/journals/scn/2017/2046735/>. Citado 3 vezes nas páginas 3, 16 e 19.

PUTMAN, T. **ECDH-based Authentication using Pre-Shared Asymmetric Keypairs for (Datagram) Transport Layer Security ((D)TLS) Protocol version 1.2**. 2017. Acessado em 09/10/2019. Disponível em: <https://tools.ietf.org/id/draft-putman-tls-preshared-ecdh-00.html>. Citado na página 17.

RABIAH, A. B. et al. A lightweight authentication and key exchange protocol for iot. 2018. Citado na página 2.

RAO, M.; NEWE, T.; GROUT, I. Secure hash algorithm-3 (sha-3) implementation on xilinx fpgas, suitable for iot applications. In: **8th International Conference on Sensing Technology (ICST 2014)**. [S.l.: s.n.], 2014. Citado na página 2.

SALMAN, T.; JAIN, R. A survey of protocols and standards for internet of things. **arXiv preprint arXiv:1903.11549**, 2019. Citado na página 4.

SANTOS, B. P. et al. Internet das coisas: da teoria à prática. **Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos**, 2016. Citado na página 1.

SILVA, P. T.; CARVALHO, H.; TORRES, C. B. **Segurança dos Sistemas de Informação**: Gestão estratégica da segurança empresarial. Lisboa: Centro Atlântico, 2003. Citado na página 7.

SKLAVOS, N.; ZAHARAKIS, I. D. Cryptography and security in internet of things (iots): Models, schemes, and implementations. **2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)**, 2016. Citado na página 1.

STALLINGS, W. **Criptografia e segurança de redes**: princípios e práticas. 6. ed. São Paulo: Pearson, 2015. Citado 7 vezes nas páginas 7, 8, 9, 10, 11, 13 e 14.

STONE-GROSS, B. et al. Your botnet is my botnet: analysis of a botnet takeover. In: ACM. **Proceedings of the 16th ACM conference on Computer and communications security**. [S.l.], 2009. p. 635–647. Citado na página 4.

UPADHYAY, S. Ongoing challenges and research opportunities in internet of things (iot). In: **International Journal of Engineering Technologies and Management Research**. [S.l.: s.n.], 2018. p. 216–222. Citado na página 1.

WEATHERLEY, R. **Arduino Cryptography Library**. 2018. Acessado em 01/07/2019. Disponível em: <https://rweather.github.io/arduinolibs/crypto.html>. Citado 2 vezes nas páginas 15 e 25.

WEISZFLOG, W. **Michaelis Moderno Dicionário da Língua Portuguesa**. Editora Melhoramentos, 2015. Acessado em 23/10/2018. Disponível em: <https://michaelis.uol.com.br/moderno-portugues/>. Citado na página 7.

WOLFSSL. **WolfSSL Embedded SSL/TLS Library**. 2019. Acessado em 11/11/2019. Disponível em: <https://www.wolfssl.com/products/wolfssl/>. Citado na página 15.

XU, T.; WENDT, J. B.; POTKONJAK, M. Security of iot systems: Design challenges and opportunities. In: IEEE PRESS. **Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design**. [S.l.], 2014. p. 417–423. Citado na página 2.

ZORZO, A. F. et al. Dependable iot using blockchain-based technology. In: IEEE. **2018 Eighth Latin-American Symposium on Dependable Computing (LADC)**. [S.l.], 2018. p. 1–9. Citado na página 5.