FRANCISCONE LUIZ DE ALMEIDA JÚNIOR

AVALIAÇÃO DA CRIPTOGRAFIA AES EM DISPOSITIVOS IOT USANDO O PROTOCOLO MQTT

LAVRAS - MG

2020

FRANCISCONE LUIZ DE ALMEIDA JÚNIOR

AVALIAÇÃO DA CRIPTOGRAFIA AES EM DISPOSITIVOS IOT USANDO O PROTOCOLO MQTT

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Bruno de Abreu Silva
Orientador

LAVRAS - MG

2020

Ficha Catalográfica preparada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo próprio autor.

Almeida Júnior, Franciscone Luiz de

Avaliação da criptografia AES em dispositivos IoT usando o protocolo MQTT / Franciscone Luiz de Almeida Júnior. – 2020.

87 p. : il.

Orientador: Prof. Dr. Bruno de Abreu Silva.

Monografia (Graduação)–Universidade Federal de Lavras, 2020.

Bibliografia.

1. Criptografia AES. 2. MQTT. 3. IoT. I. Silva, Bruno de Abreu. II. Título.

FRANCISCONE LUIZ DE ALMEIDA JÚNIOR

AVALIAÇÃO DA CRIPTOGRAFIA AES EM DISPOSITIVOS IOT USANDO O PROTOCOLO MQTT
AES CRYPTOGRAPHY EVALUATION IN IOT DEVICES USING MQTT PROTOCOL

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 4 de Setembro de 2020.

Prof. Dr. Demóstenes Zegarra Rodríguez	UFLA
Prof. Dr. Hermes Pimenta de Moraes Júnior	UFLA

Prof. Dr. Bruno de Abreu Silva
Orientador

LAVRAS - MG
2020

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais Franciscone e Débora por sempre me incentivarem nos estudos, por todo apoio nesses anos de faculdade e por estarem ao meu lado durante toda essa jornada. Agradeço aos meus avós, que também me incentivaram a buscar meus sonhos e lutar por aquilo que desejo. Agradeço, também, a todos os membros de minha família que, eventualmente, participaram de alguma forma nesse processo, seja pelo apoio emocional, financeiro, e até mesmo pela paciência comigo nos momentos mais conturbados e difíceis. Agradeço ao Professor Demóstenes Zegarra Rodríguez, que foi o primeiro professor a acreditar em meu potencial, e oferecer, a um calouro, oportunidades incríveis de imersão científica através dos projetos que desenvolvemos. Em todos esses anos de IC, só tenho a agradecer por todo o conhecimento adquirido, todo o trabalho que realizamos juntos e pelas lições aprendidas. Agradeço também ao Professor Bruno de Abreu Silva, que teve grande participação na minha jornada acadêmica, através do núcleo de estudos que fundamos juntos com nossos colegas e que, posteriormente, se tornou uma Empresa Júnior, e até mesmo neste último trabalho que acabamos de concluir. Agradeço a meus colegas da Emakers Júnior, pela confiança, por acreditarem nas minhas habilidades, pela paciência em me ouvir e por todo aprendizado durante aquele processo de fundação da empresa, que só me fez crescer como profissional. Agradeço a todos os amigos e professores que fizeram parte dessa jornada, seja aqui na UFLA ou no IPB durante o intercâmbio, um deles sendo meu amigo de várias aventuras e momentos, Lucas Rodrigues, por tudo que vivemos juntos em Portugal. Agradeço a meus colegas de república, que muitas vezes foram também colegas de trabalho nas atividades acadêmicas, pelo apoio, paciência e pelo ótimo trabalho em equipe que realizamos juntos. Agradeço a minha amiga Marina Salém, que esteve presente durante quase toda minha formação acadêmica e que me aconselhou, escutou, apoiou e aturou nos momentos difíceis da graduação. Por fim, agradeço à Universidade Federal de Lavras pela oportunidade e pelo crescimento profissional, e da qual tenho muito orgulho.

"Make everything as simple as possible, but not simpler."
(Albert Einstein)

RESUMO

Com o advento da Internet e a necessidade de automação de processos, o uso de dispositivos embarcados conectados em rede tem crescido cada vez mais. Sabe-se que no fim do primeiro semestre de 2020 o número de dispositivos IoT (*Ìnternet of Things*) é estimado em 31 bilhões. Em uma rede tão complexa e diversa quanto a IoT, na qual as implementações podem variar desde a automação de processos industriais até a criação de casas inteligentes, é necessário um protocolo de comunicação que atenda a essa diversidade e, dentre as muitas opções, uma das mais citadas e referenciadas na literatura é o *Message Queuing Telemetry Transport* (MQTT). O MQTT é um protocolo de camada de aplicação otimizado para sistemas embarcados e que busca ser simples, *lightweight*, e com baixo consumo de energia. No entanto, essa adequação para se atender a diversidade de dispositivos coloca em risco alguns princípios de segurança, visto que o protocolo sozinho não possui mecanismos nativos de segurança, como a criptografia para se ocultar os dados. Por isso, neste trabalho implementou-se a criptografia AES no *payload*, o que garante princípios de segurança e preserva as características que tornam o protocolo ideal para dispositivos com hardware limitado. Após essa implementação, foram feitas análises qualitativas, em termos dos princípios de segurança, e foram acrescentadas análises quantitativas (que se tratam do uso de recursos de hardware, como uso da memória e tempo de execução dos métodos implementados), que não foram encontradas na literatura em que este trabalho se baseia. Por fim, os resultados dessas métricas de análise são comparados com o objetivo de se averiguar a viabilidade e o comportamento de alguns dos sistemas embarcados mais usados no contexto de IoT em cada uma das implementações feitas com o AES e os modos de operação de cifra. Isso permitiu concluir que em dispositivos como o Arduíno Uno, no qual o hardware é bastante limitado, é possível se aplicar a criptografia do *payload*, com o modo de operação mais robusto que é o CTR, e ainda se ter espaço de memória para programas mais complexos e com diversos sensores. Além disso, deve-se destacar que nos outros dispositivos utilizados (Raspberry Pi 3 Modelo B e ESP8266), essa implementação pode não ser tão interessante visto que as placas possuem recursos de hardware suficientes para se implementar técnicas mais robustas como o SSL ou o TLS.

Palavras-chave: Criptografia AES. MQTT. IoT.

ABSTRACT

As the Internet grows, the need for process automation and the use of embedded devices connected to the network has also grown. It is known that late in the first semester of 2020 the number of IoT (Internet of Things) nodes is estimated at 31 billion. In a network as complex and diverse as the IoT, implementations can vary from the automation of industrial processes to the creation of smart homes, therefore a communication protocol that suits this diversity is needed, and among the many options, one of most quoted and referenced in the literature is the MQTT. Message Queuing Telemetry Transport (MQTT) is an application layer protocol optimized for embedded systems, which aims to be simple, lightweight, and with a low power consumption. However, this adequation to fits the diversity of devices puts at risk some security principles, since the protocol alone does not have any native security mechanisms, such as encryption of data. Therefore, this work aims in implementing AES encryption in the payload, which will guarantee security principles and preserve the characteristics that make the protocol ideal for devices with limited hardware. After the implementation, qualitative analyzes are carried out, in terms of the security principles, and quantitative analyzes are added (in regards of hardware usage, such as memory and execution time to each of the implemented methods), which were not found in the literature that this work is based on. Finally, the results of these metrics are compared in order to ascertain the feasibility and behavior of each implementation with the AES algorithm and the cipher operation modes in some of the most used embedded systems in the context of IoT. This allowed the conclusion that in devices such as Arduíno Uno, in which the hardware is very limited, it is possible to apply the encryption of payload, with CTR that is the most robust operation mode, and still have memory space for more complex programs with several sensors. In addition, it should be noted that in the other devices used (Raspberry Pi 3 Model B and ESP8266), this implementation may not be as interesting as the boards have sufficient hardware resources to implement robust techniques such as SSL or TLS.

Keywords: AES Encryption. MQTT. IoT

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Antes presentes apenas em filmes de ficção científica, diversas áreas e aplicações deixaram de ser apenas imaginação de diretores de cinema e passaram a ser algo muito próximo da realidade. Como exemplo, podemos citar: Casas Inteligentes (*Smart Houses*); automação industrial; relógios inteligentes; carros autônomos, que são capazes, por exemplo, de trocar informações em tempo real com os dispositivos móveis e obter qual o melhor trajeto a ser seguido com base no trânsito atual; trânsito inteligente com a finalidade de evitar congestionamentos e acidentes e; Cidades Inteligentes (*Smart Cities*). Diante desse cenário, um dos principais conceitos envolvidos é o de Internet das Coisas (*Internet of Things - IoT*), em que, segundo Alecrim (2017), "a ideia por trás do conceito é a de que todos equipamentos podem estar conectados à internet e, assim, facilitar a vida das pessoas no seu dia a dia". Tais dispositivos, ao trocar informações por meio da sua conexão, são capazes de realizar sensoriamento, tomada de decisão e atuação no ambiente de forma automatizada, com um mínimo de intervenção humana ou, muitas vezes, totalmente sem intervenção humana.

Com a popularização de recursos tecnológicos, incluindo a Internet, hoje é possível que uma pessoa crie sua própria casa inteligente, com a habilidade de controlar o sistema elétrico por comandos ou até mesmo com o uso técnicas de Inteligência Artificial (IA) e reconhecimento de padrões. Além disso, o uso da IoT pode ser ainda mais amplo e não se limitar somente às áreas da Computação. O conceito de IoT é baseado na ideia de fusão do mundo real com o mundo digital, fazendo com que os indivíduos estejam em constante comunicação e interação com outras pessoas e objetos. A IoT possui funções de reconhecimento inteligente, localização, rastreamento e gerenciamento dos diversos dispositivos, trocando informações a todo o momento (MORAIS, 2018).

Em conjunto com as demais Ciências, os conceitos de IoT são usados, por exemplo, para coletar informações sobre o clima e o ambiente a fim de prevenir

a sociedade de desastres naturais; auxiliar na criação de dispositivos autônomos que sejam ativados por voz para assistir pessoas com alguma debilidade motora; e ajudar no controle hídrico, aumentando a eficiência do controle de qualidade, escassez e o desperdício de água.

No entanto, existe uma questão relevante na área de Internet das Coisas, que é o compromisso entre segurança e eficiência, uma vez que técnicas mais robustas de segurança normalmente exigem mais do hardware disponível. Nos trabalhos de Andy, Rahardjo e Hanindhito (2017), Naik e Maral (2017), Jadhav, Kulkarni e Swamy (2017) e Mathews e Gondkar (2019) são melhor descritos esses problemas de segurança em IoT, que serão abordados detalhadamente no Capítulo 2, Seção 2.4. É sabido que, no âmbito de IoT, a variedade de dispositivos é muito grande. Logo, podemos ter dispositivos com mais recursos de *hardware* e outros com menos recursos de *hardware* - entendidos como os componentes que compõem suas arquiteturas. Assim, uma grande preocupação é ter a segurança nesses dispositivos, porém mantendo as características de eficiência. No cenário de IoT e neste trabalho em específico, a eficiência do dispositivo foi considerada como um conjunto das métricas de desempenho, uso de recursos e confiabilidade.

Quando se fala em segurança computacional, uma das áreas da Ciência da Computação que vem em mente é a ocultação da informação, ou criptografia. A criptografia estuda os métodos para codificar uma mensagem de modo que só o seu destinatário legítimo consiga interpretá-la (COUTINHO, 2014, p. 1). Atualmente, muitos dos sistemas informatizados que possuem conexão com a Internet possuem alguma forma de criptografia, seja qual for o protocolo usado (HTTPS - *Hyper Text Tranfer Protocol Secure*, VPN - *Virtual Private Network*, entre outros). No entanto, protocolos como o HTTPS que implementam o *Security Sockets Layer - SSL* na sua camada de segurança não são opções viáveis para dispositivos que possuem *hardware* limitado devido a complexidade dos algoritmos. Para além disso, segundo Mathews e Gondkar (2019), o protocolo MQTT (sigla para *Mes-*

sage Queuing Telemetry Transport), amplamente usado em redes IoT, não possui métodos confiáveis de autorização ou encriptação dos dados e os pacotes de controle não provém nenhum mecanismo para identificar o *payload* criptografado.

Na pilha de protocolos TCP/IP, tem-se um gama muito grande de protocolos de camada de aplicação, o que aumenta as opções de implementação, mas pode gerar dúvidas na escolha do protocolo a ser usado na troca de mensagens. Em se tratando de sistemas embarcados, os protocolos comumente usados segundo Jadhav, Kulkarni e Swamy (2017) são:

- MQTT: é o protocolo de camada de aplicação mais usado em sistemas embarcados. Esse é um protocolo *lightweight* que trabalha no modelo *publish/-subscribe*, é também otimizado para coleções de dados centralizadas e análises conectando dispositivos inteligente e móveis a uma aplicação sendo executada em um centro de dados. É recomendado quando se tem dispositivos com memória limitada e de baixa largura de banda na rede (JADHAV; KULKARNI; SWAMY, 2017, p. 2);

- AMQP (*Advanced Message Queuing Protocol*): Esse é um dos protocolos de camada de aplicação para troca de mensagens baseada em *middleware* que segue o conceito de *open standard*. As principais características desse protocolo são: pode ser orientado a mensagem, enfileiramento, comutação, confiabilidade e segurança. Além disso suporta também modelos ponto a ponto, *publish/subscribe* e roteamento ou comutação (JADHAV; KULKARNI; SWAMY, 2017, p. 2);

- CoAP (*Constrained Application Protocol*) - Esse protocolo de camada de aplicação é destinado ao uso em ambiente, recursos e rede restritos. É um protocolo de transferência da web e usa um modelo de solicitação-resposta. De acordo com Jadhav, Kulkarni e Swamy (2017, p. 2), foi projetado para cooperar e trabalhar junto com o HTTP;

- HTTP (*Hyper Text Transfer Protocol*): é um protocolo que permite a obtenção de recursos, tais como documentos HTML. É a base de qualquer troca de dados na Web e um protocolo cliente-servidor, o que significa que as requisições são iniciadas pelo destinatário, geralmente um navegador da Web. Clientes e servidores se comunicam trocando mensagens individuais (em oposição a um fluxo de dados). As mensagens enviadas pelo cliente, geralmente um navegador da Web, são chamadas de solicitações (*requests*), ou também requisições, e as mensagens enviadas pelo servidor como resposta são chamadas de respostas (*responses*) (MOZILLA COMMUNITY, 2020).

Com isso, este trabalho visa comparar o uso do AES, dentre as diversas formas de criptografia dos dados presentes na literatura, em diferentes dispositivos para sistemas embarcados (com diferentes configurações de *hardware*), com o uso do protocolo MQTT para transmissão de dados na rede, devido a sua baixa utilização de recursos em sistemas embarcados.

1.1 Justificativa

Após a leitura e análise da literatura encontrada sobre o tema, percebeu-se uma lacuna referente a análise quantitativa do processo de criptografia do MQTT e da criptografia aplicada a apenas o *payload*. Isso ocorre porque além do protocolo não prover nenhuma opção de criptografia de forma nativa, as demais técnicas disponíveis como o TLS/SSL são técnicas mais genéricas e que são aplicadas à camada de aplicação da pilha TCP/IP, por isso seriam contextos de avaliação diferentes. Em Mathews e Gondkar (2019), os autores descrevem uma forma de criptografia para o MQTT aplicada ao pacote de controle, sugerindo assim uma alteração na definição do mesmo. Além disso, os autores também descrevem como o método proposto visa manter a leveza do MQTT, embora não sejam fornecidos valores quantitativos. Neste trabalho, buscou-se então, aplicar a criptografia ao

payload do MQTT e avaliar o impacto dessa técnica no processo de comunicação com análises quantitativas e qualitativas.

1.2 Objetivo

Avaliar a viabilidade, o impacto, vantagens e desvantagens da utilização de algoritmos de criptografia, como AES com diferentes modos de operação de cifra, nas placas NodeMCU, Arduíno UNO e Raspberry Pi, comumente usadas em aplicações IoT.

1.2.1 Objetivos Específicos

- Definir quais bibliotecas são mais adequadas para cada uma das placas utilizadas com base na sua abrangência na comunidade, na linguagem utilizada, na licença de software aplicada e na frequência de manutenção dos desenvolvedores;
- Desenvolver, com base em conceitos da criptografia AES e seus modos de operações, soluções que implementem alguma forma de segurança no *payload* do MQTT;
- Preparar um ambiente controlado que permita a realização de ataques para análise futura;
- Definir, a partir dos resultados, qual a aplicabilidade de cada uma das implementações.

1.3 Organização do Trabalho

O restante dos Capítulos desta monografia está organizado da seguinte forma: no Capítulo 2 são apresentados os conceitos teóricos principais em cada subárea do conhecimento envolvida neste trabalho. No Capítulo 3, são descritos

os métodos e as técnicas usadas para desenvolver o ambiente de testes e como é a coleta de resultados nesse cenário. No Capítulo 4, os conceitos teóricos e técnicas referenciados são combinados para apresentar análises quantitativa e qualitativa dos testes. Por fim, o Capítulo 5 apresenta a contribuição do trabalho, além de suas limitações e trabalhos futuros.

2 REFERENCIAL TEÓRICO

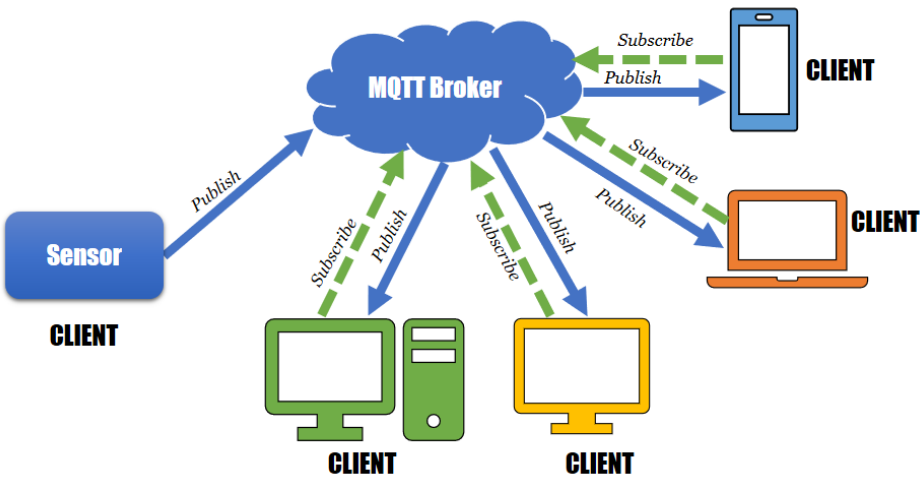
Este Capítulo busca apresentar conceitos já conhecidos e estudados na literatura com a finalidade de fornecer o embasamento teórico e científico para todos os métodos, técnicas, conceitos e padrões que são utilizados neste trabalho. De modo geral, primeiro será feita um abordagem geral sobre o MQTT com enfoque em seu funcionamento, na estrutura dos pacotes e na arquitetura de comunicação. Posteriormente, serão revisados alguns tópicos principais sobre segurança que tratam especificamente sobre o escopo deste trabalho, visto que segurança da informação é um área grande e complexa. Por fim, será feita uma abordagem sobre as técnicas de criptografia usadas, bem como suas características e processo evolutivo.

2.1 Protocolo de aplicação MQTT

Message Queue Telemetry Transport (MQTT) é um protocolo que segue a arquitetura Cliente-Servidor e usa a comunicação do tipo *publish/subscribe* (OASIS COMMITTEE SPECIFICATION DRAFT, 2014), ou seja, toda e qualquer informação enviada por um dispositivo na rede IoT (também chamado de nó IoT) deve ser feita através da publicação de uma mensagem em um tópico, e consequentemente um nó que deseja receber informações de outro deve se inscrever no mesmo tópico no qual as informações são publicadas.

O MQTT foi desenhado para ser leve, aberto e de simples implementação. Essas definições o fazem de uso ideal para diversas situações, incluindo ambientes restritos como em comunicação M2M (*Machine to Machine*) e também no contexto de IoT, onde recursos de hardware são bastante limitados e a largura de banda é reduzida. O protocolo é implementado sobre a pilha TCP/IP ou outros protocolos de rede que incluam ordenação, controle de perda e conexão bidirecional (OASIS COMMITTEE SPECIFICATION DRAFT, 2014). Além das características supracitadas, o MQTT possui outros recursos, tais como:

Figura 2.1 – Arquitetura do MQTT.



Fonte: Kpizingui (2017).

- A distribuição de mensagens de um-para-muitos e a desacoplação de aplicações;
- O transporte de mensagem que é independente de conteúdo;
- Três níveis de qualidade de serviço (*QoS - Quality of Service*) para entrega de mensagens:
  - **"No máximo uma vez":** as mensagens são entregues usando o melhor esforço do ambiente de operação. Nesse modo de operação, as perdas têm impacto pequeno se observada a aplicação como um todo;
  - **"No mínimo uma vez":** as mensagens devem ter garantia de entrega, mas duplicatas podem ocorrer;
  - **"Exatamente uma vez":** as mensagens devem ser enviadas e processadas apenas uma vez, visto que a ocorrência de duplicações pode ocorrer em falhas do sistema.
- Redução do tráfego de rede;

- Um mecanismo de notificação de partes interessadas no caso de desconexões inesperadas;

- A organização e o controle dos tópicos de mensagens de forma hierárquica usando os caracteres reservados, também chamados de *wildcards*.

2.1.1 Estrutura de tópicos do MQTT

Como dito anteriormente, o MQTT é um protocolo que trabalha com a arquitetura *publish/subscribe* (abreviamos como *pub/sub* neste trabalho) na troca de informações. O processo em si é bastante simples, e funciona com um servidor, mais conhecido como *broker*, que recebe as mensagens em cada um dos tópicos e replica para todos os nós que realizaram a inscrição no tópico em questão. Para questões de organização e estruturação da informação, o MQTT permite a separação dos tópicos de forma hierárquica com o uso do caractere especial '/', o que resulta em uma estrutura em forma de árvore (**Figura 2.2**).

Como pode ser observado na Figura 2.2, o tópico raiz da estrutura do MQTT é chamado de "casa"e, transformando o exemplo anterior para tópicos reais usados por aplicações, teríamos:

- casa/sala/presenca;

- casa/cozinha/presenca;

- casa/cozinha/gas;

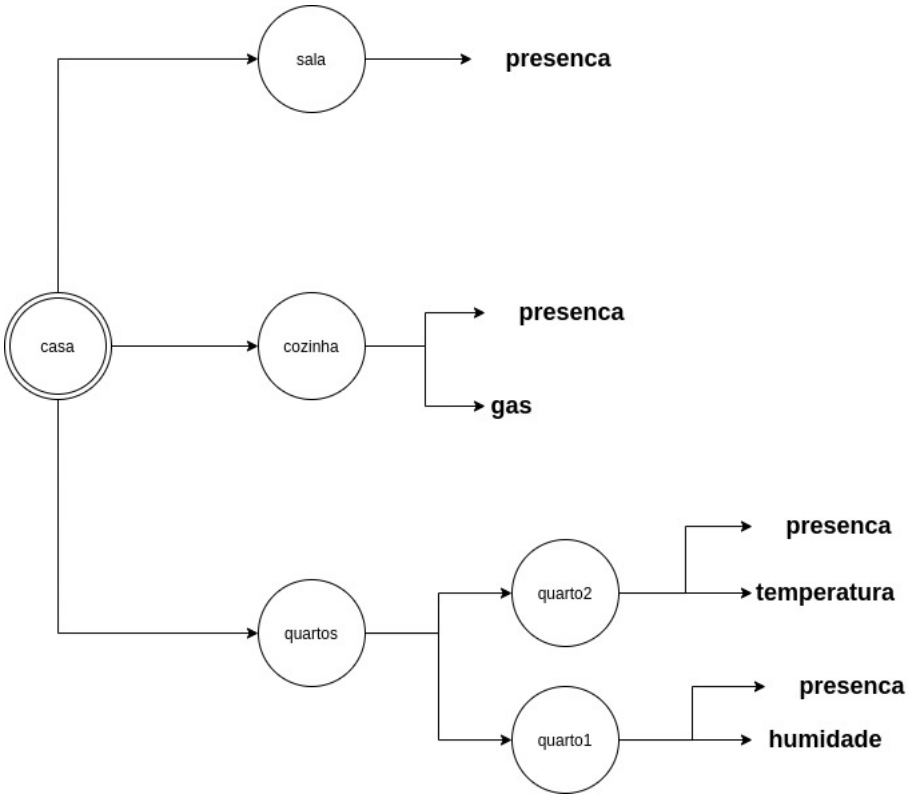
- casa/quartos/quarto2/presenca;

- casa/quartos/quarto2/temperatura;

- casa/quartos/quarto1/presenca;

- casa/quartos/quarto1/humidade.

Figura 2.2 – Árvore de tópicos do MQTT.



Fonte: Do Autor (2020).

Apesar de fácil organização e estruturação dos tópicos, essa abordagem pode se tornar verbosa e dificultar a inscrição de dispositivos em uma rede MQTT. Para evitar esses problemas, alguns caracteres especiais podem ser usados para facilitar a inscrição a tópicos. O caractere '+' implica todos os assuntos em um mesmo nível de hierarquia, enquanto o caractere '#' define todos os assuntos da mesma categoria e suas subcategorias restantes (ANDRADE, 2017). Alguns exemplos relacionados a Figura 2.2 podem ser citados:

- "casa/+/presenca": Essa inscrição indicaria pra o *broker* que o nó IoT deseja se inscrever em todos os tópicos que no nível 3 tenha a palavra "presenca", independente de qual seja o valor no segundo nível da hierarquia. Nesse

cenário, seria o mesmo que se inscrever separadamente em "casa/sala/presenca", "casa/cozinha/presenca";

- "casa/quartos/#": Nesse caso, a inscrição indica que o nó deseja se inscrever em qualquer tópico que siga corretamente a hierarquia até o nível 3, porém os níveis ou valores posteriores são irrelevantes, logo, todos os valores devem ser incluídos na inscrição. Dessa forma, a inscrição seria equivalente a: "casa/quartos/quarto1/presenca", "casa/quartos/quarto2/temperatura", "casa/quartos/quarto1/presenca", "casa/quartos/quarto1/humidade". Um ponto de observação importante é que o uso dessa *wildcard* é restrito ao fim da *string* de inscrição.

2.1.2 Arquitetura do Protocolo MQTT

O funcionamento do MQTT se baseia essencialmente no uso do Pacotes de Controle (*Control Packets*) de uma forma muito bem definida. Um pacote de controle consiste em até três partes, sempre na mesma ordem (OASIS COMMITTEE SPECIFICATION DRAFT, 2014):

- Cabeçalho Fixo;
- Cabeçalho Variável;
- *Payload*.

O cabeçalho fixo possui um campo de quatro bits para identificar o tipo do pacote e mais um campo de quatro bits para uso de *flags* de controle (que são usadas para QoS, identificar conexões e desconexões, entre outras). O cabeçalho variável e o *payload* dependem do tipo do pacote, não estão presentes em todos, no entanto, a presença de um *payload* requer necessariamente um cabeçalho variável. Os pacotes de controle são definidos em 14 tipos, os quais são:

- CONNECT: é o primeiro pacote a ser enviado para se requisitar a conexão do cliente com o servidor. Nesse pacote, são enviadas as configurações de

conexão (como a QoS, ou a mensagem de Will), bem como as credenciais de conexão caso necessário;

- CONNACK: é uma resposta do *broker* ao pacote de *connect* para identificar que a conexão foi efetuada com sucesso;

- PUBLISH: é o pacote usado para a comunicação em si. Esse pacote pode ser enviado do cliente ao servidor para enviar uma mensagem aos demais nós e do servidor para os clientes a fim de replicar uma mensagem;

- PUBACK: é um pacote de retorno para a publicação de uma mensagem com o QoS de 1. A publicação é reconhecida pelo servidor;

- PUBREC: É o segundo pacote enviado pelo servidor após a publicação de uma mensagem com QoS de 2. Nesse caso, o servidor reconhece que a publicação foi recebida;

- PUBREL: é o terceiro pacote enviado pelo servidor para uma QoS de 2. O pacote fala que a publicação foi liberada, pois a confirmação final foi efetuada;

- PUBCOMP: quarto e último pacote enviado em um QoS de 2. O receptor, após enviar essa resposta ao *broker*, trata toda requisição subsequente como uma nova comunicação;

- SUBSCRIBE: pacote usado pelo cliente para criar inscrições em um ou mais tópicos do servidor;

- SUBACK: é enviado pelo servidor para confirmar ao cliente que sua inscrição foi bem sucedida no(s) tópico(s) em questão;

- UNSUBSCRIBE: o cliente envia uma requisição para não receber mais mensagens de um mais tópicos, sendo assim ele se desinscreve do assunto;

- UNSUBACK: o servidor envia esse pacote para confirmar ao cliente que a remoção de inscrição foi recebida e processada com sucesso;

- PINGREQ: em caso de ausência de informações a serem enviadas pelo cliente, este pode enviar um pacote desse tipo a fim de informar ao *broker* que ainda está na rede, porém não possui nenhum outro pacote de controle a ser enviado. Em caso de ociosidade esse pacote é usado para manter a conexão ativa;

- PINGRESP: resposta do servidor ao PINGREQ;

- DISCONNECT: o cliente solicita a desconexão do servidor.

2.2 Tópicos em segurança

Nesta Seção, serão descritos algumas técnicas e tópicos em segurança da informação. Dessa forma, serão definidos metodologias e algoritmos, além de como esses trabalham juntos para construir sistemas informatizados ainda mais seguros.

Na atualidade, com o avanço de tecnologias de informação, surgiu uma diversidade grande de ferramentas voltadas a essa prática de se atacar um dispositivo computacional. Essa facilitação fez com que a necessidade de conhecimentos avançados em tópicos de segurança como redes e arquitetura não fossem mais tão necessários para realizar os ataques. Então, vários curiosos podem, com um computador, simplesmente usar ferramentas e realizar ataques mais simples atualmente.

Quando se fala de segurança, um dos primeiros tópicos a serem levantados se refere aos pilares que se deve garantir a um sistema efetivamente seguro. Segundo uma das organizações que regulamentam e definem os padrões para segurança computacional, a OWASP FOUNDATION (2020), para que uma aplicação

seja realmente considerada segura, deve ser desenhada com os três pilares da segurança da informação em mente:

- Confidencialidade: permitir o acesso aos dados apenas a usuários autorizados;
- Integridade: garantir que os dados não foram modificados ou comprometidos por usuários não autorizados;
- Disponibilidade: a aplicação deve garantir que o sistema e os dados estão disponíveis para usuários autorizados sempre que eles precisarem.

Hossain e Skjellum (2017 apud ANDRADE, 2017) ainda descrevem que, alguns outros critérios devem ser validados:

- anonimato: em algumas aplicações é importante garantir o anonimato dos usuários;
- se o dado é recente: trata sobre a atualização dos dados em tempo real;
- controle de acesso: garantir políticas para controle de diferentes níveis de acesso;
- operabilidade: a variedade presente em redes IoT não pode limitar as funcionalidades da aplicação;
- escalabilidade: a aplicação deve ser expansível se necessário;
- eficiente no uso da memória: garantir adequações ao *hardware*, dos sistemas embarcados;
- sobrecarga mínima de computação e comunicação: os algoritmos de segurança devem consumir o mínimo possível de recursos.

2.2.1 Segurança para IoT

Todo dispositivo conectado pode ser uma fonte de dados e esses dados podem ser usados de diversas formas com o intuito de facilitar a vida e promover uma nova evolução tecnológica, essa é a essência da IoT. Nesse cenário, todos os nós IoT conectados em rede podem ser fontes de informações, logo, a variedade de dispositivos com poderes computacionais diversos é muito grande, o que torna difícil a implementação de protocolos de segurança em comum para esses dispositivos. Em se tratando da comunicação desses dispositivos, observa-se diversas opções de protocolos de camada de aplicação e cada um tem suas vantagens e desvantagens dentro de um cenário em específico, bem como suas vulnerabilidades (entende-se por vulnerabilidade uma falha de segurança que poder ser explorada e comprometer a segurança de um sistema).

Como já citado no Capítulo 1 deste trabalho, um dos problemas maiores da segurança de uma rede IoT é a preocupação de se manter a comunicação segura, considerando fatores como a variedade de dispositivos e *hardware* limitado em alguns nós e a necessidade de comunicação rápida que garanta a autenticidade e integridade das informações. Com essas considerações em mente, uma possibilidade é o uso de criptografia RSA, que é considerada relativamente custosa em termos de uso de recursos computacionais, em dispositivos como um microcontrolador que, em geral, possui menos recursos que outros sistemas computacionais como *desktops*. Além disso, pode-se pensar no quanto a adição de criptografia dos dados acrescentaria no tempo de processamento e se isso seria de grande impacto em um sistema que requer comunicação cujo tempo gasto é um fator importante. Essas considerações são melhor descritas nos tópicos seguintes, quando são descritos os tipos de ataques abordados no trabalho e como eles afetam a integridade do sistema IoT como um todo.

2.2.2 Vulnerabilidades do MQTT

O MQTT é um protocolo que trabalha na camada de aplicação e foi desenvolvido com características adequadas para se trabalhar com baixo consumo de energia, simplicidade e recursos de hardware limitados. No entanto, isso se torna um problema, em se tratando da segurança no processo de comunicação. A arquitetura do MQTT por si permite apenas dois tipos de recursos de segurança: a **autenticação** e **autorização**.

A **autenticação** é o processo que indica que o usuário é realmente quem ele indica ser (ANDRADE, 2017). Para restringir a conexão com o *broker*, o protocolo permite a configuração de credenciais de acesso, restringindo assim o acesso há somente clientes autorizados. As credenciais são enviadas como parte do cabeçalho do pacote CONNECT e não são obrigatórias, ou seja, a sua necessidade é definida de acordo com a configuração do servidor. No entanto, essa opção não torna o processo de conexão efetivamente seguro. Como retratado por (ANDRADE, 2017), quando usados, o usuário e senha são enviados em texto claro (ou seja, sem nenhum mecanismo de criptografia), o que permite que algum indivíduo mal intencionado visualize a informação no pacote de rede capturado.

Segundo Andrade (2017), a **autorização** define que, mesmo que um usuário tenha as credenciais que o permitam conectar-se ao servidor, suas ações e seu acesso ao tráfego de rede podem ser limitados. Para isso, são necessários dois mecanismos, um que identifique o usuário (no caso o *Client Id* - Identificador do Cliente - que é enviado no pacote CONNECT) e outros que definem as políticas de acesso (no caso do MQTT são as Listas de Controle de Acesso - *ACL, Access Control List*). Essa técnica define qual cliente (ou nó) tem acesso a quais recursos e quais suas permissões naquele recurso. Por exemplo, podem ser definidas regras que só permitem a usuários específicos se inscreverem ou publicarem em um tópico em questão. Porém, a mesma vulnerabilidade citada na autorização pode ser encontrada na questão dos tópicos. Isso ocorre porque existem técnicas que per-

mitem o ataque a um conexão *Wireless* privada e, mesmo que a possibilidade de sucesso seja baixa, um atacante que conseguiu esse acesso pode analisar os dados e assim inferir quais acessos tem um usuário a um determinado recurso.

Por fim, o MQTT possui mais um recurso de segurança, o **SSL/TLS**, que não está necessariamente relacionado com o protocolo de aplicação em si, mas se faz disponível para ele devido a sua implementação sobre a pilha TCP/IP. Para entender melhor as funcionalidades do TLS/SSL, teremos a Seção 2.3, que trata sobre tópicos em criptografia e descreve melhor o seu funcionamento. Nesta Seção, o foco é definir por que o protocolo não é usado como o padrão de segurança do MQTT. Em linhas gerais, o TLS trabalha sobre o TCP/IP criando um canal de comunicação seguro, criptografado, com trocas de chaves simétricas usando o método de criptografia apresentado em Diffie e Hellman (1976) e que é assinado digitalmente para reconhecer a legitimidade dos interlocutores e, consequentemente, de suas mensagens. No entanto, esse processo em si consome mais recursos de *hardware* (como processamento e memória), além de aumentar consideravelmente o uso de banda e de recursos da rede. Esses fatores fazem com que algumas placas e microcontroladores não consigam trabalhar com o TLS, como é o caso do ATMega328P presente na placa Arduíno Uno. Isso torna inviável o seu uso em alguns cenários.

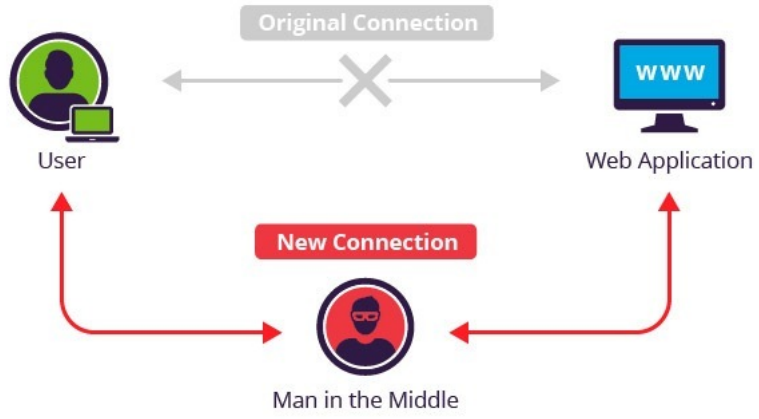
2.2.3 Ataque MITM

Embora bastante robusto e com diversas opções de ataque, o *Man-In-The-Middle, MITM*, é um ataque simples de ser executado. Como o próprio nome diz, o MITM é uma técnica em que o atacante se coloca no "meio"de um canal de comunicação, ou seja, é capaz de interceptar e modificar as mensagens trafegadas. Por exemplo, um cliente faz uma requisição para acessar um site A (não seguro) que está hospedado no servidor X. Nessa situação, se existe um indivíduo malicioso na mesma rede, este pode interceptar o tráfego e, além de capturar in-

formações importantes como credenciais de acesso, é possível também a alteração das informações em tempo de navegação.

Como descrito por Andrade (2017), esse tipo de ataque compromete alguns princípios de segurança, como a **Confidencialidade**, pois há a coleta indevida de informações privadas. **Integridade**, já que as mensagens podem ser modificadas, não garantindo sua veracidade. E **Disponibilidade**, uma vez que as mensagens são interceptadas e destruídas ou modificadas de uma forma que a comunicação seja cortada entre ambas as partes. A Figura 2.3 representa a estrutura do ataque. O *Man in the middle* é um ataque que possui diversas variantes, algumas

Figura 2.3 – Representação de um ataque MITM.



Fonte: Nunes (2017).

delas requerem que ambos usuários, legítimo e malicioso, estejam na mesma rede, como é o caso do *ARP Spoofing* (também conhecido como *ARP Cache Poisoning*) e da técnica de *Evil Twin* (na qual o atacante cria uma rede idêntica a do usuário a fim de enganá-lo). Como descrito em Nunes (2017), essa técnica também pode ter suas variantes que são executadas no lado do cliente, como o *DNS Spoofing* e *SQL Injection* (conhecido também como SQLi). Neste trabalho, no entanto, nosso foco será no *ARP Spoofing*, que é melhor descrito a seguir.

2.2.4 ARP Spoofing

O *ARP Spoofing* é uma técnica associada com ataques do tipo MITM, que consiste em "envenenar"a tabela ARP de um dispositivo de rede a fim de redirecionar pacotes para outro dispositivo. Ou seja, o atacante absorve as requisições informando à tabela ARP que o MAC da solicitação pertence ao seu *host*, a qual acredita que o *host* atacante seja o ponto final da informação (NUNES, 2017). Em termos mais técnicos, o atacante envia uma série de pacotes ARP para a vítima informando que o endereço MAC do *Gateway* está localizado, na verdade, no computador do atacante. Seguindo essa mesma ideia, um processo semelhante é feito para o *gateway*, no qual o atacante envia outro conjunto de pacotes ARP informando que o MAC do cliente é, na verdade, o MAC do seu *host*. Assim, ambas as tabelas ARP ficam com entradas inconsistentes e todo o tráfego de rede entre esses dois dispositivos agora é feito com um nó intermediário, que é o computador do atacante (representado pela Figura 2.3).

2.3 Tópicos em criptografia

Em relação à criptografia, a ideia é ocultar informações a fim de que somente interlocutores de interesse do indivíduo que envia a mensagem tenham acesso à mesma. Embora bastante contemporâneo, esse tema não é exclusivo da atualidade. A maioria dos relatos associa a criptografia a, no mínimo, Júlio César, que deu origem à conhecida cifra de César. As técnicas criptográficas permitem que um remetente disfarce os dados de modo que um intruso não consiga obter nenhuma informação dos dados interceptados. O destinatário, é claro, deve estar habilitado a recuperar os dados originais a partir dos dados disfarçados (KUROSE; ROSS, 2013).

Para realizar o processo de se ocultar as informações, são necessários alguns mecanismos, e os dois principais são o algoritmo de criptografia e a chave. O algoritmo é uma técnica normalmente pública que tem a sua definição, estrutura

lógica e teoria matemática, bem definidos em uma recomendação e que pode ser acessada por qualquer pessoa, incluído um usuário mal intencionado. No entanto, mesmo que o indivíduo saiba qual é a técnica e os princípios por trás do algoritmo, esses métodos são pensados de forma a ser impossível (até que se prove o contrário) que alguém acesse as informações criptografadas sem o conhecimento da chave. Se pensarmos no funcionamento e na ideia de um cadeado, temos princípios similares em se tratando da criptografia. Em linhas gerais um cadeado é usado para proteger algo, ideia que é semelhante ao processo de se criptografar um texto, porém, nesse caso, o algo protegido é a informação. Além disso, pensando de uma forma mais simplória, um cadeado só poderia ser destravado pela sua chave ou por cópias idênticas dessa mesma chave. O princípio da chave em um algoritmo de criptografia é o mesmo, somente o indivíduo que tiver a chave (ou uma cópia) poderá acessar a informação.

Conforme descrito em Cheswick, Bellovin e Rubin (2005), para Kaufman e Schneier, uma discussão completa sobre criptografia iria requerer um livro inteiro. Portanto, neste trabalho, serão discutidos apenas em linhas gerais os algoritmos e técnicas trabalhadas e como funcionam suas estruturas básicas.

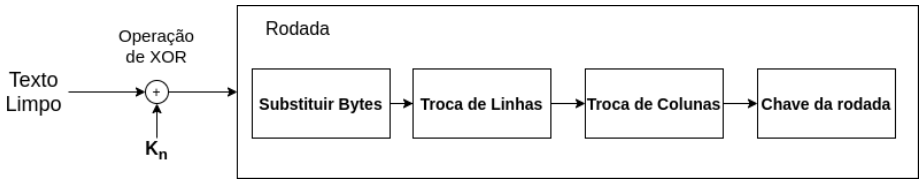
2.3.1 AES

O AES (Padrão de Criptografia Avançando, em inglês, *Advanced Encryption Standard*) é um padrão para criptografia de informações que usa chave simétrica, ou seja, há somente uma chave. O AES é uma forma de sistema de criptografia conhecida como *cifra bloco*. Isto é, ele opera em blocos de tamanho fixo; mapeia blocos de texto claro em blocos de texto cifrado, e vice-versa. Os tamanhos de blocos suportados são de 128, 196 ou 256 (CHESWICK; BELLOVIN; RUBIN, 2005). O algoritmo trabalha com o conceito de *rounds* que executam uma série de operações algumas vezes para se chegar no bloco cifrado. As operações

são: substituição, troca de linhas, mistura de colunas e adição da chave com os dados (como representado na Figura 2.4).

- **Substituição:** essa etapa, usando uma tabela de substituição dos bytes, é usada para realizar uma troca dos bytes originais com os da tabela e dar a impressão de aleatoriedade nas informações;
- **Troca de Linhas e Colunas:** essas duas operações são similares, porém realizadas para blocos diferentes. No caso da troca de linhas, são realizadas permutações com as linhas do bloco (ou tabela) de bytes. Já no caso da troca de colunas, o princípio é o mesmo mas para as colunas;
- **Adição da Chave:** nessa última etapa, é feita uma operação de XOR - "Ou Exclusivo", bit a bit dos dados com a chave.

Figura 2.4 – Passos executados pelo AES.



Fonte: Do Autor (2020).

Além das operações que ocorrem por rodada, o algoritmo tem um processo inicial no qual, antes de qualquer *round*, a chave da rodada (representada por K_n) é usada em uma operação XOR com o texto da rodada atual. Isso pode ser visto pela Figura 2.4.

2.3.2 Modos de Operação

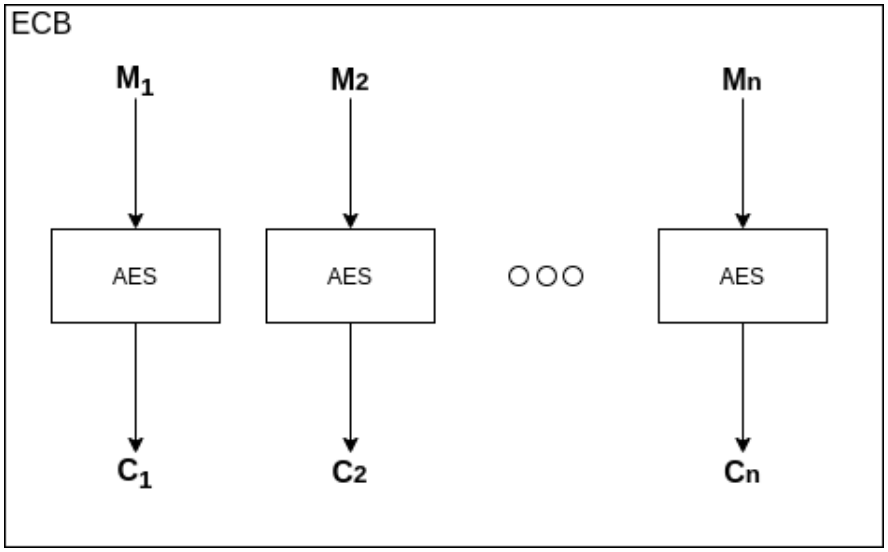
Quando trabalhamos com algoritmos de cifra por bloco, em muitos casos o algoritmo de criptografia em si não é utilizado sozinho. Isso ocorre porque, como o algoritmo de criptografia recebe uma entrada de tamanho fixo e retorna um

dado cifrado de mesmo tamanho, é necessário ter algum mecanismo que separe e trabalhe os blocos de dados a fim de não se limitar o tamanho da informação a ser criptografada. Esse recurso é conhecido como modo de operação.

2.3.2.1 ECB

O modo mais simples de operação, o *Electronic Code Book* (ECB), é também o mais óbvio: o AES é utilizado, tal como definido, em blocos de 16 bytes de dados (CHESWICK; BELLOVIN; RUBIN, 2005). Para isso, o texto claro é dividido em partes iguais de acordo com o tamanho do bloco de cifra e cada parte é criptografada como um texto individual e não como parte da mensagem em sua totalidade (Figura 2.5). No entanto, às vezes é necessário aplicar uma operação de *padding* que vai complementar os dados a fim de tornar a divisão do tamanho da mensagem pelo tamanho do bloco exata.

Figura 2.5 – Estrutura do ECB.



Fonte: Do Autor (2020).

Como nenhum contexto entra em cada encriptação, sempre que os mesmo 16 bytes são criptografados com a mesma chave, tem-se o mesmo texto cifrado

como resultado. Isso permite a um inimigo coletar um tipo de "livro de código", uma lista de textos cifrados de 16 bytes e seus possíveis (ou conhecidos) equivalentes em texto claro. Devido a esse risco, o modo ECB deve ser utilizado somente para transmissão de chaves e vetores de inicialização e nunca para criptografar dados de uma sessão (CHESWICK; BELLOVIN; RUBIN, 2005).

2.3.2.2 CBC

Segundo Cheswick, Bellovin e Rubin (2005), o *Cipher Block Chaining* (CBC) é o modo de operação mais importante. Nele, cada bloco de texto claro é combinado com, via uma operação XOR (OU Exclusivo), com o bloco de texto cifrado prévio antes da encriptação (Figura 2.6), representado pela equação:

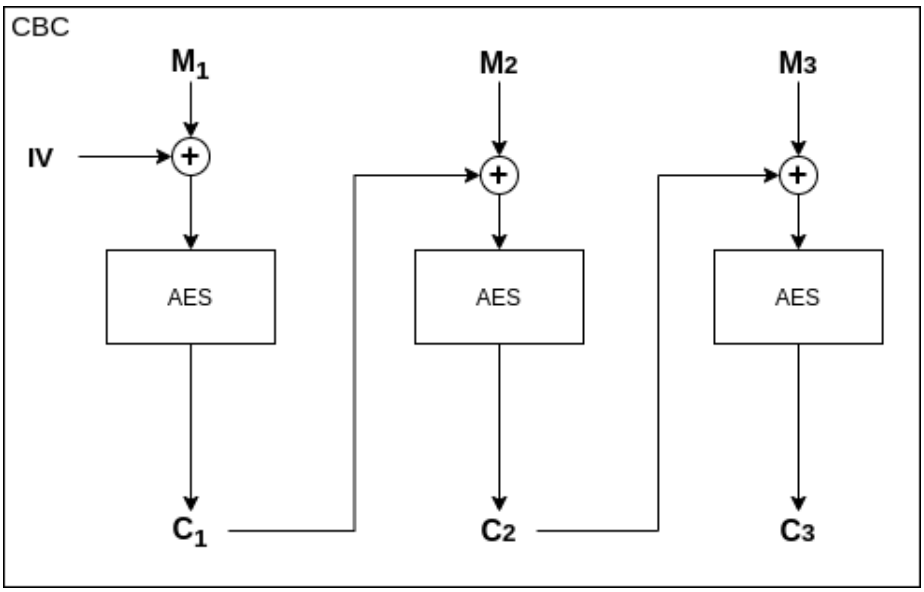
$$C_n \leftarrow K[P_n \oplus C_{n-1}] \tag{2.1}$$

Para descriptografar a equação é invertida:

$$P_n \leftarrow K^{-1}[C_n] \oplus C_{n-1} \tag{2.2}$$

No entanto, apesar de se mostrar eficaz na correção do problema apresentado pelo ECB referente a identificação de padrões, o CBC resultou em alguns problemas. O primeiro deles é como criptografar o primeiro bloco, quando não há um bloco C_0. O segundo se refere ao fato de que se o tamanho da mensagem não for múltiplo de 16 bytes não é possível criptografar o último bloco, e o terceiro problema se refere a capacidade de paralelismo da técnica. O primeiro e segundo problema são resolvidos com um vetor de inicialização (IV) e a técnica de *padding* (que acrescenta dados adicionais para adequar o tamanho da mensagem). Porém, para o último problema, que se refere ao paralelismo, o CBC gera uma dependência entre os blocos de criptografia, tornando assim impossível a paralelização do algoritmo, o que afeta significativamente seu desempenho.

Figura 2.6 – Estrutura do CBC.



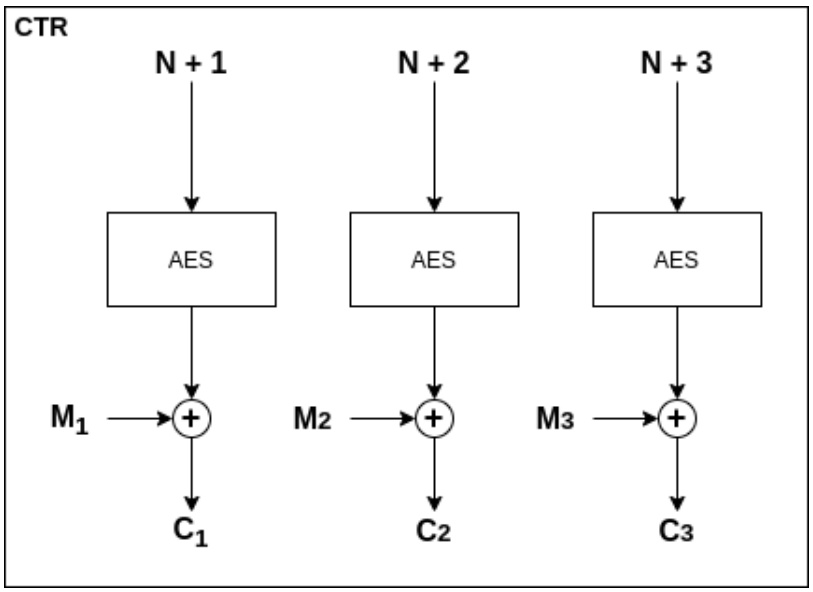
Fonte: Do Autor (2020).

2.3.2.3 CTR

O modo contador (*counter mode*, também chamado de CTR) é o modo de operação usado atualmente com alguns aditivos em conjunto com o AES. O funcionamento é bastante simples, embora a eficiência resultante do método seja elevada e robusta. Primeiramente, o algoritmo recebe um número aleatório que será usado para criptografar as mensagens. O número é então incrementado de acordo com o índice do bloco (no caso do terceiro bloco de mensagem, um número *N* seria incrementado de 3, e assim sucessivamente). Esse resultado é então criptografado usando o algoritmo que vai gerar no final um dado pseudo aleatório e que será usado em uma operação XOR com a mensagem do bloco (processo representado pela Figura 2.7).

A vantagem do modo contador é que ele é paralelizável. Isto é, cada bloco dentro de uma mensagem pode ser criptografado ou descriptografado simultaneamente com qualquer outro bloco. Isso permite a um projetista de hardware lançar

Figura 2.7 – Estrutura do CTR.



Fonte: Do Autor (2020).

uma grande quantidade de chips voltados para o problema relativo a criptografia, o que resulta em uma maior eficiência no processo como um todo (CHESWICK; BELLOVIN; RUBIN, 2005). Vale ressaltar que o processo de descriptografar a mensagem é o mesmo, no entanto, a operação XOR no final é feita com o texto cifrado.

2.3.3 Criptografia de Chave Pública e Aplicações

Como descrito por Cheswick, Bellovin e Rubin (2005), o compartilhamento de uma mesma chave pelos sistemas de criptografia tradicionais pode se tornar problemático, pois seria impossível se comunicar com qualquer sistema sem um arranjo prévio. Além disso, o número de chaves cresce de forma quadrática de acordo com o número de nós na rede (n^2) - onde n é o número de pares na rede). Para resolver esses problemas de forma mais simples, pode-se usar um sistema de chave pública, ou chave assimétrica.

Ao contrário da criptografia de chave simétrica, que só tem uma chave, os algoritmos e técnicas assimétricas usam duas chaves, chamadas de pública e privada. A chave privada é usada para gerar o texto cifrado no lado do emissor, e a mesma não é compartilhada com ninguém, já a chave pública fica disponível para os receptores da mensagem criptografada. O interessante, e pode-se dizer que o poder da técnica, está no fato de que a mensagem criptografada com a chave privada só pode ser descriptografada com a chave pública e vice-versa. Logo, mesmo que um ataque MITM ocorra e o atacante consiga ler as informações através da chave pública, torna-se impossível para ele alterar os dados e gerar o texto cifrado com os dados alterados, pois não possui a chave privada.

A técnica de chave pública é mais usada para garantir autenticidade e integridade das informações. Sendo assim, é aplicada em situações como, por exemplo, estabelecimento de conexões seguras (TLS/SSL) e assinaturas digitais (seja de arquivos ou até mesmo softwares). Por isso, para se garantir a confidencialidade, é necessário combinar essa técnica com a criptografia de chave simétrica. Vale ressaltar também que a técnica é utilizada em conjunto com algoritmos de chave simétrica para suprir alguma vulnerabilidade. Um exemplo mais prático disso é a vulnerabilidade de troca de chaves no algoritmo Diffie-Hellman. O algoritmo para troca de chaves e estabelecimento de uma comunicação segura feita por Diffie e Hellman parte do pressuposto que de não há autenticação com o interlocutor do outro lado, podendo ser o emissor correto, com o qual deseja se comunicar ou um indivíduo mal intencionado. Essa característica torna possível que um indivíduo interceptando a comunicação realize no lugar dos indivíduos a troca de chave em ambos os lados e consequentemente comprometa a comunicação. Isso, no entanto, é corrigido se aplicado um algoritmo de chave pública como o RSA, por exemplo, que assine digitalmente as mensagens, o que garantirá a identidade dos interlocutores.

2.4 Estado da Arte

O foco desta Seção é descrever o que foi encontrado na literatura e como isso contribuiu para todas as decisões tomadas no desenvolvimento deste trabalho.

2.4.1 MQTT e Criptografia

Para Andy, Rahardjo e Hanindhito (2017), existem três razões principais que dificultam a implementação de mecanismos de segurança em dispositivos IoT. A primeira delas é a grande quantidade de dispositivos com recursos de hardware limitados e que, devido a essas limitações, a grande maioria dos dispositivos não conseguem lidar bem com as operações de segurança. A segunda diz que a grande quantidade de dispositivos conectados parece criar ainda mais vulnerabilidades, visto que, para departamentos de TI, é comumente mais difícil gerenciar dispositivos de vários tipos. E por último, o lapso do desenvolvedor com a segurança, que tende a se preocupar mais com funcionalidade do que segurança. Pode-se acrescentar ainda mais um ponto referente ao relapso com a segurança em redes IoT, que é decorrente da limitação do alcance da comunicação e a concentração desses dispositivos em redes privadas e limitadas, o que dificultaria o acesso de atacantes.

Agora que são conhecidas algumas da razões que dão origem a vulnerabilidades ou falhas de segurança, pode-se explorar um pouco mais sobre os ataques em si. Em Naik e Maral (2017), os autores descrevem quatro principais tipos de ataques a esses dispositivos e qual o objetivo em cada um deles. O primeiro é a clonagem de dispositivos, onde um hardware externo pode ser conectado de forma a se parecer e até mesmo agir como o dispositivo correto, mesmo não sendo. Esse problema pode ser rapidamente escalável, o que torna ainda mais difícil identificar quais são os dispositivos legítimos e quais não são. Em segundo, temos a captura de dados sensíveis, na qual os dados trafegados não são protegidos por criptografia e por isso podem ser capturados e analisados. O DDoS também faz parte dos ataques que se enquadram nos dispositivos IoT e pode ser um dos que mais causa

prejuízo para companhias que trabalham com processamento de grandes volumes de dados desses dispositivos. Nesse ataque, são enviadas diversas requisições inúteis do ponto de vista da aplicação, mas que buscam ofuscar e ocupar a rede a fim de tornar o serviço indisponível. E por fim, temos o ataque de acesso não autorizado, no qual um usuário não legítimo se ingressa na rede e pode fazer diversas operações como a análise de tráfego e captura de mais informações.

Além dos problemas já citados anteriormente, em Jadhav, Kulkarni e Swamy (2017), são definidos outros problemas ainda comuns em aplicações de rede, principalmente se tratando de serviços que trabalham na camada de aplicação. Um deles é a forma como o sistema trabalha com grande quantidade de dados e, nesse caso, os problemas vão desde o controle de perda pelos protocolos de transporte até a criptografia de grande volume de informações. O outro é referente as vulnerabilidades de *buffer overflow*. Nesse tipo de vulnerabilidade, se algum dado é tratado de forma incorreta, a pilha de memória referente a parte de dados pode ultrapassar sua região e chegar na região de programa, permitindo então que o atacante execute um código próprio no dispositivo, levando até mesmo ao controle do dispositivo. Isso em um servidor que executa o serviço do *broker* poderia ser catastrófico.

Em grande parte da literatura encontrada, são citados os mesmos problemas aqui descritos referentes às vulnerabilidades do MQTT. No entanto, alguns dos autores buscaram corrigir essa defasagem com algumas propostas. Em Mathews e Gondkar (2019), os autores propõem uma modificação no protocolo com a finalidade de suportar criptografia no *payload* de forma nativa. A abordagem corrige grande parte dos problemas já citados, no entanto, requer modificações no tamanho do cabeçalho e nos pacotes de controle do protocolo. Isso, no entanto, apesar de ser uma proposta interessante e que deve ser amplamente considerada, é uma solução que requer um tempo maior para ser efetivada, visto que grande quantidade dos serviços usando MQTT precisam sofrer modificações e ter

bibliotecas atualizadas, além de requerer que desenvolvedores façam alterações completas em sistemas legados.

3 MATERIAIS E MÉTODOS

Neste Capítulo, são apresentados os materiais e métodos utilizados para o desenvolvimento deste trabalho. Inicialmente, são apresentados os softwares usados, desde a configuração dos serviços até as ferramentas para realização de ataques. Depois, tem-se uma breve descrição sobre as bibliotecas necessárias para comunicação com o *broker* em cada uma das plataformas de desenvolvimento e a descrição das placas em si. Por fim, é feita uma abordagem geral sobre todo o ambiente, definindo processos de configuração, arquitetura da comunicação e o processo de criptografia para cada um dos dispositivos, a fim de se prover uma visão geral que ilustre melhor o trabalho.

3.1 Softwares

Esta Seção se destina a descrever as ferramentas utilizadas na simulação dos ataques e alguns de seus recursos. Algumas ferramentas, como o *Wireshark*, são bem mais completas e, por isso, vamos abordar apenas o essencial e os recursos que foram utilizados efetivamente neste trabalho.

3.1.1 Mosquitto Eclipse Broker

Eclipse Mosquitto é um *broker* de mensagem que implementa o protocolo MQTT nas versões 5.0, 3.1.1 e 3.1. Mosquitto é um programa *lightweight* adequado para uso em todos os dispositivos desde os computadores de placa única (com baixa potência) até servidores completos. O Projeto Mosquitto também provê um biblioteca em C para implementar clientes MQTT (ECLIPSE FOUNDATION, 2020), que pode ser utilizada por um desenvolvedor para incorporar a comunicação com o MQTT na sua aplicação C, sem a necessidade de se implementar todo o protocolo do início na linguagem. Por ser de código aberto, o *mosquitto* pode ser instalado e usado em qualquer Sistema Operacional, seja esse GNU/Li-

nux, Windows ou OS X, basta que o usuário tenha as ferramentas necessárias para fazer a compilação da ferramenta.

3.1.2 Arpspoof

O *Arpspoof* é um programa para executar um ataque do tipo *ARP Spoofing* contra algum indivíduo na rede local descriptografada (ARPSPOOF, 2016). O programa envia duas requisições do tipo ARP, uma para o *gateway* e outra para a vítima para obter seus endereços MAC. Os endereços obtidos são então usados para construir uma resposta ARP falsa para a vítima indicando que o computador do atacante é o *gateway* padrão o que faz todo o tráfego passar pelo computador do atacante. Alguns sistemas operacionais, como GNU/Linux, possuem heurísticas para identificar essas respostas ARP forjadas e provavelmente vão identificar o ataque, mas outros, como o Windows, conseguem ser facilmente enganados.

3.1.3 Wireshark

O *Wireshark*, originalmente desenvolvido por Gerald Combs, é a maior ferramenta do mundo voltada para análise de protocolos de rede. É usado desde o âmbito industrial até instituições educacionais, e alguns de seus recursos segundo Combs (1998), incluem:

- Inspeção profunda de vários protocolos de rede, com a adição de vários de forma recorrente pela comunidade de desenvolvedores;
- Captura ao vivo e análise *offline*;
- Multiplataforma: pode ser executado em Linux, Windows, OS X, FreeBSD, entre outros;
- Os dados capturados podem ser analisados via interface gráfica (GUI) ou através de um terminal interativo (TTY);

- Uma grande quantidade de filtros;
- Análise de pacotes VoIP (*Voice over IP*);
- Pode salvar os dados capturados em diferentes formatos para diferentes plataformas;
- Suporta diferentes protocolos da camada de Enlace (Ethernet, IEEE 802.11, Bluetooth, etc);
- Suporte de criptografia para diversos protocolos, como IPsec, ISAKMP, Kerberos, SSL/TLS, WEP e WPA/WPA2;
- Diversos formatos de exibição dos dados, como XML, CSV, PostScript ou texto "puro".

3.2 Bibliotecas Utilizadas

Esta Seção visa descrever as bibliotecas utilizadas nos testes, bem como seus principais recursos e limitações para cada plataforma.

3.2.1 PubSubClient (MQTT)

O PubSubClient é uma biblioteca MQTT desenvolvida para suportar tarefas de publicar mensagens e se inscrever em tópicos de um *broker*. A biblioteca possui uma grande variedade de exemplos e rascunhos que visam introduzir e exemplificar seu uso. Apesar de fornecer a grande maioria dos recursos do MQTT, a biblioteca possui algumas limitações, dentre as quais pode-se citar:

- Só é possível se publicar mensagens com QoS 0, mas a inscrição em tópicos pode ser feita para QoS 0 ou 1;
- O tamanho máximo da mensagem é de 256 bytes incluindo o cabeçalho. Isso pode ser configurado chamando a função PubSubClient::setBufferSize(tamanho), onde tamanho é o tamanho da mensagem;

- O recurso de *keepalive* é configurado por padrão para 15 segundos, mas isso também pode ser configurado com uma chamada de método (PubSubClient::setKeepAlive);

- Esse cliente do MQTT usa, por padrão, a versão 3.1.1 do protocolo. No entanto, pode ser mudado para a 3.1 alterando o arquivo de cabeçalho PubSubClient.h;

- Não há suporte, até a publicação deste trabalho, para a versão 5.0 do MQTT.

A biblioteca foi a escolhida como cliente MQTT para sistemas embarcados por ser simples e por trabalhar em duas das plataformas utilizadas neste trabalho, o Arduíno Uno com o módulo Ethernet e o ESP8266. Mais informações podem ser encontradas no site oficial em (O'LEARY, 2020).

3.2.2 Eclipse Paho

O projeto Eclipse Paho provê um cliente de código aberto para o MQTT e MQTT-SN voltado para projetos novos, existentes e emergentes de Internet das Coisas (CRAGGS; DASGUPTA; PAGLIUGHI, 2020). O projeto envolve uma série de bibliotecas em diferentes linguagens, dentre as quais tem-se o C, Python, Java, JavaScript e o C#. Para desenvolver o algoritmo no Raspberry Pi foi escolhido o Python, devido a sua robustez e simplicidade na sintaxe e configuração do ambiente de desenvolvimento.

O projeto reflete as restrições físicas e de custo inerentes à conectividade do dispositivo. Seus objetivos incluem níveis eficazes de desacoplamento entre dispositivos e aplicativos, projetados para manter os mercados abertos e encorajar o rápido crescimento de aplicativos e *middleware* corporativos e aplicações Web escaláveis. Contém também implementações de cliente de publicação/assinatura MQTT para uso em plataformas integradas, junto com o suporte de servidor cor-

respondente conforme determinado pela comunidade (CRAGGS; DASGUPTA; PAGLIUGHI, 2020).

3.2.3 PyCrypto

A PyCrypto (Python *Cryptography Toolkit*) é um biblioteca escrita em Python, que busca fornecer uma coleção de algoritmos de criptografia e funções de *hashing*. O pacote é estruturado de forma a facilitar a adição de novos módulos. Segundo a documentação oficial, algumas das possibilidades de uso desses módulos é a escrita de ferramentas de administração seguras, uso em *daemons* e servidores, além da comunicação segura entre clientes e servidores.

3.3 Sistemas Embarcados

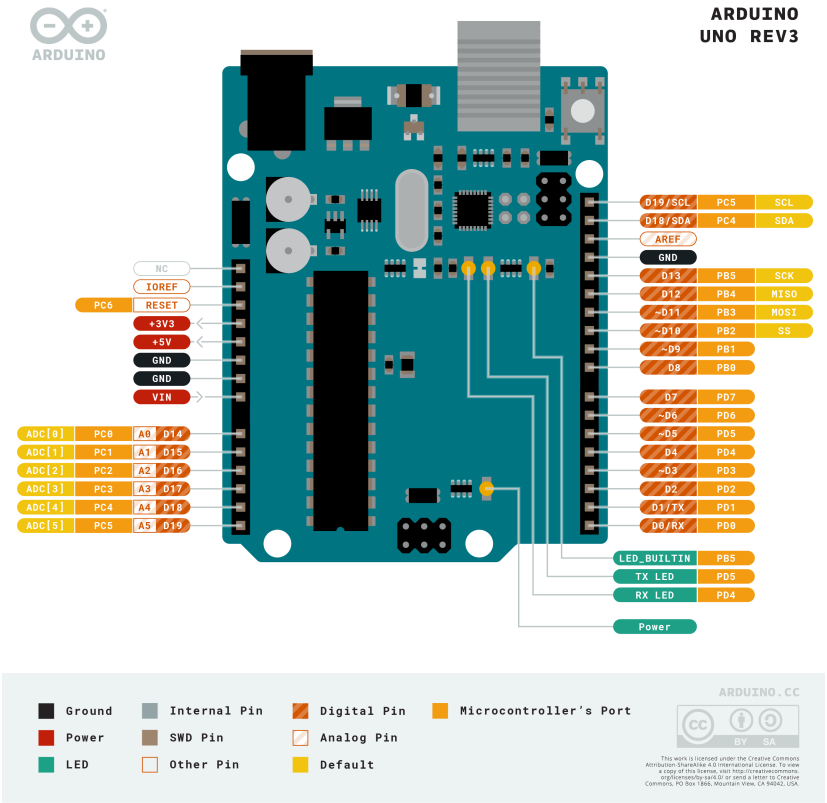
Diferentemente dos computadores de uso geral que são projetados para executar quaisquer tarefas, um sistema embarcado é projetado para uma tarefa específica, o que implica que, em muitos casos, seu *hardware* seja limitado, de baixo custo e consumo de energia reduzido. Uma rede IoT é composta não só por computadores de uso geral mas em grande parte por esses dispositivos, por isso, neste trabalho, vamos usar algumas dessas plataformas para executar os testes e análise de desempenho em cada caso, com seu hardware em específico. Vale ressaltar que esses dispositivos foram usados neste trabalho devido a sua grande popularidade no mercado e por serem também as opções que tínhamos disponíveis.

3.3.1 Arduíno UNO R3 com Ethernet Shield

Segundo o site oficial da plataforma Arduíno (ARDUÍNO COMMUNITY, 2020), o Arduíno UNO é uma placa indicada para começar os estudos em eletrônica e programação, visto que é a placa mais usada e documentada de toda a família Arduíno. A placa é construída com base no microcontrolador ATmega328P, o qual possui 14 pinos digitais de entrada e saída (seis dos quais suportam saídas

PWM), seis entradas analógicas, um ressonador de cerâmica de 16 MHz, 64KB de memória, uma conexão USB, um pino de energia, um *header* ICSP e um botão de reiniciar.

Figura 3.1 – Mapa de pinos do Arduino UNO.



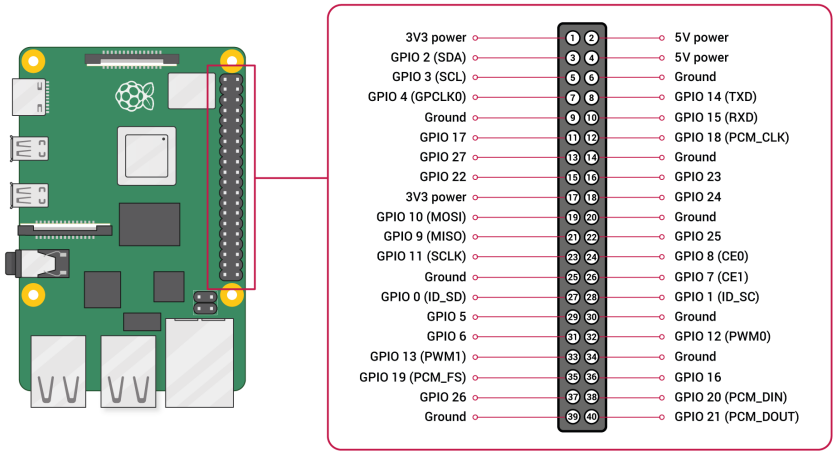
Fonte: ARDUÍNO COMMUNITY (2020).

Apesar da variedade de funções e recursos do UNO, a placa não possui uma conexão nativa com a internet, para isso é necessário um módulo adicional chamado de *Ethernet Shield*. O módulo é encaixado em cima da placa Arduino e isso faz com que os pinos da placa não sejam todos ocupados, permitindo assim o uso de outros periféricos e dispositivos adicionais. Uma descrição da pinagem da plataforma pode ser encontrada na Figura 3.1.

3.3.2 Raspberry Pi 3 Model B

O Raspberry Pi é um dispositivo bastante utilizado por parte da comunidade *maker*, mas que, diferentemente do Arduíno ou do ESP8266, possui um hardware mais avançado e próximo dos computadores de uso geral. O dispositivo foi criado pela Fundação Raspberry Pi, no Reino Unido, com a finalidade de incentivar o ensino da Ciência da Computação básica para jovens em escolas e universidades da região, com o intuito de possuir também um preço acessível. O dispositivo é capaz de executar um sistema operacional como o Linux completo, com acesso a interface gráfica e acesso à internet. A Figura 3.2 mostra a estrutura de pinos do Raspberry 3 modelo B.

Figura 3.2 – Mapa de pinos do Raspberry PI.



Fonte: RASPBERRY PI FOUNDANTION (2020).

A plataforma conta com processadores ARM, que aplicam o conceito RISC (*Reduced Instruction Set Computer* - Conjunto Reduzido de Instruções de Computador), a fim de promover um microprocessador de uso geral que seja eficaz e barato para dispositivos que não requeiram um grande poder computacional. Por isso, apesar de ser um computador completo e ter a capacidade de rodar um

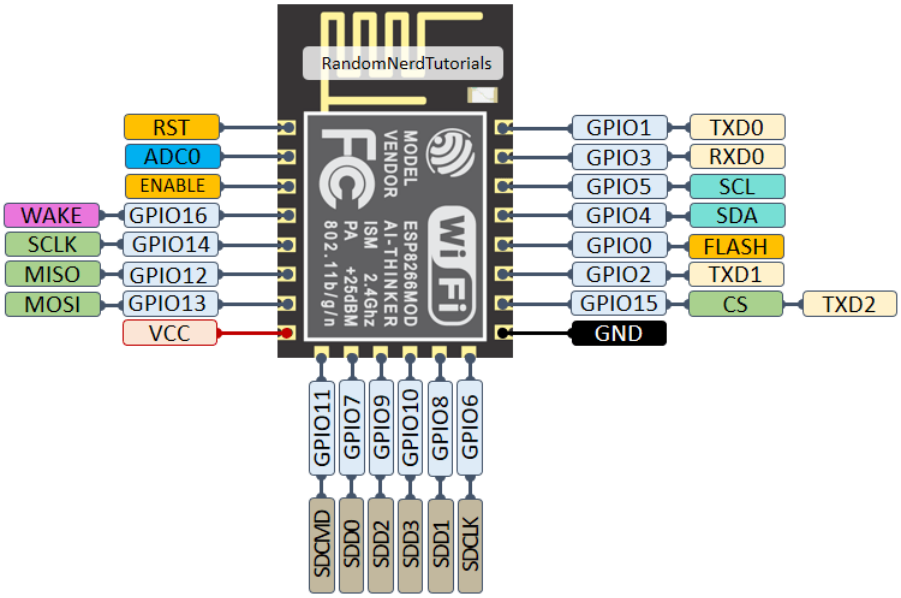
sistema operacional, algumas tarefas mais complexas, como edição de imagens, não podem ser executadas de forma eficiente, logo o Raspberry é mais utilizado em aplicações que não requerem muito desempenho, sendo uma alternativa interessante para uso em uma rede IoT. Para reforçar um pouco mais a capacidade desse dispositivo, um exemplo prático seria configurar o software Mosquitto em um Raspberry e fornecer assim uma rede MQTT, onde outros nós poderiam publicar e se inscrever em tópicos no servidor, que no caso seria um Raspberry.

3.3.3 NodeMCU/ESP8266

O ESP8266 é um chip desenvolvido pela empresa Chinesa Espressif, e é integrado com um processador Tensilica de 32-bit. Diferente de outros dispositivos IoT, o ESP8266 possui o módulo Wi-Fi integrado, o que facilitou bastante o mercado e a cultura *maker* por fornecer conexão com a internet de fábrica. Além disso, na sua primeira versão (ESP8266-01) a plataforma já contava com um *clock* de 80MHz, 64KB de memória (o dobro do Arduíno UNO R3), 16 pinos de GPIO e a conexão Wi-Fi na velocidade de 2.4GHz, o que garante o baixo consumo de energia.

O *chip* do microcontrolador em si não possui a capacidade de fazer a conexão com o computador para o ambiente de desenvolvimento, logo, em conjunto com o chip, é necessário ter uma plataforma de hardware que se encarregue da comunicação. Neste trabalho, será usado o NodeMCU que vai se encarregar de toda a parte eletrônica do chip, além de fornecer uma interface micro USB para comunicação com o computador. Este trabalho não entrará em detalhes sobre a pinagem do microcontrolador visto que o foco é a criptografia em rede, mas a Figura 3.3 mostra o esquema de pinagem da versão usada.

Figura 3.3 – Mapa de pinos do ESP8266.



Fonte: Random Nerd Tutorials (2020).

3.4 Métricas de Análise

Esta Seção visa descrever as métricas usadas na análise dos testes e implementações realizadas. São analisados diferentes aspectos: segurança da informações e seus princípios básicos; consumo de memória e; tempo de execução (desempenho).

Uma questão que não foi abordada neste trabalho devido a limitações de equipamento e as condições de acesso a laboratórios da universidade, foi a análise do consumo de energia com a aplicação da criptografia. Foi feita uma tentativa de se medir a corrente consumida com um uso do multímetro, mas a imprecisão e variação dos resultados, pela falta de equipamentos mais qualificados resultou na não abordagem desse tema no presente trabalho.

3.4.1 Métricas sobre Consumo de Memória

Nessa métrica, os resultados foram obtidos de forma simples e direta, usando os dados fornecidos pelo próprio compilador no caso do Arduíno e do ESP-8266 e o tamanho do processo alocado na memória, no caso do Raspberry. Em se tratando dos dois primeiros dispositivos citados anteriormente, o valor escolhido como referência foi o do compilador por dois motivos principais. O primeiro está relacionado com o código de testes desenvolvido. O algoritmo trabalha exclusivamente com variáveis estáticas, salvo no caso do objeto instanciado para realizar a criptografia, por isso, a diferença entre o valor fornecido pelo compilador e o código em execução (com a alocação dinâmica) seria mínima. O segundo fator diz respeito a não trivialidade de se obter a informação da memória do programa em execução nesses dispositivos (diferente do Raspberry, que por ter um Sistema Operacional, torna esse processo mais fácil). Esses dois fatores alinhados, resultaram nessa escolha de avaliação.

Já no caso do Raspberry, a quantidade de memória é um problema bem menor. A placa, além de possuir uma memória não volátil expansível (o cartão SD pode ser substituído por outro com mais capacidade), possui um sistema operacional, que é responsável pelo controle de recursos. Por isso, nesse último caso, o uso de memória RAM é o considerado, visto que seu gerenciamento incorreto pode gerar problemas diversos (até mesmo falhas de segurança, como o *buffer overflow*).

3.4.2 Métricas de Tempo de Execução

Neste tópico, é dado um foco maior nos dispositivos de forma individual, em como o algoritmo de criptografia utiliza os recursos de hardware e qual o tempo gasto para executar os três passos básicos da criptografia, que são: setar a chave, criptografar e descriptografar a mensagem. Para isso, foi desenvolvido, para cada uma das plataformas, um algoritmo que executa as 3 etapas do processo de comu-

nicação criptografada e cada uma dessas etapas foi analisada de forma individual e posteriormente como um todo. Em se tratando de medir o tempo de execução de uma sequência de instruções, é sabido que alguns processos podem ter valores muito pequenos (na casa de microssegundos), o que torna necessária uma série de execuções subsequentes para se obter uma média que representa uma melhor estimativa do valor real. Neste trabalho, cada etapa citada anteriormente foi executada 1000 vezes.

Ainda em relação aos dados obtidos, pelo fato de termos o Raspberry Pi, que é um dispositivo que funciona através de um sistema operacional, e que este é responsável por alocar processos e gerenciar o hardware, percebeu-se a necessidade de executar também o algoritmo de testes múltiplas vezes e obter o desvio padrão que reforçaria a confiabilidade dos resultados e das conclusões inferidas. Assim, todos os dados apresentados posteriormente serão baseados em valores estatísticos como média e o desvio padrão com uma população de 10 elementos (ou seja, dados obtidos de 10 execuções do algoritmo da Figura 3.4).

Figura 3.4 – Pseudocódigo métrica de desempenho.

```
DECLARACOES
    INTEIRO Contador
    INTEIRO LONGO Inicio
    INTEIRO LONGO Fim

ATRIBUIR TEMPO_EXECUCAO A Inicio
PARA Contador DE 0 ATE 1000 FACA
    EXECUTAR OperacaoCriptografia
FIM FACA
ATRIBUIR TEMPO_EXECUCAO A Fim
IMPRIMIR ((Fim - Inicio) / 1000)
```

Fonte: Do Autor (2020).

Sobre a técnica usada para coletar as informações, o algoritmo é bem simples. Todo dispositivo computacional que use um microprocessador ou um microcontrolador, tem um função nativa que obtém o valor do tempo de execução na-

quela instrução desde que o programa foi iniciado. Com esse conceito em mente, esse valor foi capturado antes de se iniciar cada etapa do processo de criptografia e logo após o fim do processo. Com isso, podemos calcular a diferença entre esses dois valores e obter o tempo de execução em microssegundos (pseudocódigo da Figura 3.4) para o total de execuções feito na etapa. O exemplo dado trata um operação de criptografia qualquer exatamente porque o princípio é o mesmo para qualquer uma delas e uma vez definida a técnica usada, fica mais fácil entender e interpretar os valores.

3.5 Ambiente implementado

A partir da revisão e entendimento das ferramentas, bibliotecas e plataformas citadas nas Seções anteriores, foi estruturado um ambiente de testes onde o *pentest* nos diferentes cenários poderia ser executados de forma controlada e dentro dos meios legais, em relação à segurança da informação. A Figura 3.5 demonstra como esse sistema foi estruturado.

Para o *broker* local, utilizamos um computador comum com o sistema operacional GNU/Linux, sendo Debian 10 a distribuição usada. Nesse computador, foi instalado o Mosquitto *broker* diretamente pelo gerenciador de pacotes *aptitude*, disponível por padrão no sistema. Após a instalação, foi criada uma credencial de acesso usando o módulo *mosquitto_passwd*, que é instalado junto com o Mosquitto. O módulo anterior gera um arquivo de usuários, onde cada entrada está no formato "usuário:senha", sendo a senha um texto cifrado, por questões de segurança. Com o arquivo anterior gerado, foram feitas modificações no arquivo de configuração do software localizado em: "/etc/mosquito/mosquitto.conf"com a inserção das informações "allow_anonymous false"e "password_file <diretório do arquivo>", em que a primeira entrada se refere a permissão de usuários anônimos (sem credenciais) para se conectarem ao *broker*. Após essas configurações, foi ne-

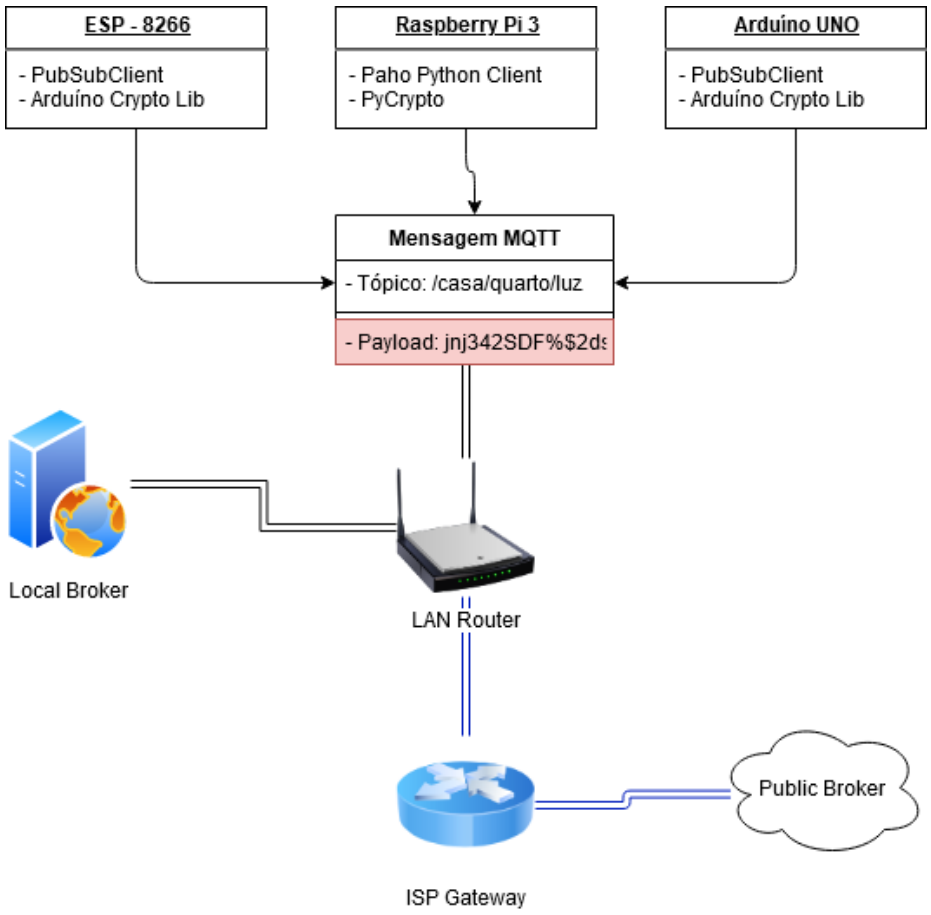
cessário apenas reiniciar o serviço do Mosquitto. Um ponto que vale ser destacado é que nenhum desses passos precisou ser feito no *broker* público.

O desenvolvimento do software para o Arduíno e o ESP foi feito utilizado o editor de código *Visual Studio Code* (codinome VS Code), com a extensão PlatformIO. O VS Code é um editor lightweight, porém muito poderoso que é disponível para Windows, macOS e Linux. A ferramenta vem com suporte nativo para JavaScript, TypeScript e Node.js e tem ainda um rico ecossistema de extensões para outras linguagens (MICROSOFT CORPORATION, 2020). O PlatformIO é uma *cross-platform*, *cross-architecture*, ferramenta múltipla e profissional para engenheiros de sistemas embarcados e também para desenvolvedores que escrevem aplicações para produtos nessa categoria (PlatformIO COMMUNITY, 2020, tradução nossa).

Na Figura 3.5, está cada um dos dispositivos representados por um retângulo, e no corpo do retângulo são definidas as bibliotecas usadas por cada um dos nós. Após receber os dados de algum sensor ou qualquer outra fonte de entrada, os dispositivos iniciam a etapa de se criptografar os dados e enviar o pacote MQTT. Nessa etapa, o *payload* da mensagem é criptografado (representado pela linha *payload* no retângulo com o nome "Mensagem MQTT", onde os dados são aparentemente aleatórios).

Um vez criptografada, a mensagem envia o pacote MQTT para o roteador local, que vai assim decidir, através do endereço IP, se o dispositivo final se encontra na rede local ou na internet pública. Caso o endereço definido ao se criar a conexão seja o IP do servidor que hospeda o *broker* local, o roteador irá então então encaminhar a mensagem publicada para o *broker*, que posteriormente fará o mesmo envio para todos os dispositivos inscritos no tópico. No caso do *broker* estar hospedado na Internet pública, o pacote deverá então ser encaminhado para o roteador do ISP (*Internet Service Provider*) até alcançar o servidor de destino,

Figura 3.5 – Arquitetura do sistemas de testes.



Fonte: Do Autor (2020).

que posteriormente fará o mesmo processo de enviar o pacote para os dispositivos inscritos.

O próximo Capítulo apresenta os resultados dos testes executados no ambiente descrito neste Capítulo e a análise dos mesmos a partir das métricas descritas anteriormente.

4 RESULTADOS

O ambiente de testes neste trabalho consiste em dois cenários separados que envolvem a internet pública e uma rede privada local (conforme Figura 3.5). No primeiro cenário, os três dispositivos computacionais (o Arduíno, o ESP8266 e o Raspberry Pi) foram conectados a um *broker* público fornecido pela fundação Eclipse e que é accessível através de Craggs, Dasgupta e Pagliughi (2020). Porém, por se tratar de um ambiente público para testes, nenhuma credencial de acesso foi requerida e as opções de configurações no servidor eram inexistentes para usuários externos. No segundo ambiente, foi configurada uma rede local com um *broker* ativo em um computador também na mesma rede, no entanto, esse dispositivo possuía credenciais de acesso para que a conexão com o *broker* fosse efetivada. Em ambos ambientes de teste, foram realizados ataques com quatro tipos de algoritmos, no qual um não possuía nenhum tipo de criptografia e os outros utilizavam criptografia AES de 128 bits variando o modo de cifragem de bloco utilizado. Este capítulo visa, portanto, mostrar os resultados da aplicação dos algoritmos em termos da segurança e do uso de recursos dos dispositivos utilizados em cada um dos cenários citados anteriormente.

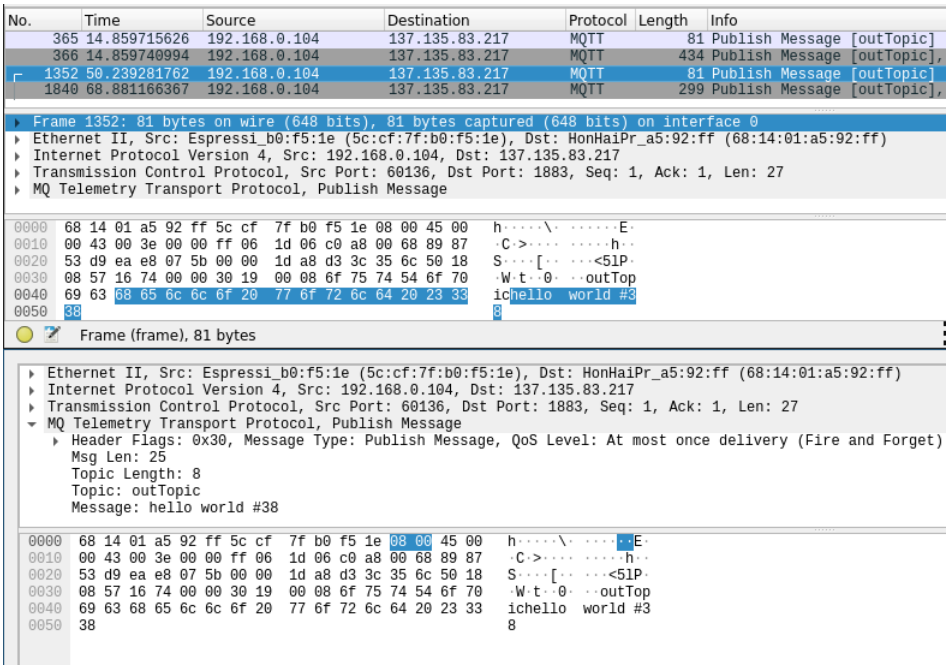
Primeiramente, vamos apresentar como os ataques ocorreram no sistema testado, quais as vulnerabilidades e informações que puderam ser obtidas com esses ataques; como a criptografia ajuda a proteger o sistema como um todo; e por fim, quais as vulnerabilidades restantes.

4.1 Simulação de ataque

Nessa primeira análise, em ambos os cenários, foi efetuado um ataque MITM utilizando a ferramenta *Arpspoof*. A máquina atacante usava o software *netdiscover* para listar todos os dispositivos na rede e, após identificar os alvos, o *arpspoof* entra em ação para enviar requisições falsas e falsificar registros na tabela ARP a fim de alterar quais são os destinatários e emissores legítimos das

mensagens que são transmitidas. Uma vez executado com sucesso, todo o tráfego entre os dois dispositivos é enviado através do computador do atacante, o que permite a captura e análise dos pacotes trafegados. No primeiro cenário citado na introdução deste Capítulo, o ataque ocorreria entre o dispositivo IoT e o *Gateway* da rede local que é responsável pela comunicação com a internet global. Primeiramente, o ataque foi executado no sistema sem nenhum tipo de criptografia e, ao analisarmos o pacote capturado (Figura 4.1), percebemos que podemos obter todas as informações referentes à requisição, tais como: tamanho da mensagem, tópico de publicação, e mensagem a ser publicada (no caso o texto "hello world 438").

Figura 4.1 – Pacote capturado sem criptografia.



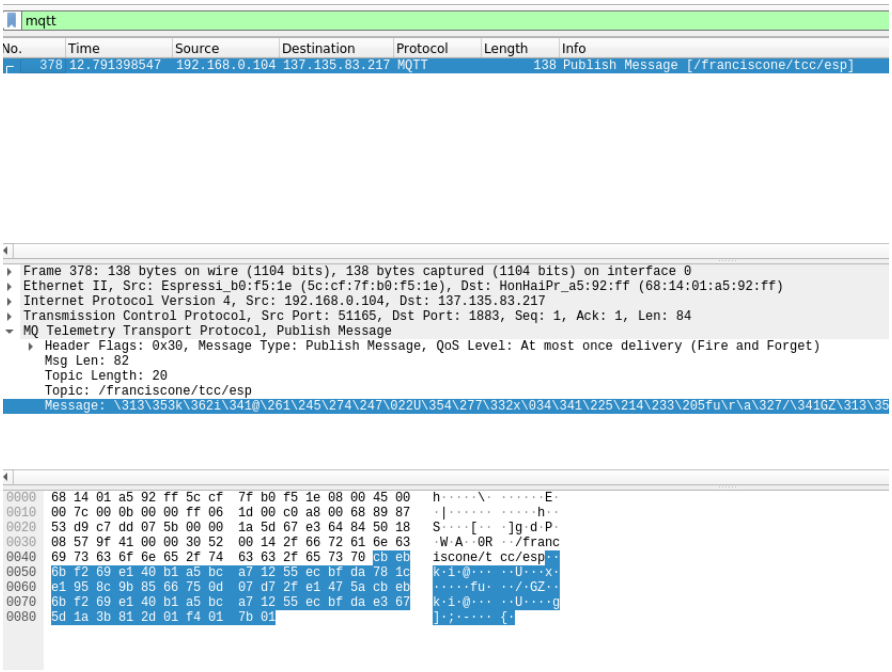
Fonte: Do Autor (2020).

Esse tipo de mensagem, além de ser vulnerável ao vazamento de informações, possui uma vulnerabilidade ainda mais grave, que é a permissão de alteração de conteúdo. Nesse caso, foi enviado um texto puro e simples, com nenhuma informação relevante. Mas, caso fosse enviado, por exemplo, um JSON (*Java Script*

Object Notation) contendo informações de sensores que impactariam na tomada de decisões no sistema, a mínima alteração poderia ter efeitos negativos de grande magnitude.

Avançando um pouco mais nos tópicos de criptografia, foi feita um implementação do AES 128 no modo ECB (Figura 4.2) e, com isso, podemos ter uma visão mais clara e prática do sistema, de seus problemas e a melhoria obtida.

Figura 4.2 – Pacote capturado com criptografia AES 128 modo ECB.



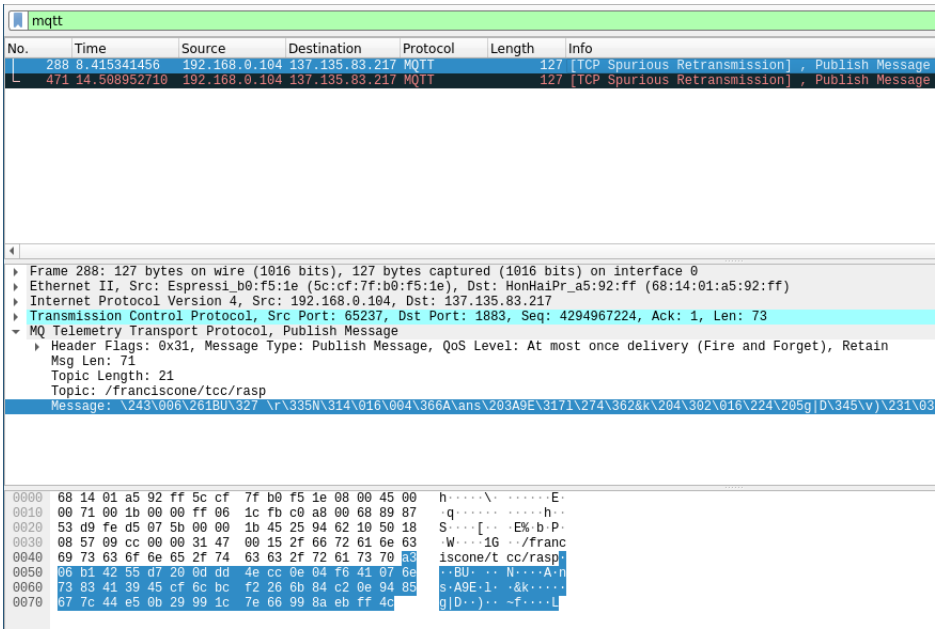
Fonte: Do Autor (2020).

Ao se comparar o AES 128 ECB com o método sem nenhuma forma de criptografia, é possível perceber que, apesar de algumas informações ainda estarem visíveis, como o tópico e IPs de origem e destino, a mensagem se tornou um texto embaralhado e que parece sem sentido nenhum, protegendo assim a informação, em uma primeira análise. No entanto, a finalidade desse pacote é demonstrar a vulnerabilidade descrita no Capítulo 2 na Seção que fala sobre os modos de ope-

ração, em específico sobre o ECB. O tópico em questão descreve como o ECB vai gerar o mesmo texto cifrado para um mesmo bloco de 16 bytes dado como entrada.

Analisando a última parte da Figura 4.2, que mostra os bytes do final do pacote, referentes à parte da mensagem, pode-se perceber que os 16 primeiros bytes (os que começam em "cb eb 6b"e que começam na posição 004E) são exatamente iguais às 16 bytes localizados na posição 006E (que também se iniciam em "cb eb 6b"). Assim, do endereço 004E até 005D, existe uma mensagem completamente idêntica a informação do endereço 006E ao 007D, permitindo assim, que, mesmo que a mensagem esteja criptografada, o atacante consegue obter alguma informação no ataque, o que quebra um dos objetivos da criptografia, que é não permitir o acesso a nenhum tipo de informação no texto cifrado.

Figura 4.3 – Pacote capturado com criptografia AES 128 modo CBC.

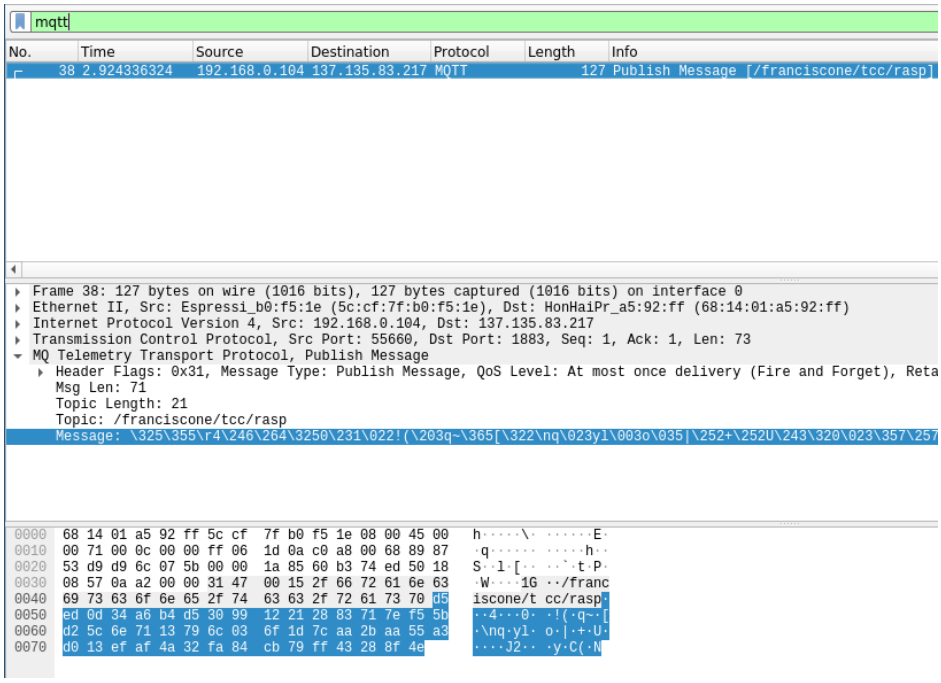


Fonte: Do Autor (2020).

Tratando-se de questões de segurança e de vulnerabilidades, o modo CBC, como já citado no Capítulo 2, corrige o problema apresentado pelo ECB, porém apresenta problemas de desempenho, que são abordados em uma Seção específica.

Na Figura 4.2, é possível ver de forma prática que o problema anterior foi corrigido impedindo que informações sobre a mensagem sejam obtidas no texto cifrado. No entanto, ao observar o diagrama de funcionamento do modo na Figura 2.6, nota-se que a dependência gerada pela criptografia entre os blocos da mensagem gera uma nova vulnerabilidade, que consiste na adulteração dos dados pelo atacante que se encontra no meio. O texto cifrado, uma vez alterado, tornaria impossível a descriptografia do bloco em questão e, consequentemente, dos demais blocos, visto que os blocos subsequentes dependem dos anteriores. Logo, uma alteração no primeiro byte da mensagem faria com que o processo de leitura dos dados por parte do destinatário fosse impossível. Isso poderia ser caracterizado como um tipo de ataque de negação de serviço (*Denial of Service - DoS*), visto que os clientes envolvidos ficariam impedidos de se comunicar.

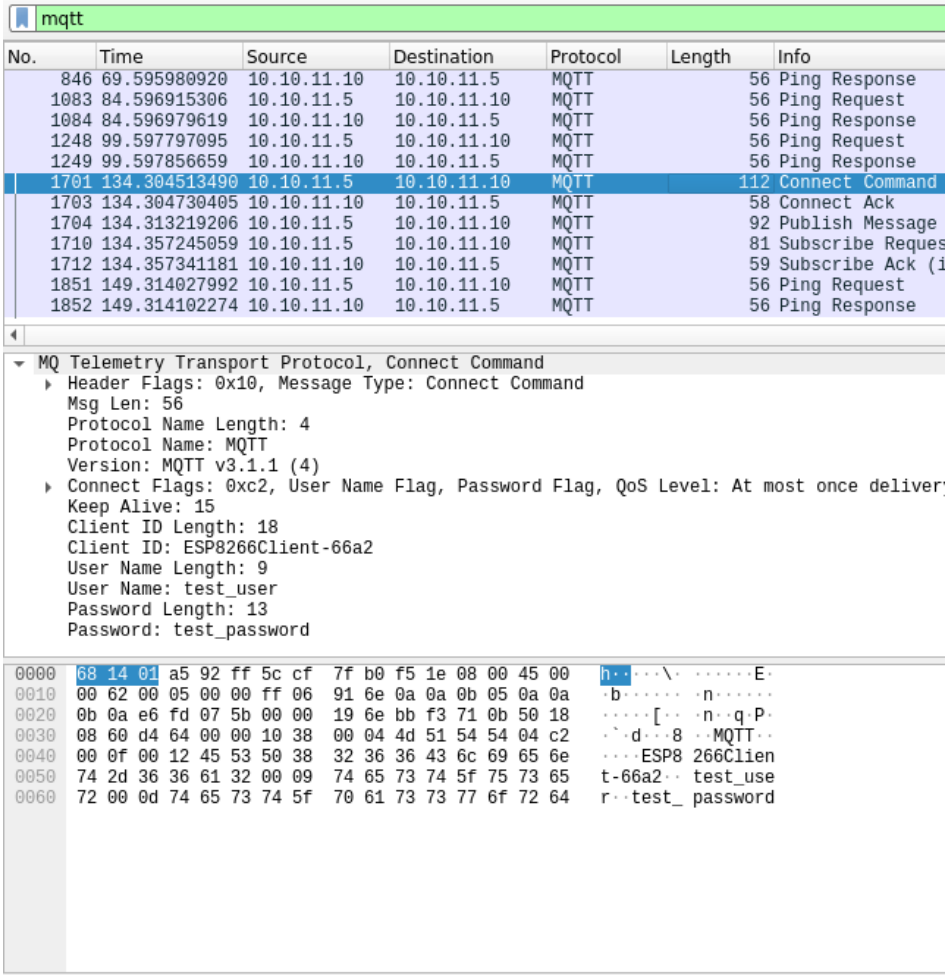
Figura 4.4 – Pacote capturado com criptografia AES 128 modo CTR.



Fonte: Do Autor (2020).

Com esse problema em mente, foi desenvolvido o modo CTR (Figura 4.4), que, além de resolver a questão da dependência entre os blocos, torna o algoritmo paralelizável, permitindo, assim, o aproveitamento do *pipeline* do microcontrolador ou, no caso de um processador com múltiplos núcleos (como é o caso do Raspberry Pi), permite a execução e o processamento paralelo.

Figura 4.5 – Captura de usuário e senha enviados para um conexão a um *broker* local.



Fonte: Do Autor (2020).

Por fim, o último ataque realizado foi a fim de capturar o pacote de conexão do MQTT. Nesse ataque, foi simulada uma reinicialização do nó IoT para que todo o sistema reiniciasse. Logo, a conexão com o *broker* seria solicitada nova-

mente. Nesse momento, foi feito o mesmo ataque de ARP *Spoofing* que capturou os pacotes enviados e, conforme pode ser observado na Figura 4.5, o pacote com as credenciais de acesso foram obtidos com sucesso. Vale ressaltar que, nesse caso, apenas uma conexão segura com TLS ou SSL poderia prevenir esse tipo de ataque (técnicas que não são possíveis no Arduíno UNO pela limitação do seu hardware). Pode ser estudada uma forma de se modificar ambos o software do *broker* e do cliente MQTT para suportar a conexão criptografada nativamente e sem o uso do SSL e do TLS. Entretanto, esse não é o foco deste trabalho e seriam tópicos para serem desenvolvidos em outros trabalhos.

4.2 Métricas relacionadas ao uso de memória

Uma vez que os ataques foram executados e a eficiência dos métodos foi descrita em termos da criptografia e da capacidade de se ocultar informações, é necessário analisar outros fatores como o consumo de recursos de hardware, sejam esses de memória ou tempo de execução, conforme descrito nas Seções 3.4.1 e 3.4.2 no Capítulo 3. Primeiramente, será analisado o consumo de memória em cada um dos dispositivos (representado pela Tabela 4.1).

Tabela 4.1 – Valor em bytes do uso de memória.

Algoritmo/Modo	**Arduíno UNO R3**	**ESP-8266**	**Rasp. Pi (RAM)**
Memoria Total (em bytes)	**32256**	**1044464**	**927000000**
Sem criptografia	15842	276764	12951000
AES 128 - ECB	21766	278864	13024000
AES 128 - CBC	22056	279492	13076000
AES 128 - CTR	22010	279376	13056000
Percentual correspondente			
Sem Criptografia	49,11%	26,50%	1,40%
AES 128 - ECB	67,48%	26,70%	1,40%
AES 128 - CBC	68,38%	26,76%	1,41%
AES 128 - CTR	68,24%	26,75%	1,41%

Fonte: Do Autor (2020).

A Tabela 4.1 mostra, na segunda linha, o valor total da memória em bytes de cada um dos dispositivos usados. Um pequena diferença que deve ser ressaltada é que, no caso do Raspberry Pi, a memória considerada é RAM, visto que é uma memória de tamanho fixo (e que não pode ser aumentada de forma trivial) e que é a memória onde o sistema operacional aloca o processo para a execução. Além disso, deve-se levar em conta que a memória secundária do Raspberry é feita com uso de um cartão de memória micro SD, que pode ser aumentada e elevada facilmente. Dito isso, no caso do ESP-8266 e do Arduíno foi considerada a memória de programa, ou seja, a memória na qual o programa compilado (em termos das instruções e de variáveis alocadas de forma estática, portanto, nesse cenário a alocação dinâmica de memória é considerada nos cálculos) é armazenado para posteriormente ser executado.

Outro ponto a ser considerado é a estrutura do programa de testes, que tinha a finalidade apenas de criptografar um texto, enviar pela rede e descriptografar qualquer mensagem recebida através do MQTT. Pela Tabela 4.1, percebe-se que, como esperado, o algoritmo sem criptografia usou menos recursos de memória em todos os casos, mesmo que com uma diferença percentual mínima, como foi o caso do Raspberry Pi.

Ao analisar as métricas com as proposições citadas, pode-se perceber que, no caso do Arduino, a adição da parte de criptografia do código aumentaria no mínimo 18% do consumo da memória de programa (que é o caso do ECB que consome menos recursos), o que faz o total de memória usado por um programa simples que apenas envia um texto e não captura dados de nenhum sensor chega a 67% do total de memória disponível, o que deixa poucos recursos restantes para um aplicação mais robusta e com mais entradas e saídas. Além disso, o ECB que é um modo que não provê nenhuma forma de se tratar mensagens maiores de 16 bytes nativamente, iria requerer um código adicional para tratar os blocos e adicionar *padding*, se necessário, aumentando assim a verbosidade e a complexidade de

manutenção, mesmo que pouco. Isso, alinhado com as vulnerabilidades do ECB, torna o método inviável, visto que técnicas mais robustas ocupariam somente 1% a mais dos recursos de hardware. O CTR, que é um método que iria prover todos os recursos necessários, se torna uma alternativa mais atrativa. No entanto, deve-se ponderar antes de mais nada qual a finalidade e complexidade da aplicação antes de se escolher o Arduíno na sua versão Uno. Vale ressaltar que existem outras versões do Arduíno que provêm mais recursos de hardware e, talvez, mesmo para os casos supracitados, o Arduíno ainda seja uma opção, tendo que ser feita uma análise de custo *versus* recursos de hardware e opções disponíveis no mercado.

O ESP-8266, em geral, não tem muito dos seus recursos de hardware consumidos como pode ser observado na Tabela 4.1. Esse se torna, então, uma alternativa interessante quando o Arduíno deixa de ser viável. Porém, deve-se considerar o consumo de energia de uma rede *wireless* e a disponibilidade da mesma. Por fim, em se tratando do Raspberry, vemos que a memória RAM é utilizada minimamente, tendo quase que nenhum impacto nos recursos de hardware como um todo. No entanto, assim como no caso do ESP-8266, algumas considerações devem ser feitas. Como mencionado em 2, o Raspberry é tido por muitos como um computador completo e, apesar do uso arquitetura RISC, o Raspberry possui uma Arquitetura do Conjunto de Instruções (ISA - *Instruction Set Architecture*) bem mais robusta (contando até com um processador de 64 bits), o que torna seu hardware mais completo elevando assim o custo e o consumo de energia. Logo, devem ser feitas algumas ponderações principais:

- Um programa que utilizará apenas 1% dos recursos de hardware realmente requer um dispositivo com esse poder de processamento e de recursos computacionais?

- O sistema IoT como um todo necessitará de portabilidade e alimentação por bateria?

- Qual o gasto esperado com essa arquitetura como um todo?

Essas ponderações vão auxiliar na tomada de decisão porque a aquisição de um dispositivo com hardware que não será utilizados em sua maioria é um desperdício de recursos, e o mesmo vale para a necessidade de bateria e portabilidade que, dependendo do caso, pode aumentar consideravelmente o valor final do produto. Um exemplo que ilustra melhor essa questão, segundo o site oficial (**??**), é o crescimento de preço do Raspberry Pi 4 em relação ao tamanho da memória RAM, que aumenta de R$250,00 na sua versão mais básica (com 2GB) para R$550,00 na sua versão com maior memória (8GB).

Por fim, considerando as características de cada um dos métodos em termos dos recursos fornecidos, da necessidade de implementações adicionais e da memória gasta, o melhor modo de operação para o uso do AES seria o CTR, visto que a definição do modo provê recursos para ser paralelizável e corrige vulnerabilidades presentes nos outros modos, consumindo praticamente o mesmo tanto de memória.

4.3 Métricas relacionadas a desempenho

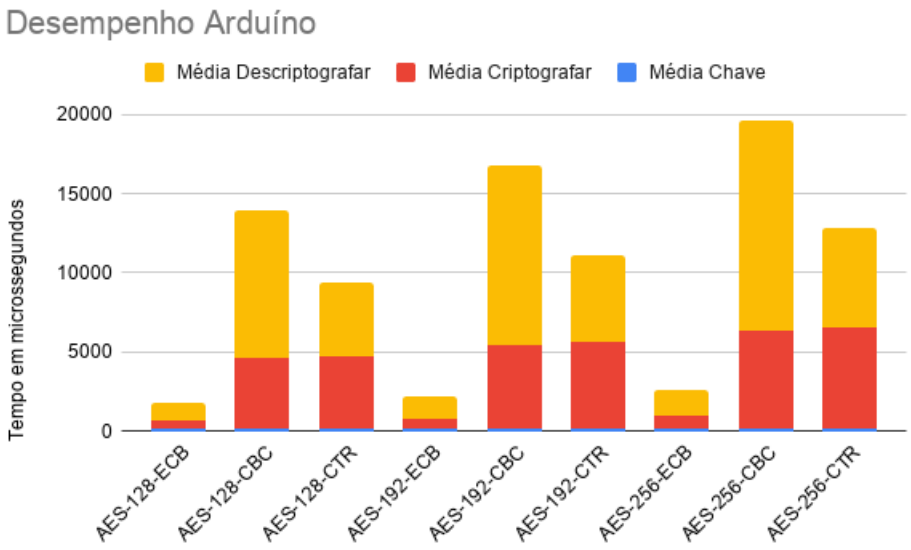
Na Seção 3.4.1 do Capítulo 3, foi descrito um pouco sobre como a configuração de hardware do Raspberry afetou no método de coleta dos resultados e, na Seção anterior, os exemplos e testes realizados foram feitos somente com o AES 128 (variando o modo de bloco). Isso ocorreu por que o aumento no consumo de memória, nesse caso, estaria relacionado com a alocação dinâmica de memória, valor que não foi foco deste trabalho e que pode variar muito dependendo de como o algoritmo foi implementado. Outro ponto a ser ressaltado antes de se prosseguir com as análises é a ocultação do valor do desvio padrão para o tempo de descriptografar as mensagens nas Tabelas 4.2 e 4.3 devido ao valor do desvio padrão ser zero para todas as entradas na coluna em questão.

Tabela 4.2 – Estatísticas da métrica de tempo em μs para Arduíno Uno.

Algoritmo/Modo	$\bar{X}$ **Chave**	σ**Chave**	$\bar{X}$ **Cript.**	σ**Cript.**	$\bar{X}$ **Descrip.**
AES-128-ECB	**142,26**	0,00632	532,785	0,00707	1154
AES-128-CBC	157,98	0,00000	4428,65	0,01054	9400
AES-128-CTR	157,98	0,00000	4610,934	0,00516	4611
AES-192-ECB	156,60	0,00632	639,565	0,00527	1399
AES-192-CBC	172,32	0,00000	5282,81	0,00943	11358
AES-192-CTR	172,32	0,00000	5465,1	0,00000	5465
AES-256-ECB	191,75	0,00632	746,332	0,00919	1643
AES-256-CBC	207,47	0,00000	6136,974	0,00699	13316
AES-256-CTR	**207,47**	0,00000	6319,268	0,00422	6320

Fonte: Do Autor (2020).

Figura 4.6 – Gráfico estatísticas da métrica de tempo em μs para Arduíno Uno.



Fonte: Do Autor (2020).

Em se tratando dos valores para o Arduíno Uno, como pode-se observar na Tabela 4.2 e no gráfico da Figura 4.6, percebe-se que o tempo para qualquer uma das operações aumenta proporcionalmente à complexidade do algoritmo, saindo de aproximadamente 142 microssegundos no modo ECB com 128 bits de chave para 207 microssegundos no modo CTR com 256 bits de chave (valores destacados em negrito). Considerando apenas a operação de setar chave, nota-se que o

paralelismo obtido pelo modo CTR não favorece em nada a operação em questão, visto que os valores são os mesmos. Entretanto, esse recurso se torna mais interessante quando comparamos os valores de criptografia e descriptografia nos modos CBC e CTR. Ao se fazer isso, pode-se perceber que o tempo de se descriptografar uma mensagem no CTR é aproximadamente o mesmo tempo de criptografar uma mensagem no mesmo modo, enquanto que no modo de operação CBC, ao fazer essa mesma comparação, pode-se observar que o tempo mais que dobra de uma operação para outra.

Vale ressaltar também que, no Arduíno Uno, os valores de criptografia no modos CBC e CTR ficaram muito próximos, porém o modo CTR é alguns microssegundos mais lento, o que é curioso considerando sua capacidade de paralelismo da técnica e independência entre os blocos. Isso pode decorrer devido a limitações no ATMega328P relacionadas com *pipeline* e recursos de paralelismo, características que não foram encontradas na documentação da biblioteca usada. Seria necessário um estudo mais aprofundado na arquitetura do microcontrolador, em como o código é gerado pelo compilador e na própria estrutura da biblioteca de criptografia para entender essa situação, o que não é o foco deste trabalho.

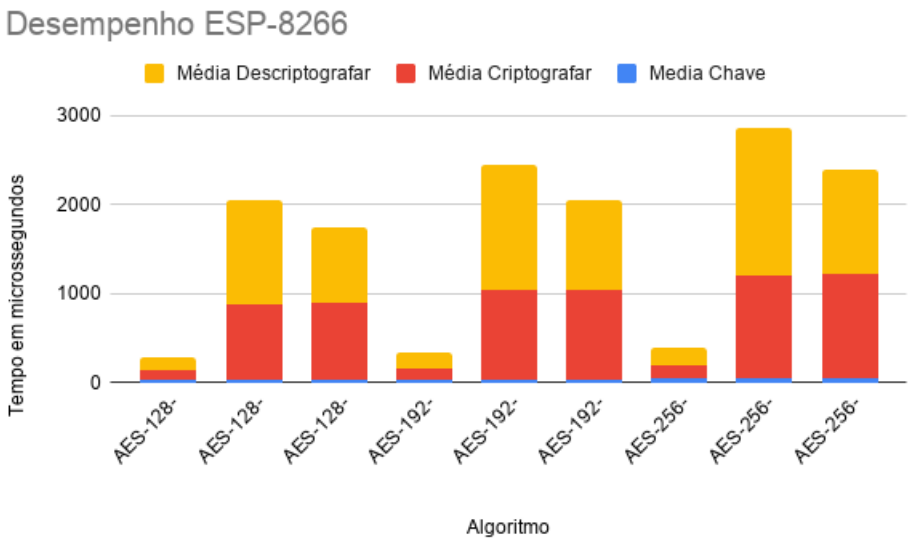
Tabela 4.3 – Estatísticas da métrica de tempo em μs para ESP-8266.

Algoritmo/Modo	$\bar{X}$ **Chave**	σ**Chave**	$\bar{X}$ **Cript.**	σ**Cript.**	$\bar{X}$ **Descrip.**
AES-128-ECB	**34,611**	**0,00316**	**100,662**	**0,06179**	**144**
AES-128-CBC	36,6	0,00000	833,906	0,01838	1177
AES-128-CTR	36,602	0,00422	850,21	0,03300	850
AES-192-ECB	33,358	0,00632	120,839	0,06064	174
AES-192-CBC	35,351	0,00316	995,319	0,01912	1416
AES-192-CTR	35,35	0,00000	1011,59	0,00816	1011
AES-256-ECB	43,722	0,00632	141,023	0,05716	204
AES-256-CBC	45,7	0,00471	1156,681	0,04358	1655
AES-256-CTR	45,705	0,00850	1174,486	0,12267	1174

Fonte: Do Autor (2020).

A respeito do ESP-8266, pode-se ver, conforme a Tabela 4.3 e o gráfico da Figura 4.7, que a placa obteve resultados bem melhores em termos do tempo

Figura 4.7 – Gráfico estatísticas da métrica de tempo em μs para o ESP-8266.



Fonte: Do Autor (2020).

de execução, se comparada com o Uno. De modo geral, a configuração e o padrão dos resultados se manteve, apenas com a mudança no tempo e na proporção dos valores, que, nesse caso, cada etapa do processo de criptografia foi executada mais rapidamente. Isso já era esperado considerando que o microcontrolador do ESP trabalha em uma frequência de *clock* cinco vezes maior, o que, consequentemente, torna as operações mais rápidas. No desvio padrão dos dados, os valores, apesar de uma certa divergência, estão muito próximos de zero chegando a até três casa decimais reforçando ainda mais a tendência da distribuição dos dados.

Considerando o aumento de cinco vezes na taxa de *clock* entre as duas placas, nota-se ainda que, dividindo os valores do tempo para se encriptar a mensagem do Arduíno pelo tempo do ESP, é obtido em média um decréscimo de cinco vezes, o que mostra uma certa proporcionalidade da frequência de *clock* das placas com o tempo para se realizar as operações (representado pela Tabela 4.4). Além disso, pode-se destacar um aumento ainda mais significativo no processo de descriptografia. Ao se analisar os quadros com o tempo de cada um dos dispositivos,

Tabela 4.4 – Taxa de redução do tempo de execução do ESP em relação ao Arduíno UNO para cada uma das etapas do algoritmo: geração de chave, criptografia e descriptografia.

Modo do Algoritmo	**Chave**	**Criptografar**	**Descriptografar**
AES-128-ECB	4,11	5,29	8,01
AES-128-CBC	4,32	5,31	7,99
AES-128-CTR	4,32	5,42	5,42
AES-192-ECB	4,69	5,29	8,04
AES-192-CBC	4,87	5,31	8,02
AES-192-CTR	4,87	5,40	5,41
AES-256-ECB	4,39	5,29	8,05
AES-256-CBC	4,54	5,31	8,05
AES-256-CTR	4,54	5,38	5,38
Média	4,52	5,33	7,15
Desvio Padrão	0,26	0,05	1,31

Fonte: Do Autor (2020).

observa-se também que, quanto maior o tempo gasto pela operação, maior será o ganho em função do aumento da frequência de *clock*, salvo casos como o do método CTR que gastam o mesmo tempo em ambas operações (de criptografia e descriptografia).

No Raspberry Pi as coisas se tornam um pouco diferentes em alguns pontos. A principal diferença é que o Raspberry Pi 3 Modelo B tem um processador ARM de 64 bits com 4 núcleos e 1.2GHz de *clock*. Apesar da capacidade de paralelismo em grande parte dos algoritmos nos modos de operações, não foi encontrado nada na documentação da biblioteca que faça de forma nativa o processamento paralelo dos algoritmos que trabalham melhor com esse recursos. Logo, o *clock* de 1.2GHz faz com que o tempo para executar cada uma das etapas do processo de criptografia seja reduzido consideravelmente e, no caso de alguma modificação para se implementar o processamento paralelo, espera-se que esse tempo seja reduzido ainda mais. A Tabela 4.5 mostra os resultados em notação científica para facilitar a exibição e visualização dos dados.

Na Tabela 4.5 e no gráfico da Figura 4.8, nota-se o impacto do hardware na velocidade de execução do algoritmo. No dispositivo com menor poder compu-

tacional - o Arduíno Uno - obtivemos um valor de aproximadamente 532 microssegundos por bloco (μ/bloco) para criptografar um texto no AES de 128 bits com modo de operação ECB, enquanto que no Raspberry o valor foi de $5,46x10^{-6}$ (ou 0,00000546) microssegundos por bloco para o mesmo algoritmo e modo, resultado quase $1x10^{8}$% (ou 100.000.000%) mais rápido que o citado anteriormente.

Um ponto de importante relevância é a capacidade do Raspberry em trabalhar com conexões TLS ou SSL, o que torna esse processo de criptografia do *payload* totalmente desnecessário em um ambiente onde só existem conexões seguras com o *broker*. No entanto, o Mosquitto permite conexões seguras e inseguras no mesmo ambiente, portanto a criptografia do *payload* pode ser interessante em casos onde um nó não possui recursos de hardware suficientes para se estabelecer uma conexão segura.

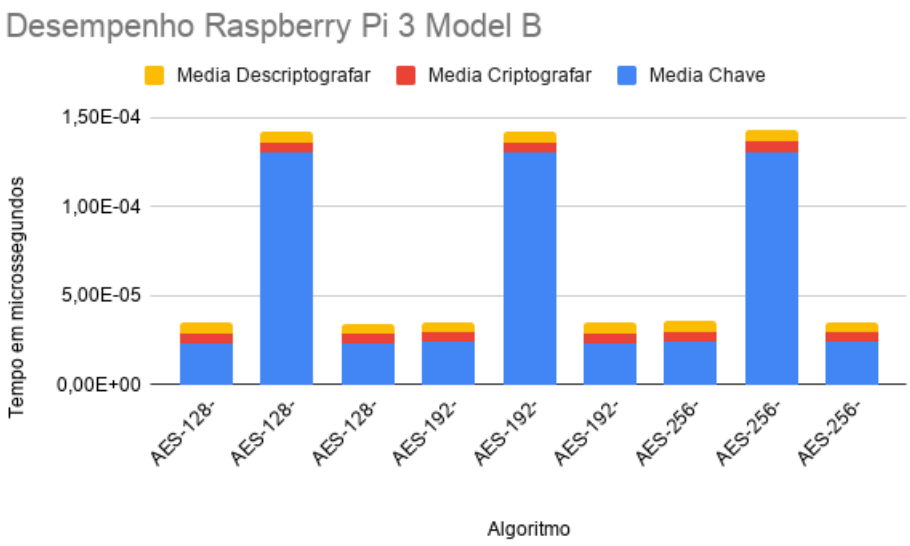
Partindo de uma análise que relacione o pacote MQTT com o *payload* criptografado com a métricas de redes, percebe-se dois cenários principais. No primeiro cenário, o tamanho da mensagem a ser criptografada é múltiplo de 16 e, por isso, não será necessária nenhuma técnica de *padding*, o que resultará em um texto cifrado de mesmo tamanho da mensagem original, não alterando a latência de rede em termos do tamanho do pacote. No segundo cenário, se a mensagem original não tem um tamanho múltiplo de 16, o texto puro precisará ser incrementado até que o tamanho seja o próximo valor múltiplo de 16. Isso aumentará o tamanho da mensagem e afetará diretamente a latência de rede, mesmo que minimamente.

Tabela 4.5 – Estatísticas da métrica de tempo em μs para Raspberry Pi 3.

Algoritmo/ Modo de Operação	**Média Chave**	**Desvio Padrão Chave**	**Média Criptografar**	**Desvio Padrão Criptografar**	**Média Descriptografar**	**Desvio Padrão Descriptografar**
AES-128/CBC	$2,34x10^{-5}$	$4,24x10^{-8}$	$5,46x10^{-6}$	$2,61x10^{-8}$	$5,79x10^{-6}$	$2,31x10^{-8}$
AES-128/CTR	$1,30x10^{-4}$	$1,42x10^{-7}$	$5,83x10^{-6}$	$2,61x10^{-8}$	$5,97x10^{-6}$	$3,03x10^{-8}$
AES-128/ECB	$2,31x10^{-5}$	$5,29x10^{-8}$	$5,38x10^{-6}$	$3,11x10^{-8}$	$5,67x10^{-6}$	$3,45x10^{-8}$
AES-192/CBC	$2,37x10^{-5}$	$2,19x10^{-8}$	$5,56x10^{-6}$	$2,54x10^{-8}$	$5,92x10^{-6}$	$1,39x10^{-8}$
AES-192/CTR	$1,30x10^{-4}$	$1,32x10^{-7}$	$5,95x10^{-6}$	$2,36x10^{-8}$	$6,09x10^{-6}$	$2,26x10^{-8}$
AES-192/ECB	$2,33x10^{-5}$	$2,98x10^{-8}$	$5,49x10^{-6}$	$3,94x10^{-8}$	$5,74x10^{-6}$	$1,92x10^{-8}$
AES-256/CBC	$2,40x10^{-5}$	$2,10x10^{-8}$	$5,68x10^{-6}$	$2,32x10^{-8}$	$6,02x10^{-6}$	$2,11x10^{-8}$
AES-256/CTR	$1,31x10^{-4}$	$1,59x10^{-7}$	$6,06x10^{-6}$	$2,19x10^{-8}$	$6,21x10^{-6}$	$2,75x10^{-8}$
AES-256/ECB	$2,36x10^{-5}$	$4,81x10^{-8}$	$5,61x10^{-6}$	$3,52x10^{-8}$	$5,87x10^{-6}$	$2,14x10^{-8}$

Fonte: Do Autor (2020).

Figura 4.8 – Gráfico estatísticas da métrica de tempo em μs para Raspberry Pi 3.



Fonte: Do Autor (2020).

5 CONCLUSÃO E TRABALHOS FUTUROS

Este Capítulo apresenta as conclusões das atividades realizadas neste trabalho e fala sobre trabalhos futuros com base nas conclusões obtidas.

5.1 Conclusões

A grande abrangência e diversidade tecnológica associada à Internet das Coisas mostra a necessidade de se aprimorar e reinventar as técnicas e tecnologias existentes atualmente a fim de se atender a esse nova rede de dispositivos conectados. Neste trabalho, o objetivo foi avaliar a viabilidade, o impacto, vantagens e desvantagens da utilização de algoritmos de criptografia, como AES com diferentes modos de operação de cifra, nas placas NodeMCU, Arduíno UNO e Raspberry Pi, comumente usadas em aplicações IoT. O intuito foi adotar uma abordagem que garantisse alguns dos princípios da segurança da informação e, ao mesmo tempo, fosse acessível para dispositivos mais limitados e de baixo custo. No entanto, as limitações existentes também foram analisadas, tendo como exemplo as credenciais de acesso para conexão ao *broker* que foram descobertas no ataque. Por isso, com os dados de desempenho obtidos, buscamos contribuir para a análise de algoritmos de criptografia em sistemas embarcados e dispositivos IoT e mostramos as limitações do MQTT nesse quesito, com o objetivo de se chamar a atenção para a necessidade de se adaptar e evoluir as técnicas de criptografia a fim de melhorar seu funcionamento com dispositivos de baixo consumo de energia e com menos recursos computacionais.

Com a criptografia do *payload*, foi possível garantir os seguintes princípios da segurança da informação:

- Confidencialidade: com a criptografia do *payload*, apenas os nós finais poderão ter acesso a informação. Nem mesmo o *broker* consegue descriptografar as informações;

- Integridade: caso as mensagens criptografadas sejam alteradas, os dados não poderiam ser descriptografados, seja de forma integral ou parcial, dependendo do modo de cifra, o que de certa forma permitiria a identificação da alteração.

A solução desenvolvida e testada, no entanto, não garante:

- Autenticidade: mesmo com a criptografia dos dados, não se pode garantir que o remetente é legítimo, visto que a técnica não possui nenhuma forma de assinatura.

Além disso, vale ressaltar que, como já citado, essa técnica pode ser usada em sistemas mistos, onde o uso da conexão segura é ou não permitido. Entretanto, seu uso só é recomendado em casos de dispositivos que não têm recursos para uma conexão SSL ou TLS, como o Arduíno Uno. Deve-se ressaltar também que em casos onde a mensagem foi alterada ou modificada por um atacante, o conteúdo não poderia ser descriptografado, necessitando assim de uma retransmissão da informação. No mais, cabe ao arquiteto da rede ponderar qual é a melhor opção de acordo com o sistema IoT a ser desenvolvido, escolhendo e ponderando se deve ou não utilizar esse método.

5.2 Trabalhos Futuros

No âmbito da Internet das Coisas, as possibilidades são ilimitadas e continuam crescendo a cada dia. Assim, o estudo feito neste trabalho ajuda a definir algumas opções:

- Modificação dos softwares que implementam o MQTT - Mosquitto e PubSubClient, que são de código aberto - a fim de suportar nativamente conexões criptografadas com uma alternativa *lightweight* ao SSL e ao TLS, e que funcione em dispositivos como o Arduíno Uno;

- Estudo da relação entre configurações de rede e *firewall* e como também são uma alternativa para evitar ataques do tipo *arpspoof*;
- Percebe-se, neste trabalho, que a capacidade de se trabalhar com criptografia até mesmo nos dispositivos com menos recursos é satisfatória, logo, pode ser criada uma variante da rede Tor para o MQTT que adiciona várias camadas de criptografia AES ao pacote.

REFERÊNCIAS

ALECRIM, E. **O que é Internet das Coisas (`Internet of Things`)?** 2017. Disponível em: <https://www.infowester.com/iot.php>. Acesso em: 19 ago. 2020.

ANDRADE, F. C. d. **Uma abordagem leve e segura para comunicação utilizando o protocolo MQTT em dispositivos IoT**. 89 f. Monografia — DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO, Universidade de São Paulo, São Paulo, 2017.

ANDY, S.; RAHARDJO, B.; HANINDHITO, B. Attack scenarios and security analysis of mqtt communication protocol in iot system. In: **2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)**. [S.l.: s.n.], 2017. p. 1–6.

ARDUÍNO COMMUNITY. **Arduíno Store Website**. 2020. Disponível em: <https://store.arduino.cc/usa/arduino-uno-rev3>. Acesso em: 10 jul. 2020.

ARPSPOOF. 2016. Disponível em: <https://github.com/smikims/arpspoof>. Acesso em: 18 ago. 2020.

CHESWICK, W. R.; BELLOVIN, S. M.; RUBIN, A. D. **Firewall e segurança na Internet: repelindo o hacker ardiloso**. 2. ed. [S.l.]: Bookman, 2005.

COMBS, G. **About Wireshark**. 1998. Disponível em: <https://www.wireshark.org/>. Acesso em: 07 jul. 2020.

COUTINHO, S. C. **Números inteiros e criptografia RSA**. 2. ed. [S.l.]: IMPA, 2014.

CRAGGS, I.; DASGUPTA, R.; PAGLIUGHI, F. **Eclipse Paho**. 2020. Disponível em: <https://projects.eclipse.org/projects/iot.paho/>. Acesso em: 17 ago. 2020.

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

ECLIPSE FOUNDATION. **Eclipse Mosquitto: An open source MQTT broker**. 2020. Disponível em: <https://mosquitto.org/>. Acesso em: 02 jul. 2020.

HOSSAIN, R. H. M.; SKJELLUM, A. Securing the internet of things: A meta-study of challenges, approaches, and open problems. In: IEEE (Ed.). **Distributed Computing Systems Workshops (ICDCSW)**. [S.l.: s.n.], 2017. p. 220–225.

JADHAV, D.; KULKARNI, N.; SWAMY, S. N. Security threats in the application layer in iot applications. In: **2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)**. [S.l.: s.n.], 2017. p. 477–480.

KPIZINGUI, M. **Demystifying the MQTT maze: clients, servers, connection, publish, subscribe and its applications.** 2017. Disponível em: <https://morioh.com/p/93ba2353480e>. Acesso em: 01 jul. 2020.

KUROSE, J.; ROSS, K. **Redes de computadores e a internet: uma abordagem top-down**. 6. ed. [S.l.]: Pearson Education do Brasil, 2013.

MATHEWS, S. P.; GONDKAR, R. R. Protocol recommendation for message encryption in mqtt. In: **2019 International Conference on Data Science and Communication (IconDSC)**. [S.l.: s.n.], 2019. p. 1–5.

MICROSOFT CORPORATION. **Visual Studio Code**. 2020. Disponível em: <https://code.visualstudio.com/docs>. Acesso em: 15 ago. 2020.

MORAIS, I. S. de. **Introdução a Big Data e Internet das Coisas (IoT)**. [S.l.]: SAGAH, 2018. Recurso Eletrônico.

MOZILLA COMMUNITY. **Um visão geral do HTTP**. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 15 ago. 2020.

NAIK, S.; MARAL, V. Cyber security — iot. In: **2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)**. [S.l.: s.n.], 2017. p. 764–767.

NUNES, R. O. **Man In The Middle**. 2017. Disponível em: <https://www.bebee.com/producer/@renata-oliveira-nunes-06wVjc/man-in-the-middle>. Acesso em: 01 jul. 2020.

OASIS COMMITTEE SPECIFICATION DRAFT. **MQTT Version 3.1.1**. 2. ed. [S.l.], 2014. 137 p. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html>.

OWASP FOUNDATION. **Open Web Application Security Project**. 2020. Disponível em: <https://owasp.org/>. Acesso em: 20 ago. 2020.

O'LEARY, N. **Arduino Client for MQTT**. 2020. Disponível em: <https://pubsubclient.knolleary.net/>. Acesso em: 10 jul. 2020.

PlatformIO COMMUNITY. **PlatformIO Website**. 2020. Disponível em: <https://docs.platformio.org/en/latest/what-is-platformio.html>. Acesso em: 15 ago. 2020.

Random Nerd Tutorials. **ESP8266 Pinout Reference: Which GPIO pins should you use?** 2020. Disponível em: <https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/>. Acesso em: 11 jul. 2020.

RASPBERRY PI FOUNDANTION. **Raspberry GPIO**. 2020. Disponível em: <https://www.raspberrypi.org/documentation/usage/gpio/>. Acesso em: 11 jul. 2020.