UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Uma abordagem leve e segura para comunicação utilizando o protocolo MQTT em dispositivos IoT

Autor: Fernando Camargo de Andrade

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2017

Fernando Camargo de Andrade

Uma abordagem leve e segura para comunicação utilizando o protocolo MQTT em dispositivos IoT

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia de Computação

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos
2017

Andrade, Fernando Camargo de
A553u Uma abordagem leve e segura para comunicação utilizando o protocolo MQTT em dispositivos IoT. / Fernando Camargo de Andrade; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2017.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos e Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2017.

1. Internet das Coisas. 2. MQTT. 3. Criptografia. 4. Comunicação Máquina-Nuvem. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fernando Camargo de Andrade

Título: "Uma abordagem leve e segura para comunicação utilizando o protocolo MQTT em dispositivos IoT"

Trabalho de Conclusão de Curso defendido em 30/11/2017.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luis Linhari Rodrigues (Orientador) - SEL/EESC/USP	APROVADO
Profa. Associada Kalinka Regina Lucas Jaquie Castelo Branco SSC/ICMC/USP	Aprovado
Mestre André Luis Martins Doutorando - SEL/EESC/USP	Aprovado.

Coordenador do Curso Interunidades Engenharia de Computação:

Prof. Dr. Maximiliam Luppe

Dedicatória

Dedico este trabalho em memória da minha avó paterna Maria Lorette de Andrade, que sempre desejou que eu fizesse faculdade na Universidade de São Paulo, me estimulando desde pequeno à alcançar este objetivo, e, que mesmo com uma saúde limitada, se manteve firme durante toda a fase de vestibulares, mas que, infelizmente, nos deixou dias antes de sair a divulgação da lista de aprovados. Mas que mesmo de longe sempre me deu forças para continuar em momentos adversos.

Fernando Camargo de Andrade.

Agradecimentos

Agradeço, primeiramente, aos meus pais, Fernando e Cristina, e à minha irmã Carolina, por todo o apoio durante os meus anos na faculdade, por sempre me incentivarem a estudar para alcançar uma formação acadêmica de qualidade e por me ensinarem a batalhar por meu projetos.

Agradeço a minha tia Cida pelo suporte, não só nestes últimos anos de faculdade, mas também durante toda minha vida, e por sempre me lembrar que a alimentação de um universitário não se limita apenas ao cardápio do bandejão, mas também às frutas frescas vindas da feira. Agradeço aos meus avós maternos, José e Leonor, pelo orgulho que eles sentem em ter o neto estudando em outra cidade, mesmo não se lembrando do nome do curso, embora nunca esquecendo da data do meu aniversário.

Agradeço à minha namorada Eloísa e aos seus pais, Josemar e Luciana, pela paciência, principalmente nos momentos mais desgastantes, e por entenderem que as ausências em aniversários e churrascos de família não eram por brigas ou por término de relacionamento, mas sim por compromissos com a graduação. Agradeço mais uma vez a Eloísa por financiar os gastos de impressão deste material para que a sua apresentação fosse impecável, e pelas hospedagens sem nenhum custo em São Carlos.

Agradeço aos amigos Gustavo, por não se importar em jogar em monitor de baixa resolução e emprestar seu monitor de 22 polegadas para que eu fizesse meu trabalho, e Pedrinho, por disponibilizar um escritório exclusivo para que pudesse me concentrar nos estudos. Agradeço ao meu Orientador Prof. Dr. Evandro Luis Linhari Rodrigues por aguentar um orientando muitas vezes perdido e por saber me direcionar à convergir minhas ideias e chegar na definição do tema do trabalho. Por fim, agradeço à Universidade de São Paulo por toda a infraestrutura fornecida para uma educação de qualidade e ao seu corpo docente por todo o conhecimento compartilhado nestes anos.

Fernando Camargo de Andrade.

"Security is a process, not a product."

[Bruce Schneier]

Resumo

Com a forte chegada do conceito da Internet das Coisas(IoT), em que a internet é levada à objetos do cotidiano das pessoas, da indústria e até na infraestrutura de meios urbanos e onde há a constante troca de informações entre todos estes dispositivos, a comunicação máquina-nuvem se torna extremamente relevante e alguns protocolos para este propósito começaram a ganhar destaque por se encaixarem nos requisitos de recursos limitados, geralmente visto em aparelhos de IoT. Entre estes protocolos existe o Message Queue Telemetry Transport (MQTT), que por meio da estrutura de inscrições e publicações em tópicos, realiza a comunicação entre 2 dispositivos. Porém, a segurança ainda é de grande importância na comunicação para evitar que pessoas indesejadas leiam as mensagens. Por isto, este trabalho realiza a implementação de formas de comunicação criptografadas, que ofereçam segurança para as mensagens enviadas e não descaracterize a leveza oferecida pelo protocolo. Para os resultados foram calculados o tempo necessário para o processamento dos dados e a quantidade de informações adicionais exigidos pela solução. No geral, foram adicionados 3 bytes por item criptografado, além de somar aproximadamente 1 ms de processamento para cada byte criptografado em dispositivos com recursos computacionais reduzidos.

Palavras-Chave: Internet das Coisas, MQTT, criptografia, comunicação máquina-nuvem.

Abstract

With the arrival of the concept of the Internet of Things (IoT), in which the Internet is taken to the everyday objects of the person, to the processes of the industry and even in the infrastructure of urban means and where there is the constant exchange of information between all these devices, machine-to-cloud communication becomes extremely relevant and some protocols for this purpose have begun to gain prominence by meeting the limited resource requirements generally seen in IoT devices. Among these protocols there is the Message Queue Telemetry Transport (MQTT), which through the structure of subscriptions and publications in topics, makes the communication between two devices. However, security is still of great importance in communicating to prevent unwanted people from reading the messages. Therefore, this work implements an encrypted form of communication that provides security for the messages sent and does not detract from the lightness offered by the protocol. For the results, the time needed to process the data and the amount of additional information required by the solution was calculated. Overall, 3 bytes per encrypted item were added, as well as adding approximately 1 ms of processing for each encrypted byte on devices with constrained computacional resources.

Keywords: Internet of Things, MQTT, Encryption, Machine-to-Cloud Communication.

Lista de Figuras

Lista de Tabelas

Siglas

ACL	*Acess Control List* - Lista de Acesso de Controle
AMQP	*Advanced Message Queuing Protocol* - Protocolo Avançado de Enfileiramento de Mensagens
ASCII	*American Standard Code for Information Interchange* - Código Padrão Americano para o Intercâmbio de Informação
CoAP	*Constrained Application Protocol*
CPU	*Central Processing Unit* - Unidade Central de Processamento
DDoS	*Distributed Denial Of Service Attack* - Ataque Distribuído de Negação de Serviço
DNS	*Domain Name System* - Sistema para Nomes de Domínios
GPS	*Global Positioning System* - Sistema de Posicionamento Global
GPU	*Graphics Processing Unit* - Unidade de Processamento Gráfico
GSM	*Global System for Mobile Communications* - Sistema Global para Comunicação Móveis
HTTP	*Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto
IANA	*Internet Assigned Numbers Authority* - Autoridade de Atribuição de Números da Internet
ICMP	*Internet Control Message Protocol* - Protocolo de Controle de Mensagem para Internet
IEEE	*Institute of Electrical and Electronics Engineers* - Instituto de Engenheiro Eletricistas e Eletrônicos
IoT	*Internet of Things* - Internet das Coisas
IP	*Internet Protocol* - Protocolo de Internet
IPSO	*Internet Protocol for Smart Objects* - Protocolo de Internet para Objetos Inteligentes

LTE	*Long Term Evolution* - Evolução de Longo Prazo
M2M	*Machine-to-Machine* - Máquina-à-Máquina
MAC	*Media Access Control* - Controle de Acesso à Mídia
MAC	*Message Authenticaion Code* - Código de Autenticação de Mensagem
MITM	*Man-In-The-Middle*
MQTT	*Message Queue Telemetry Transport*
NFC	*Near Field Communication* - Comunicação por Campos Próximo
OASIS	*Organization for the Advancement of Structured Information Standards* - Organização para o Avanço dos Padrões de Informação Estruturada
QoS	*Quality of Service* - Qualidade do Serviço
RAM	*Random Access Memory* - Memória de Acesso Randômico
RFID	*Radio-Frequency Identification* - Identificação por Rádio Frequência
SSL	*Security Socket Layer*
TCP	*Transmission Control Protocol* - Protocolo de Controle de Transmissão
TLS	*Transport Layer Security* - Segurança na Camada de Transporte
UDP	*User Datagram Protocol* - Protocolo de Datagrama de Usuário
UMTS	*Universal Mobile Telecommunication System* - Sistema Universal de Telecomunicação Móvel
URL	*Uniform Resource Locator* - Localização Padrão de Recurso
UTF-8	*8-bit Unicode Transformation Format* - Formato de Transformação Unicode de 8-bit
WSAN	*Wireless Sensor and Actuator* - Sensor e Atuador sem Fio
XEP	*XMPP Extensions Protocols* - Protocolos de Extensões para XMPP
XML	*Extensible Markup Language* - Linguagem de Marcação Extensível
XMPP	*Extensible Messaging and Presence Protocol* - Protocolo Extensível de Mensagem e Presença

Sumário

Capítulo 1

Introdução

Já é uma realidade que a internet chegou à objetos inanimados, permitindo que eles gerem dados do ambiente, aprendam padrões de comportamento, mudem seus padrões de funcionamento e comuniquem entre si para melhor desempenharem suas atividades. Bilhões de aparelhos já se conectam à internet atualmente e a expectativa é de um crescimento um pouco mais acelerado para os próximos anos, sendo estimado uma média de aproximadamente 3,4 dispositivos por pessoa no mundo [1].

A segurança por outro lado é algo que preocupa de clientes a desenvolvedores. Considerando que os aparelhos irão coletar e transmitir dados que podem ser pessoais, não é desejado que essas informações sejam recebidas por usuários indesejados e mal intencionados. Como os dispositivos são geralmente pequenos e limitados, é necessário que os dados sejam transmitidos de maneira simples e a criptografia da informação, quando utilizada, deve ser leve o suficiente para ser suportada em ambientes de pouco processamento.

Para isto, este trabalho propõe uma solução leve e segura para comunicação *machine-to-cloud* utilizando o protocolo *Message Queue Telemetry Transport* (MQTT) para a Internet das Coisas baseado em aplicações para ambiente doméstico, assim como é feito uma análise de sua eficiência e segurança.

Este Capítulo introduz ao problema e apresenta uma visão geral do trabalho. A seção 1.1 expõe a motivação da realização deste projeto. Os objetivos são dissertados na seção 1.2 e a organização do trabalho é indicada na seção 1.3.

1.1 Motivação

Com a promessa de que aparelhos ajudem nas tarefas do dia-a-dia da população mundial, melhorando a qualidade de vida das pessoas e desenvolvendo ambientes mais sustentáveis e saudáveis, soluções que melhorem o tráfego em centros urbanos, gerencie de forma eficiente a qualidade das plantações, garanta a proteção das residências ou aumente eficácia da utilização dos recursos naturais já estão sendo planejados e introduzidos no mercado.

Em termos técnicos, diversos padrões e protocolos vêm ganhando relevância quando o assunto é IoT. Para definir a arquitetura das redes na qual inúmeros aparelhos estarão conectados e se comunicando, modelos de comunicação, tais como *device-to-device*, *device-to-cloud* e *device-to-gateway*, e protocolos de transmissão leves, por exemplo *Constrained Application Protocol* (CoAP), *Extensible Messaging and Presence Protocol* (XMPP) e MQTT estão sendo discutidos e estudados a fim de definir as aplicações ideais para cada tópico.

Por outro lado, um grande contraponto a esse mundo conectado é em relação à segurança da comunicação e dos dados gerados por estes dispositivos. Como muitas das informações serão transmitidas apenas entre máquinas, sem a intervenção humana, é necessário que os dados se mantenham íntegros durante toda a comunicação e essa informação não deve ser interceptada por usuários indesejados, pois não é de interesse das pessoas que ladrões saibam, através dos aparelhos instalados dentro de uma residência, quando os moradores estão ausentes de suas casas.

Considerando que as aplicações direcionadas à Internet das Coisas (IoT), em que muitos dispositivos são extremamente simples, sendo utilizados para fins muito específicos e com recursos limitados de *hardware*, soluções complexas de proteção da comunicação não são escaláveis para a realidade dos objetos inteligentes. Portanto, soluções leves, que não exijam cálculos complexo nem grandes quantidades de dados trafegados para proteger a comunicação devem ser estudados a fim de viabilizá-los.

Dessa forma, devido à baixa capacidade e velocidade de processamento, à pouca disponibilidade energética, em que muitos sistemas são alimentados por bateria, e à limitação das memórias dos *hardwares*, em comparação com celulares e computadores atuais, é inviabilizado, nos dispositivos de IoT, a utilização de algoritmos de criptografia convencionais que existem nos dias de hoje [2].

1.2 Objetivo

Realizar de forma rápida e leve a criptografia dos dados enviados em uma comunicação MQTT entre o *publisher* ou o *subscriber* e o *broker*, garantindo a confidencialidade, integridade e autenticidade da informação.

1.3 Organização do Trabalho

O Capítulo 2 descreve um panorama sobre a Internet das Coisas, comentando sobre suas características, evolução, casos de uso e desafios a percorrer. Neste Capítulo também é apresentando os padrões e protocolos de comunicação voltados para IoT, além de expor os problemas de segurança enfrentados por este conteúdo.

No Capítulo 3 é exposto todo o material utilizado para realizar as atividades propostas nesta dissertação, sendo evidenciadas as características do protocolo MQTT, seus serviços de cliente e servidor e os dispositivos físicos utilizados na comunicação.

O Capítulo 4 apresenta toda a metodologia utilizada para atingir o objetivo proposto por este trabalho, sendo detalhado as maneiras de dificultar que invasores tenham acesso à dados relevantes dos usuários utilizando uma comunicação segura, com criptografia das mensagens, geração de chaves e aleatorização dos conteúdos.

No Capítulo 5 é divulgado os dados de eficiência da implementação da solução conforme o objetivo desta dissertação. Por fim, no Capítulo 6 são expressas as conclusões inferidas por esta solução.

Capítulo 2

Embasamento Teórico

2.1 A Internet das Coisas

A evolução da internet chegou aos objetos, sendo que atualmente é grande a tendência no mercado a produção de diversos produtos, que antes apenas ficavam passivos ao meio, sendo acionados apenas quando o usuário realizava algum tipo de interação, mas que agora conseguem agir por conta própria, podendo automaticamente mudar seu padrão de funcionamento conforme a necessidade do meio e dos usuários, reagindo as mudanças do ambiente, gerando dados, realizando análises, aprendendo padrões de comportamentos e possuindo comunicação com outros dispositivos e com a Internet. Essa tendência de objetos totalmente conectados ficou conhecida com Internet das Coisas (IoT) [3].

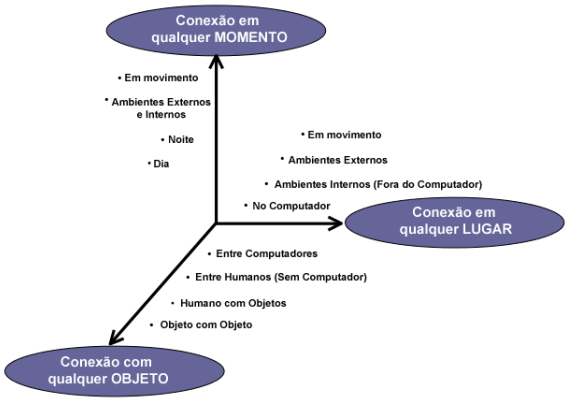
Algumas áreas possíveis de utilizar o conceito da Internet das Coisas são, por exemplo: i) *Previsão de Desastre Naturais*, no qual sensores captarão sinais e, com o estudos dos padrões de comportamentos, esses dados gerados irão prever alguma anomalia em situações diversas e avisar a população com antecedência; ii) *Aplicações Industriais*, aumentando o controle das máquinas e dos processos de produção ampliando a eficiência da indústria e iii) *Controle de Qualidade e Escassez de Água*, por meio de uma rede de sensores é possível monitorar todo o ciclo da água e sua utilização a fim tomar providências para redução de consumo e prevenção de esgotamento do recurso, além de garantir toda a qualidade do serviço que chega à população [4].

Construir ambientes eficientes e inteligentes também é uma área promissora dentro da IoT. Conceitos como iv) *Casas Inteligentes* estão cada vez mais presentes no cotidiano das pessoas. Melhorar o consumo energético, aumentar a segurança dos moradores e adaptar condições do ambiente conforme a necessidade dos moradores são alguns dos exemplos para

aplicar a IoT dentro de uma residência. A ideia de se ter sensores nas ruas, estradas, semáforos e carros, contribui para um v) *Trânsito Inteligente*, em que os dados coletados podem ser utilizados na melhora do fluxos dos veículos e na prevenção de acidentes. As cidades também poderão se beneficiar dos sensores, com o conceito das vi) *Smart Cities*, com uma melhor eficiência de gastos com a iluminação pública, fluxos de pessoas, segurança da população e para estabilização de rotas dinâmicas para serviços de emergência.

Para [5], o paradigma da IoT irá trazer significativas mudanças para a vida cotidiana das pessoas e da indústria, proporcionando também diversas oportunidades de negócios e grandes contribuições para a economia. A comunicação e a conectividade serão elevadas à uma nova dimensão [6], que, além das possibilidades atuais de se conectar a qualquer momento e em praticamente qualquer lugar, com a introdução da Internet das Coisas, todos os objetos terão a possibilidade de ficarem *online*. A figura 2.1 apresentam um gráfico que ilustra as três dimensões da conexão, alcançadas agora por meio da IoT. Desta forma, em um mundo onde humanos, dispositivos eletrônicos e objetos se integrarão, a comunicação entre carros e estacionamentos, implantes e *softwares* de monitoramento da saúde, sensores de umidade e irrigadores, são alguns poucos exemplos da infinidade de possibilidades que serão abertas no âmbito da integração entre sistemas.

Figura 2.1: As três dimensões da conexão.



Fonte: Adaptado de Geneva: International Telecommunication Union (ITU), 2005. [6]

Segundo [7], a IoT é o resultado da convergência de tecnologias relacionadas dentro de grandes concepções, das quais possuem orientação ao objeto e à Internet.

O primeiro conceito envolve todo tipo de padrões que permitam aos objetos serem vistos e identificados, oferecendo opções de acompanhamento em relação as suas condições, características e localização. Tecnologias como *Near Field Communication* (NFC), *Wireless*

Sensor and Actuator Networks (WSAN) and *Radio-Frequency Identification* (RFID) [7] são fundamentais no processo de conexão entre os mundos real e digital.

O campo da orientação à Internet define o estabelecimento de protocolos leves para endereçamento e que levem fácil acesso em qualquer momento e lugar. De uma forma simplificada, padrões de IoT deverão fornecer direcionamento de comunicação para qualquer dispositivo conectado, reduzindo a complexidade do roteamento de endereços atual. Padrões como *Internet Protocol for Smart Objects* (IPSO) e Internet 0 vêm surgindo com propostas que levam à facilitação do endereçamento do Protocolo de Internet (IP).

2.1.1 A IoT no Mundo Atual

Uma pequena amostra de produtos com essas características já podem ser facilmente encontrados e adquiridos no mercado, como por exemplo o *Nest Thermostat* [8] que aprende sozinho o padrão de comportamento e preferências de temperatura dos moradores da casa, podendo, automaticamente, ajustar a temperatura do ambiente conforme o gosto das pessoas e, quando não há ninguém em casa, ele entra em modo econômico, para reduzir o consumo de energia na residência. Todo seu controle também pode ser realizado remotamente via internet e com o aplicativo do celular.

Outro exemplo é a caixa de som *Google Home* [9] que possibilita realizar todo o controle dos objetos inteligentes conectados dentro de casa, realizar pesquisa no Google, planejar tarefas e lembrar compromissos, reproduzir músicas, tudo isso por comando de voz e com conexão à internet para personalizar cada vez mais sua interação com o usuário para deixá-la conforme seu gosto.

Não só a casa é que está ganhando com produtos inteligentes, já faz alguns anos que empresas de dispositivos eletrônicos vende relógios inteligentes que se conectam com a internet e com o celular do usuário, elevando a função de relógio a não apenas mostrar as horas, mas também reproduzir músicas, disponibilizar mensagens e indicar direções com o Sistema de Posicionamento Global (GPS). Um exemplo é o *Apple Watch* [10] que monitora toda a movimentação, as atividades físicas realizada pelo usuário, seu batimento cardíaco, entre outros dados, em prol de incentivar a prática de exercícios e uma boa saúde. Toda a informação gerada é facilmente acessada pelo *smartphone*, e o usuário pode gerar gráficos e relatórios de suas atividades.

Devido às diversas possibilidades de dispositivos que se conectem à internet, foi estimado que em 2015 havia um total de 4,9 bilhões de objetos conectados à internet, não considerando

smartphones, *tablets* e computadores [11]. A mesma pesquisa estima que para 2020 cerca de 20,8 bilhões de dispositivos estarão conectados com a internet. Em termos financeiros, a expectativa é que, também em 2020, o mercado movimente 2 trilhões de dólares na economia mundial [12].

2.1.2 Desafios

Para [4], a IoT trará diversos benefícios à seus usuários, porém sua implementação já enfrenta e ainda deverá lidar com diversos desafios e problemas para garantir a entrega de soluções seguras, em grandes quantidades e de forma totalmente conectada à internet.

De forma resumida, alguns desses desafios podem ser definidos por i) *Identificação*, pois considerando que a estimativa é de bilhões de aparelhos conectados, é necessário, portanto, que cada um possua um identificador único para ser visto por meio da internet, então, sistemas de gerenciamento de usuários deverão ser eficientes o suficiente para lidar com uma quantidade elevada de dispositivos. Outro aspecto é a ii) *Padronização*, já que inúmeros fabricantes desenvolverão distintas soluções para a IoT e elas deverão se comunicar com outros dispositivos de diferentes desenvolvedores, sendo a padronização necessária para a fluidez na troca de informações. Outro aspecto interessante de ser analisado e de suma importância para o contexto da IoT no mundo atual é o iii) *Consumo de Recursos*. O crescimento da quantidade de dispositivos e da infraestrutura para permitir que os serviços funcionem ininterruptamente irá aumentar a necessidade do consumo energético, portanto, os dispositivos deverão ser eficientes em seus gastos com energia [4].

Soluções relacionadas à iv) *Segurança e Privacidade* são amplamente debatidas para garantir que toda informação gerada e enviada ou recebida seja acessada apenas por serviços autorizados a recebê-las. Invasores, tanto do dispositivo quanto do meio de comunicação, pode causar sérios problemas à integridade dos usuários. Para ajudar na proteção, a v) *Criptografia* aparece para garantir a integridade da informação, porém este tópico ainda deve ser muito estudado, visto que determinados padrões de criptografias exigem muitos processamentos, recursos limitantes em pequenos sensores e dispositivos que fazem parte do mundo da IoT [4].

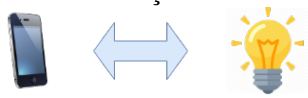
2.2 Padrões de Comunicação para IoT

Devido às novas possibilidades de comunicação oferecida pela conectividade dos objetos na internet, principalmente em relação à comunicação *machine-to-machine* (M2M), em que os dispositivos se comunicam entre si de forma autônoma, foi definida a *"Architectural Considerations in Smart Object Networking"* (RFC7452) que estabelece 4 modelos de comunicação para a Internet das Coisas [13], sendo elas *Device-to-Device*, *Device-to-Cloud*, *Device-to-Gateway* e *Back-End Data Sharing*.

2.2.1 Comunicação *Device-to-device*

Compreende a comunicação entre 2 dispositivos diferentes, que podem ou não ser desenvolvidos pela mesma empresa, e que relacionam-se e comunicam-se diretamente. Um modelo exemplificado pode ser o aplicativo de alarme do celular capaz de acender de forma gradual as luzes do quarto no momento em que o despertador for acionado, conforme figura 2.2.

Figura 2.2: Comunicação *Device-to-Device*.



Fonte: Autoria Própria

Outro modelo para este mesmo tipo de padrão *device-to-device* é utilizado por redes de sensores sem fio desenvolvidos por diversos fabricantes e que se comunicam entre si, sendo que, na maioria dos casos, possuem recursos de memória, energia e processamento limitados. A rede de sensores é formada por diversos dispositivos próximos que se comunicam entre si e com seus vizinhos, conforme pode ser visto na figura 2.3. Como um exemplo, pode monitorar toda a situação das ruas e avenidas das cidades, em relação à trânsito, clima, alagamento e gerar avisos de melhores caminhos a seguir, além de poder alterar comportamento de semáforos conforme o fluxo e a demanda de carros.

Figura 2.3: Rede de sensores com comunicação *Device-to-Device*.

Fonte: https://www.embarcados.com.br/modelos-de- comunicacao-para-iot/ [14]

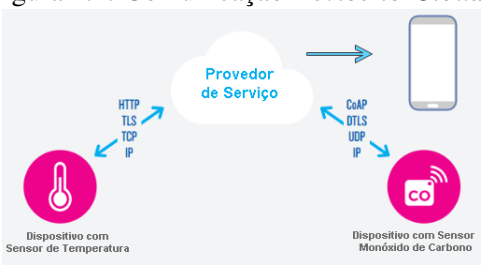
Com o objetivo de que os dispositivos de diversos fabricantes se comuniquem, os fornecedores de aplicativos devem entrar em acordo para padronizar os protocolos de troca de informações que deverá ser usado, tendo em mente algumas questões como, por exemplo, se eles terão suporte à camada física, se permitirão transmissão via rádio de baixa potência (*Bluetooth Smart* ou IEEE 802.15.4). As arquiteturas de rede também deverão ser definidas, analisando quais as restrições dos dispositivos. Na camada de aplicação, será necessário estabelecer qual protocolo será usado, *Message Queue Telemetry Transport* (MQTT), *Constrained Application Protocol* (CoAP) ou outro. Além de garantir toda a segurança e a privacidade da comunicação para assegurar a integridade dos dados do usuário.

2.2.2 Comunicação *Device-to-Cloud*

Toda a comunicação é feita entre o dispositivo de IoT com um provedor remoto na nuvem, ou seja, todos os dados gerados por esse dispositivo são enviados via internet, sem necessidade de um equipamento intermediário, para um servidor, que será responsável por realizar toda a análise e processamento desses dados, para em seguida disponibilizá-los por meio de sites ou aplicativos para acesso dos usuários. Portanto, para que ocorra a comunicação de dispositivos com os serviços prestados pelo servidor, é necessário que haja a definição dos protocolos usados para a transmissão. Existem atualmente diversos padrões já estabelecidos, assim como o *Constrained Application Protocol*, o *Message Queue Telemetry Transport* [14].

Esse tipo de comunicação apresenta uma maior complexidade em relação à segurança, visto que será necessário a implementação de credenciais para acesso à rede e para os serviços da nuvem. Um exemplo de comunicação *Device-to-Cloud* pode ser visto na figura 2.4, em que um sensor de temperatura e outro de monóxido de carbono enviam informações para um provedor de serviços, que envia seus dados para um aplicativo de celular [15].

Figura 2.4: Comunicação *Device-to-Cloud*.



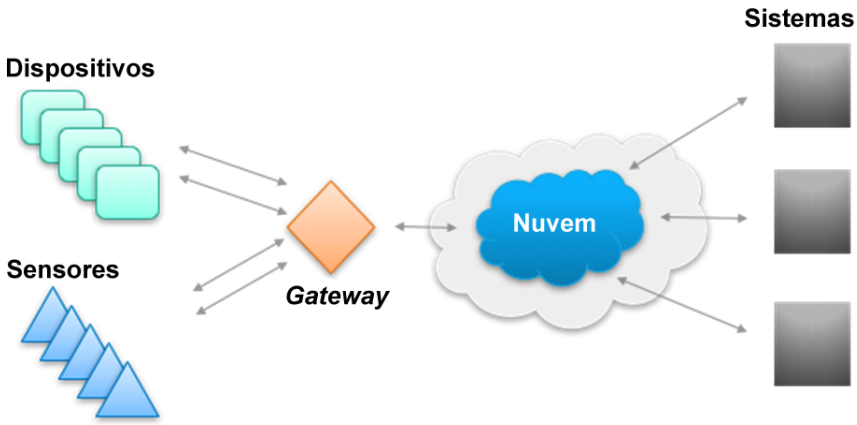
Fonte: Adaptado de http://www.thewhir.com [15]

2.2.3 Comunicação *Device-to-Gateway*

Para os casos em que sensores e atuadores ainda dependam de um serviço intermediário para que eles se conectem à internet, é utilizado o padrão de comunicação *Device-to-Gateway*. Neste caso, o dispositivo, por exemplo, se conecta à um aparelho próximo, que, por fim, se conecta à internet, fazendo a tradução de diferentes padrões de comunicação para que distintos dispositivos se comuniquem. Ele também pode providenciar algumas outras funcionalidades na camada de aplicação, tais como autenticação e autorização local dos dispositivos, aumentando a segurança da rede e de seus usuários.

O *gateway* pode ser tanto um dispositivo móvel, como o próprio celular, ou dispositivos fixos instalados dentro da rede de uma casa para tal finalidade. Um exemplo de rede que pode ser analisado o uso do *gateway* é uma plantação onde diversos sensores estão instalados em alguns pontos do solo para medir sua qualidade, como os nutrientes, o PH e a umidade. Uma vez que são vários sensores espalhados por uma vasta área e que possivelmente estão posicionados em locais sem fácil acesso à internet, sua comunicação é feita diretamente com um ou mais *gateways* e estes, com uma capacidade de comunicação mais robusta se conectam à internet. Um exemplo de arquitetura *Device-to-Gateway* pode ser analisado na figura 2.5.

Figura 2.5: Comunicação *Device-to-Gateway*.



Fonte: Adaptado de http://internetofthingsagenda.techtarget.com [16]

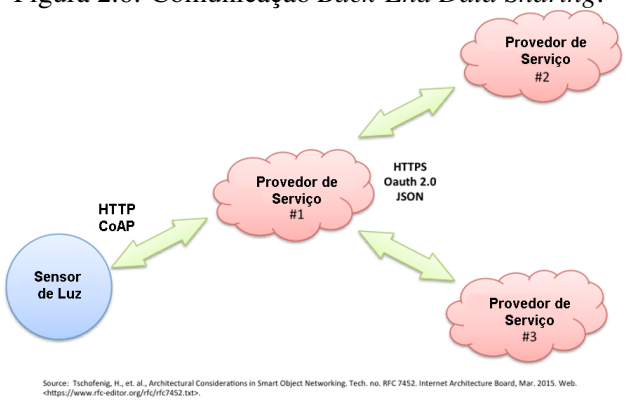
2.2.4 Padrão *Back-End Data Sharing*

Como uma das vantagens da IoT é a constante troca de diversos dados de vários tipos de fontes para que seja possível realizar a análise dessas informações e inferir conclusões a respeito do comportamento de pessoas e das massas ou sobre as previsões climáticas, por

exemplo. Foi proposto, então, a utilização do padrão *Back-End Data Sharing* no qual é possível combinar informações de diferentes bancos de dados e servidores para, utilizando o mesmo exemplo acima, entender como é o comportamento das pessoas em dia de chuva.

Toda essa comunicação é feita de forma transparente ao usuário e todos os serviços serão realizados por computadores especializados para tratamento de dados e controle de sistemas. Uma aplicação possível para esse serviço é na junção dos dados de posicionamento do carro e do serviço de previsão climática. Se for detectado que uma pessoa parou um carro em uma região de alagamento e a possibilidade de chuva neste dia for alta, o proprietário do veículo poderá ser notificado sobre o perigo. Outro cenário, voltando ao caso da fazenda, integrando a análise das condições do solo com o clima, a quantidade de água utilizada na irrigação poderá ser maior ou menor visto a previsão de chuva no dia. A figura 2.6 ilustra este padrão [17].

Figura 2.6: Comunicação *Back-End Data Sharing*.



Fonte: Adaptado de http://www.innvonix.com/blog/iot/iot-internet-of-things/ [17]

2.3 Protocolos de Comunicação

Para que os objetos se conectem à internet é necessário que seja definido modelos e padrões que atendam as reais necessidades desse mercado. Em muitos casos, como esses dispositivos são criados para fins específicos e que não exigem um *hardware* muito complexo para realizar suas funções, eles são limitados em recursos, tais como memória, processamento e fornecimento de energia, enfatizando a utilização de protocolos de comunicação mais simples e de fácil implementação, porém sem que haja um forte apelo para a segurança, devido sua simplicidade.

Neste capítulo, será feita uma breve análise dos seguintes protocolos de comunicação, que

atualmente estão em grande uso na implementação de dispositivos de IoT [18]:

- *Message Queue Telemetry Transport* (MQTT)
- *Constrained Application Protocol* (CoAP)
- *Extensible Messaging and Presence Protocol* (XMPP)
- *Advanced message Queuing Protocol* (AMQP)

2.3.1 MQTT

Message Queue Telemetry Transport (MQTT) é um protocolo leve para comunicação, que roda em cima do Protocolo de Controle de Transmissão (TCP), tendo definido pela Autoridade de Atribuição de Números da Internet (IANA) a porta 1883 por padrão para a comunicação do servidor com o cliente. Seu funcionamento é baseado no princípio de publicação de mensagens em um tópico e na adesão de leitura de tópicos [19], ou seja, sempre que um sensor for enviar um dado ele deve especificar o tópico para o qual a informação será enviada, por outro lado, toda aplicação que quiser receber informações sobre um assunto deverá se inscrever no tópico desejado e sempre que houver alguma novidade neste tópico, todos os inscritos serão informados.

Por exemplo, em uma rede doméstica, o sensor de temperatura deverá realizar o envio de dados para o tópico /sensores/temperatura, portanto, o termostato precisará estar inscrito neste mesmo tópico para receber todos os dados da temperatura e ligar ou desligar o ar-condicionado conforme as condições do ambiente.

Para que haja essa comunicação é necessário a utilização de um *broker*, que age como um servidor, no qual os clientes se conectam tanto para publicar quanto para inscrever em tópicos. Não há a necessidade de criação e configuração dos tópicos. Para realizar uma publicação é requisitado qual o tópico será postado a mensagem, sendo que é possível criar níveis de hierarquia separados pelo caractere '/'. Em uma casa com sensores de temperatura em todos os cômodos, um modo de padronização que poderá ser utilizado em tópicos pode ser o seguinte:

1. casa/sensores/quarto/temperatura

Portanto, um cliente que quiser receber informações de temperatura do quarto deverá se inscrever no mesmo tópico. Alguns caracteres especiais podem ser usados para facilitar a

inscrição à tópicos. O caractere '+' implica em todos os assuntos em um mesmo nível de hierarquia, enquanto o caractere '#' define todos os assuntos da mesma categoria e suas subcategorias restantes, por exemplo:

1. casa/sensores/+/temperatura

2. casa/sensores/quarto/#

O cliente que se inscrever no primeiro tópico receberá os dados de temperatura de todos os cômodos presentes dentro de 'casa/sensores/'. Enquanto quem se inscrever no segundo tópico receberá informações de todos os tipos sensores de dentro do quarto, seja ele temperatura, umidade do ar, luminosidade ou qualquer outro dispositivo enviando informações para 'casa/sensores/quarto/'.

Em relação à Qualidade do Serviço (QoS), que define qual o grau de confiança da entrega das mensagens, o MQTT implementa três níveis:

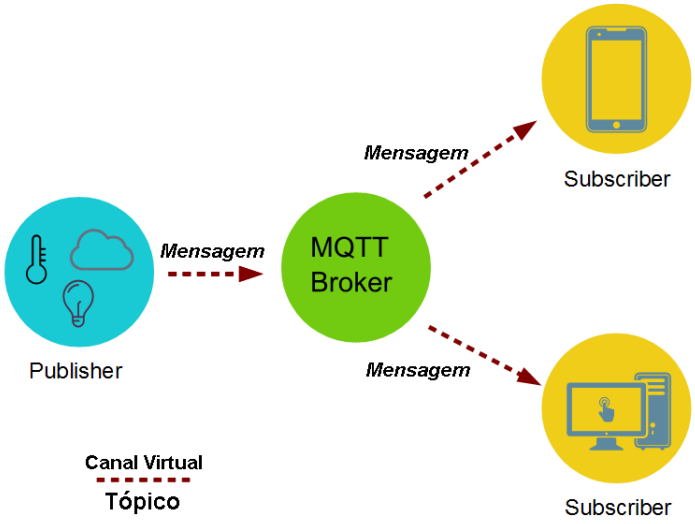
- Nível 0: A mensagem será enviada apenas uma vez sem que haja a necessidade de confirmação de recebimento.

- Nível 1: Necessário a confirmação de recebimento, sendo que a mensagem poderá ser enviada uma ou mais vezes.

- Nível 2: Será estabelecido uma confirmação de quatro vias (*four steps handshake*) para confirmar a comunicação entre as partes e a mensagem será enviada apenas uma vez.

A arquitetura básica do protocolo MQTT é definida na figura 2.7.

Figura 2.7: Arquitetura do protocolo MQTT.



Fonte: Adaptado de https://www.survivingwithandroid.com/2016/10/mqtt- protocol-tutorial.html [20]

2.3.2 CoAP

O *Constrained Application Protocol* (CoAP) foi desenvolvido pensando sua utilização para que dispositivos com recursos limitados pudessem se comunicar com a internet. Seu funcionamento se dá por meio do Protocolo de Datagrama de Usuário (UDP), tendo a porta 5683 definida para sua comunicação. Este protocolo se baseia em um serviço de pedido e resposta em uma arquitetura cliente e servidor, ou seja, é o cliente que solicita o dado desejado ao servidor, semelhante ao modelo de Protocolo de Transferência de Hipertexto (HTTP), porém de uma forma mais simples leve e com suporte a *multicast*, características necessárias para a IoT e a comunicação M2M.

O CoAP possui definido quatro tipos de mensagem: *confirmable*, *non-confirmable*, *acknowledgement* e *reset*. Quando há necessidade de confirmação do recebimento de um pacote, é enviado a mensagem do tipo confirmable, sendo que para todo pacote desse tipo é enviado uma mensagem de *acknowledgement* ou *reset* deve ser retornado, considerando que não ocorram perdas no meio do caminho.

Por outro lado, as mensagens *non-confirmable* não requisitam resposta. As mensagens de *acknowledgement* confirmam que uma mensagem *confirmable* foi devidamente recebida, mas não indicam o sucesso da requisição presente no pacote. Já a mensagem *reset* confirma que uma mensagem específica foi recebida, seja ela *confirmable* ou *non-confirmable*, indicando também a falta de conteúdo para sua devida interpretação [21].

Este padrão possui suporte aos métodos GET, PUT, POST e DELETE que realizam as funções descritas na tabela 2.1 [22] .

Tabela 2.1: Métodos CoAP

Método	**Descrição**
GET	Pega a informação correspondente ao recurso requisitado
POST	Realiza o processamento da mensagem enviada
PUT	Cria ou atualiza a informação de um determinado recurso com a nova mensagem
DELETE	Apaga o recurso selecionado

2.3.3 XMPP

Extensible Messaging and Presence Protocol (XMPP) é um padrão aberto de comunicação baseado em *Extensible Markup Language* (XML), que com as especificações estabelecidas pelo XMPP *Extensions Protocols* (XEP) suas funcionalidades podem ser customizadas e adaptadas conforme as necessidades do sistema em que realizará as trocas de mensagens, sendo possível se adequar as condições de recursos limitados recorrentes em dispositivos de IoT. Seu funcionamento ocorre por meio do protocolo TCP, possuindo suporte tanto para sistemas de pedido e resposta, utilizado pelo CoAP, quanto ao sistema publicar e inscrever, realizado pelo MQTT. O protocolo implementa notificações de presença que avisa quando o usuário está disponível ou não.

Sua arquitetura também é baseada em cliente e servidor, no qual cada cliente só conversa com seu servidor, e, caso haja a necessidade, os servidores comunicam-se entre si para a troca de mensagens entre clientes que estão em diferentes domínios, não havendo uma aplicação centralizada, conforme pode ser observado na figura 2.8. É também responsabilidade do servidor realizar todo o gerenciamento dos usuários conectados a ele. [23]

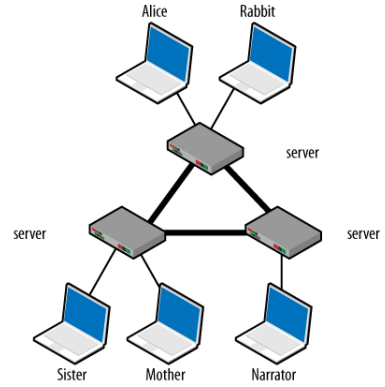
Para a identificação dos clientes, o XMPP possui um padrão conhecido como *Jabber-ID* (JID) composto por um nome de usuário, seguido por um caractere '@' e o domínio do servidor e, por fim, o recurso utilizado separado por um caractere '/' do identificador do domínio. Neste protocolo é possível a conexão de um mesmo usuário em diversos clientes de forma simultânea. Um exemplo de JID pode ser observado abaixo:

usuario1@domínio1/appCelular

Quando o cliente inicia uma conexão, um servidor XMPP é estabelecido utilizando co-

municação TCP de longa duração e, em seguida, a comunicação entre ambas as partes são realizadas com três tipos de mensagens do protocolo de Linguagem de Marcação Extensível (XML), <presence/>, <message/> e <iq/> [23], conhecidas como *stanza*. A primeira se baseia no método publicar e inscrever, e é possível receber informações do status do usuário, por exemplo, online, ausente ou offline. A *stanza* <message/> é utilizado para pegar os dados de um dispositivo e enviar para o outro, contendo os identificadores de remetente e destinatário. Por fim o *Info/Query* ou *stanza* <iq/> funciona na forma de interação pedido/resposta, possuindo atributos que funcionam de forma similar aos métodos GET, POST e PUT visto no protocolo CoAP.

Figura 2.8: Arquitetura do protocolo XMPP.



Fonte: https://www.safaribooksonline.com/ [24]

2.3.4 *Advanced Message Queuing Protocol*

O *Advanced Message Queuing Protocol* (AMQP), assim como os demais anteriormente apresentados, é um padrão para troca de mensagens entre dispositivos de maneira simples e confiável. Suas principais características são: eficiência, pois realiza um controle de fluxo de forma a reduzir a ociosidade da rede e melhorar a conexão dos seus nós; confiabilidade, realiza o envio de mensagens com várias possibilidades de garantia de entrega; e flexibilidade, o que garante o suporte a diferentes topologias, sendo usados para comunicação entre dois clientes, dois servidores ou cliente e servidor.

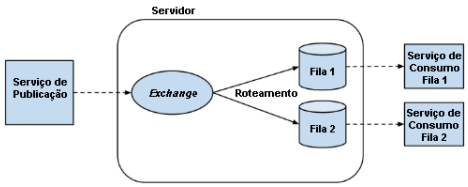
Este protocolo utiliza os conceitos de filas para gerenciar as mensagens, sendo que quando uma aplicação realiza a publicação de uma mensagem ao servidor, este irá armazená-la em uma fila enquanto elas não são consumidas por alguma outra aplicação. É possível que haja diversas filas que se relacionam com diferentes assuntos, sendo este o conceito de vínculo, ou seja, é um critério utilizado para realizar o direcionamento das mensagens dentro do servidor.

Para identificar as mensagens exitem as chaves de roteamento, enquanto as chaves de vínculos servem para especificar a fila em que a mensagem deverá ser direcionada.

Quem realiza o roteamento das mensagens para as filas dentro do servidor é o *exchange* utilizando as chaves de vínculos que são associadas aos vínculos. Existem alguns tipos de *exchange* que utilizam diferentes critérios para realizar o roteamento das mensagens. O tipo Direto envia as mensagens para a fila com a exata correspondência da chave de roteamento, enquanto o tipo *Fanout* faz a distribuição das mensagens para todas as filas que estão ligadas a ele, independente da chave de roteamento. Os *exchanges* do tipo tópico, como o próprio nome já define, cria estruturas para agrupar conteúdos próximos e criar vínculos em categorias e subcategorias, por exemplo, 'sensores.quarto.temperatura' ou 'dispositivos.sala.luz1'. Como já discutido no protocolo MQTT há a possibilidade de utilizar caracteres especiais, tais como, '#' para definir uma ou mais correspondências dentro de um tópico [25].

A arquitetura do protocolo AMQP pode ser observado na figura 2.9 abaixo:

Figura 2.9: Arquitetura do protocolo AMQP.



Fonte: Adaptado de https://blogs.vmware.com/ [Editado] [26]

2.4 A Segurança na IoT

Com o aquecimento do mercado em relação aos produtos inteligentes, ocasionou que as fabricantes, querendo rapidamente atender a demanda do público pelos serviços de IoT, desenvolvessem seus produtos sem uma regulamentação de qualidade e preocupação com a segurança dos seus dispositivos.

Com os dispositivos conectados com a internet, realizando tarefas críticas do cotidiano das pessoas, como controle de acesso à residência, câmeras de vigilância ou controle funcional do carro, e atuando na coleta contínua de informações pessoais dos usuários, abriu-se diversas oportunidades para que os ciberataques aumentassem aceleradamente nos últimos anos, transformando a segurança em um fator fundamental para preservar os dados privados e proteger a integridade dos clientes.

Segundo [27], a segurança é um dos mais importantes tópicos a serem considerados no

desenvolvimento de produtos ligados à IoT, principalmente devido seus recursos limitados de memória, processamento, energia e seu fácil acesso físico, estando exposto ao ambiente e podendo ser facilmente corrompido por *hackers* com más intenções.

A comunicação é outro fator que necessita ser mantido íntegro e seguro, sendo sua utilização essencial ao conceito da IoT. Como a troca de informações entre os dispositivos é realizado em meios públicos e não seguros, a rede está sujeita a ser corrompida por ataques de personificação, à protocolos, de negação de serviço, de força bruta para acesso em nós específicos.

Segundo [28] existem algumas propriedades que devem ser verificadas para garantir a segurança da comunicação na rede.

Confidencialidade: A interpretação da mensagem deve ser privilégio apenas de quem envia e quem recebe o pacote, não podendo haver possibilidade de terceiros interceptarem as informações e conseguirem ler o conteúdo. Para garantir esta propriedade existe o conceito de criptografia, no qual é criado um código a partir da mensagem inicial e apenas quem tiver uma chave específica poderá decifrar o código e ler seu conteúdo original.

Integridade da Mensagem: É fundamental que a mensagem seja entregue ao seu destinatário sem sofrer alterações no meio do caminho, seja de forma intencional ou não.

Autenticação de Ponto Final: Cada ponto da comunicação, remetente e destinatário, precisam confirmar que são quem realmente dizem ser por meio da autenticação de ambas as partes. No meio da IoT, como muita comunicação é realizada apenas entre duas máquinas, M2M, sem que haja a intervenção e interação de humanos, é fundamental ter um processo de autenticação eficiente para que cada equipamento consiga identificar padrões de que a informação está vindo por meio de nós confiáveis.

Para [2], no mundo da internet das coisas mais algumas propriedades deverão ser respeitadas:

Anonimato: Para garantir a privacidade é importante que não se saiba a fonte que está enviando dados, por exemplo, não é seguro que seja divulgado que informações sobre as condições de uma rodovia seja enviada por determinado carro ou usuário.

Recente: Como o funcionamento da IoT é muito dinâmico, lidando com informações em tempo real, é necessário que os dados sejam constantemente atualizados e que dados antigos não sejam mais utilizados. Para o caso de um gerenciador de vagas de estacionamento é fundamental que a informação de uma vaga livre seja realmente a realidade do momento, uma vez que não é interessante ao usuário ir até o local da vaga e encontrá-la ocupada, devido

ao atraso de alguns minutos da informação.

Controle de Acesso: Garante diferentes níveis de acesso para usuários, ou seja, nem todo mundo na rede precisa acessar todos os conteúdos. Em muitos casos algumas aplicações não precisam ter acesso à certas funcionalidades ou informações, mesmo estando autorizada a fazer parte da rede, portanto o controle de acesso assegura que apenas quem realmente precisa da informação receba o conteúdo.

Operabilidade: Devido a diversidade de soluções possíveis na IoT, é importante que os requisitos de funcionalidade não limite as possibilidades das aplicações.

Escalabilidade: A tendência é que a quantidade de 'coisas' conectadas sejam cada vez maiores, portanto é requerido que as soluções de segurança consigam abranger desde redes com poucos dispositivos tendo acesso até ambientes com milhares de sensores acessando e se comunicando.

Eficiente no uso da memória: Como a memória é limitada nos dispositivos, os algoritmos de segurança devem ser eficientes em economizar tanto a Memória de Acesso Randômico (RAM) no momento do processamento quanto devem poupar no tamanho dos arquivos de criptografias que serão armazenados.

Sobrecarga mínima de computação e comunicação: Devido aos recursos limitados já mencionados, os algoritmos de segurança devem consumir o mínimo possível do processamento da Unidade Central de Processamento (CPU) e também realizar poucas trocas de mensagens.

2.4.1 Criptografia

Segundo [29], a necessidade de impedir que uma mensagem fosse lida por uma pessoa indesejada ou um inimigo em uma guerra foi a motivação para que códigos fossem criados de forma que apenas as partes interessadas pudessem criar e interpretar um conteúdo.

Utilizando este mesmo princípio é que a criptografia foi introduzida atualmente ao mundo da internet e, agora mais do que nunca, na realidade da IoT. Como a geração de informação e sua troca entre diferentes sistemas é a base para o funcionamento do mundo onde os objetos possuem certa autonomia, a necessidade de preservar os dados e de proteger a comunicação é fundamental.

Contudo, a aplicação da segurança e criptografia pode exigir uma ótima performance dos dispositivos envolvidos, visto que as técnicas de codificar e decodificar um conteúdo são derivadas de equações matemáticas que podem ser extremamente complexas a ponto de

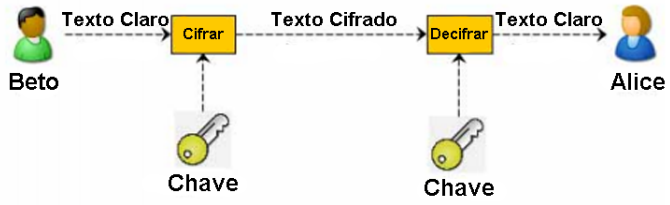
sistemas simples e limitados de processamento, como é o caso na atual realidade de sensores e aparelhos disponíveis para IoT, não sejam capazes de realizar as operações.

Portanto surgiu a necessidade de desenvolver formas leves e simplificadas, porém seguras, de realizar a criptografia.

Criptografia de chaves simétricas

As criptografias de chaves simétricas utilizam de uma mesma senha para realizar a codificação e a decodificação da mensagem, sendo, portanto, de conhecimento do emissor e do receptor qual o segredo utilizado para privatizar a informação. Porém, por outro lado, esta abordagem é arriscada, pois como é utilizado sempre uma única chave, uma vez descoberto este segredo, o invasor poderá facilmente decodificar a mensagem. Na figura 2.10 pode ser observado o funcionamento deste tipo de criptografia [30].

Figura 2.10: Modelo de criptografia de chave simétrica.

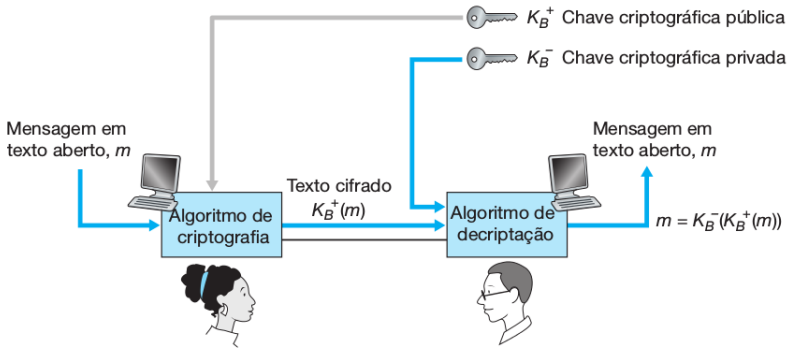


Fonte: Introdução a Infraestrutura de Chaves Públicas e Aplicações [30]

Criptografia de chave pública

Uma outra abordagem para criptografia é utilizar pares de chaves pública e privada, conhecida também como criptografia assimétrica. Como seu nome já diz, existe uma chave que é de conhecimento geral dos usuários. Essa chave é específica de um determinado destinatário e ela deve ser utilizada pelo remetente ao codificar uma mensagem à este receptor. O destinatário poderá descriptografar a mensagem apenas se possuir uma outra chave, de esfera privada que será apenas de seu conhecimento. Seu funcionamento pode ser observado na figura 2.11.

Figura 2.11: Modelo de criptografia de chave pública.



Fonte: Computer networking: a top-down approach, volume 5 [28]

Porém, como qualquer usuário possui a chave de criptografia, um ataque de texto aberto pode ser aplicado para tentar descobrir a chave privada. O invasor tendo acesso ao método de criptografia, à chave pública, à mensagem criptografada e à mensagem original ou à parte dela, ele consegue realizar a decodificação dos dados e encontrar a chave privada do destinatário. Outra desvantagem é a tentativa de algum intruso com a chave pública tentar se passar por um usuário confiável e enviar mensagens falsas ao destinatário. Neste caso, portanto, há a necessidade de autenticação dos usuários válidos para que a identidade do remetente seja verdadeira e ninguém tente se passar por ele.

2.4.2 A criptografia com a IoT

Um fator que tem segurado o avanço da Internet das Coisas no mercado atual é a segurança destes dispositivos, considerando a numerosidade e a heterogeneidade dos sistemas. São bilhões de aparelhos desenvolvidos por diferentes empresas, que se comunicam utilizando diversos protocolos, porém toda essa comunicação deverá ser segura para proteger os dados trafegados.

Entretanto, grande parte das soluções em IoT possuem tamanhos reduzido e, por consequência, recursos de processamento limitados. Desta forma, será preciso estudar com cuidado qual a melhor abordagem para cada caso, considerando se é viável ou não adaptar algoritmos de criptografia já existentes para os *hardwares* com pouca capacidade de processamento. Outra vertente pesquisada é a da introdução de novas implementações de algoritmos criptográficos que sejam naturalmente rápidos e compactos [31].

2.4.3 Principais Ataques

Distributed Denial Of Service Attack

Com o intuito de retirar da internet determinados serviços ou sistemas, o ataque conhecido com *Distributed Denial Of Service Attack* (DDoS) utiliza milhões de dispositivos comprometidos espalhados pelo mundo para realizar inúmeros acessos em um curto período de tempo em um servidor, estourando sua capacidade e fazendo com que ele pare de funcionar momentaneamente.

Softwares maliciosos, como o *Mirai* [32] são colocados na rede para analisá-la em busca de dispositivos com falhas em recursos de segurança ou sistemas fracos de autenticação de usuários, que utilizam senhas comuns ou que vêm em padrões de fábrica, como por exemplo o par usuário e senha sendo, respectivamente, *root* e *root*, como em diversos sistemas *linux*.

Uma vez invadidos, esses dispositivos passam a responder comandos de um usuário indesejado e permite que ele crie uma rede de dispositivos ou *botnet* que é capaz de gerar *gibabytes* de dados, quando em milhões de dispositivos, fornecendo grandes fluxos de informações direcionados para servidores que não possuem tamanha capacidade.

Alguns dos tipos mais comuns de ataques DDoS são [33]:

UDP ataque: Utilizando-se do protocolo UDP, o invasor envia uma enorme quantidade de acessos em diversas portas do usuário, que tentará encontrar por aplicações que estejam utilizando as portas e, caso não haja, ele gera um pacote de negação de destino. Todo esse processo pode deixar o servidor saturado e inacessível.

ICMP ping ataque: Utilizando do recurso de envio de uma mensagem do tipo *ICMP Echo Request* a um destinatário e esperando um retorno de outra mensagem tipo *ICMP Echo Reply*, este ataque faz o envio constante do *request* ao servidor, sem esperar uma resposta. Com uma grande quantidade de requisições o destinatário não consegue gerenciar todos os pedidos e resposta e acaba gerando uma lentidão em sua execução.

SYN ataque: utilizando da sequência *three-way handshake* utilizada pelo protocolo TCP para estabelecer uma conexão, em que um serviço que irá se conectar com um servidor envia inicialmente um sinal SYN para o servidor, que responde com uma mensagem SYN-ACK e, por fim, o solicitante retorna um sinal ACK para completar a conexão. Dessa forma o invasor, envia inúmeras solicitações SYN ao servidor, sem retornar a confirmação ACK, o que faz com que o servidor fique aguardando a resposta, atingindo um momento em que o servidor se sature e possua a negação do seu serviço.

Conforme pode ser observado nos exemplos acima, para realizar um ataque de negação é necessário um número elevado de conexões. Com a vinda de IoT, em que bilhões de dispositivos estarão conectados, se não houver uma segurança relevante dos aparelhos, prospectar grandes quantidades de usuários para um ataque não será um tarefa difícil.

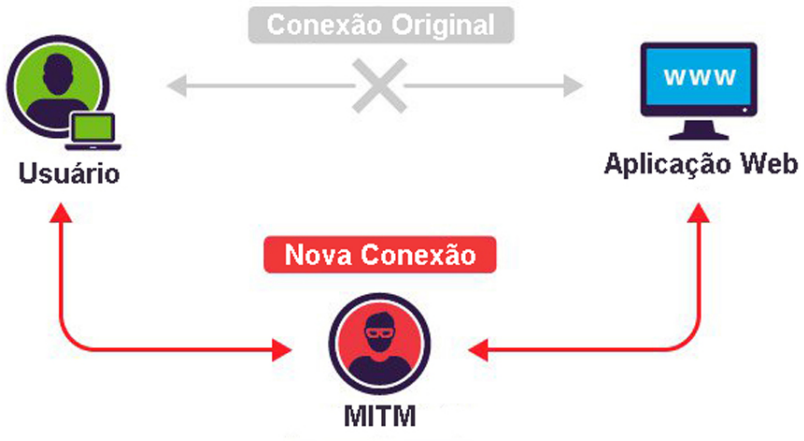
Um dos maiores e mais comentados ataques DDOS que ocorreram foi realizado contra os servidores da empresa Dyn, que realiza o controle de muitos domínios famosos da internet, tais como Netflix, Twitter, CNN e outros. Um de seus dados relevantes está relacionado à fonte do ataque sendo identificada pelo *software Mirai*, que utiliza justamente da rede fornecida por dispositivos de IoT para realizar o ataque[34]. Segundo a própria informação da empresa vítima [35], estima-se que foram envolvidos mais de cem mil dispositivos infectados em um ataque que gerou um tráfego de 1,2 Terabytes de dados por segundo.

Man-In-The-Middle

Com o intuito de roubar informações da vítima, o ataque *Man-In-The-Middle* (MITM), como o próprio nome já diz, posiciona o invasor entre dois clientes confiáveis e que se comunicam para interceptar os dados trocados entre os serviços e conseguir de alguma forma captar informações relevantes. Por exemplo, quando um cliente de banco deseja acessar sua conta por meio da internet, uma pessoa com intenções maliciosas pode explorar vulnerabilidades presentes na infraestrutura da rede, como em roteadores mal configurados para receber todos os dados enviados ao roteador e depois enviá-los de volta à rede como se nada houvesse acontecido, e portanto receber os credenciais de acesso do cliente à sua conta no banco [36].

Não apenas interceptar uma comunicação, mas com o ataque do tipo MITM, o *software* malicioso pode alterar alguns pedaços sou toda a mensagem enviada sem que ambos, remetente ou destinatário, identifique a alteração. Considerando que diferentes canais de comunicação, tais como o Sistema Global para Comunicação Móveis (GSM), o Sistema Universal de Telecomunicação Móvel (UMTS), o *Long-Term Evolution* (LTE), o *Bluetooth*, o *Near Field Communication* (NFC) e o Wi-Fi podem sofrer com o ataque. [37]. Este tipo de atividade 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. Uma representação de um ataque MITM pode ser visto na figura 2.12.

Figura 2.12: Representação de um ataque MITM.



Fonte: Adaptado de https://www.kaspersky.com.br/blog/what-is-a-man-in-the-middle-attack/462/ [38]

Em [38], caso seja utilizado o protocolo de segurança SSL/TLS (*Security Socket Layer/Transport Layer Security*), ainda é possível realizar um ataque MITM, porém duas fases são consideradas em um ataque deste tipo: Interceptação e Descriptografia. A primeira, que realiza o roubo dos dados, possui distintas abordagens, como por exemplo, a implementação de uma fraude no endereço IP, em que o invasor se disfarça sendo uma aplicação que por meio da modificação dos cabeçalhos dos endereços IP, faz com que quem tenta acessar um site específico seja direcionado para o site malicioso.

Outra abordagem, realiza uma associação do endereço MAC (Controle de Acesso à Mídia) de um dispositivo invasor com o endereço IP de um usuário válido de uma rede, fazendo com que os pacotes enviados para a vítima, que possui o endereço IP roubado, seja roteado para o invasor. Por fim, alterar servidores do Sistema para Nomes de Domínios (DNS) também é outra forma de implementar uma interceptação da conexão, uma vez que alterado este endereço, uma vítima, ao realizar o acesso à um site, tem sua conexão redirecionada para o endereço invasor.

Após interceptar uma conexão, os dados recebidos devem ser descriptografados. Existem alguns métodos que auxiliam nesta tarefa. Realizando a Falsificação do Protocolo HTTPS em que um certificado ilegítimo é enviado para o navegador da vítima quando uma solicitação inicial à um site seguro é realizado, sendo assim o invasor consegue acesso a qualquer dado enviado pela vítima, antes que ele chegue ao seu destino correto.

Outra forma de superar a criptografia é por meio do Sequestro da Conexão SSL, que é

realizado quando o invasor envia chaves falsas de autenticação tanto para o usuário vítima quanto para a aplicação no momento em que é estabelecido a conexão por TCP. Mesmo que pareça uma conexão segura para ambas as partes envolvidas, quem está controlando a sessão é um terceiro usuário invasor, que realiza o ataque MITM.

O invasor ainda tem a possibilidade de Remover a Segurança SSL enviando uma versão não criptografada de uma aplicação para a vítima no momento da autenticação TLS, enquanto ele mantém ativa conexão segura. Neste caso, toda a sessão da vítima com o serviço fica aberta e disponível para que o invasor veja as informações.

Um dos casos que obteve grande repercussão e que gerou grande discussão em relação à segurança dos dispositivos conectados foi o sequestro virtual de um carro que possui todos os seus sistemas integrado e conectado à internet. Os invasores conseguiram assumir remotamente o controle de diversas funções do veículo, que variam desde o sistema de entretenimento, como o rádio e o painel de mídia, até recursos mais críticos, como os freios e a transmissão [39]. Casos como este trazem uma insegurança com o modo em que as empresas estão agindo para proteger a privacidade e a integridade dos usuários, além de evidenciar a necessidade de adotarem políticas de segurança mais rígidas no desenvolvimento de um produto que se conecte à internet.

Capítulo 3

Material

3.1 MQTT

O protocolo de comunicação utilizado neste trabalho foi *Message Queue Telemetry Transport* (MQTT) é implementado sobre o protocolo TCP e se baseia no conceito de inscrição e publicação para troca de mensagens entre duas máquinas de uma forma simples e leve. Seus criadores, Andy Stanford- Clark e Arlen Nipper, desenvolveram-no com o intuito de realizar a comunicação de oleodutos por meio de conexão via satélite [40].

Mas com o surgimento do conceito de IoT, suas características o fizeram atrativo para a comunicação máquina-máquina entre dispositivos com pouco recurso de banda e processamento disponíveis. Sendo que em 2013, na versão 3.1.1, a Organização para o Avanço dos Padrões de Informação Estruturada (OASIS) o declarou oficialmente como um padrão leve e de código aberto para a comunicação de dispositivos.

3.1.1 *Broker*, *Publishers* e *Subscribers*

Seu funcionamento se baseia em inscrições em tópicos e publicações de mensagens. São considerados apenas dois tipos de clientes, o *publisher* e o *subscriber*, e o servidor, denominado *broker*. Os *Subscribers*, que ao se conectarem com o *broker*, enviam suas inscrições para os tópicos em que estão interessados em receber informações. Os *Publishers* também se conectam ao *broker* e, quando for enviar uma mensagem, eles enviam qual o tópico em que essa mensagem deverá ser publicada.

O *broker*, além de manusear toda a conexão dos clientes, ao receber as mensagens enviadas pelos *Publishers*, ele as direciona para todos os *Subscribers* que estão relacionados com o tópico. Portanto, toda a ligação entre os *Publishers* e os *Subscribers* se baseiam nos tópicos

que ambos se referem. No geral, o histórico de mensagens não é armazenado pelo *broker*, sendo que toda vez que ela é enviada aos *Subscribers*, seu conteúdo é apagado. Se algum cliente não está conectado no momento em que a mensagem foi enviada, ele simplesmente não receberá estes dados, começando a receber novas mensagens a partir do momento em que se conecta.

Os tópicos são definidos em níveis, sendo eles separados pelo caractere ´/´, semelhante ao observado na árvore de diretórios de sistemas operacionais ou em *links* URL (*Uniform Resource Locator*), não havendo limite para a quantidade de subníveis. Porém, pelo menos um caractere, utilizando o padrão UTF-8 (Formato de Transformação Unicode de 8-bit), é necessário para sua definição e no máximo 65535 *bytes* são permitidos. Alguns caracteres chaves são permitidos para facilitar a inscrição a topicos pelo *Subscriber*. Por exemplo, o caractere ´#´ define todos os itens do nível atual e subníveis e o caractere ´+´ indica qualquer item dentro de um determinado nível.

3.1.2 QoS

O MQTT implementa três tipos de Qualidade de Serviço (QoS, *Quality of Service*, em inglês), que se refere à possibilidade de sucesso do envio das mensagens. O QoS 0 estabelece que a mensagem deve ser enviada apenas uma vez, não sendo necessário um *acknowledged* para confirmar o recebimento e os dados não são armazenados pelo *Publisher*.

O QoS 1 garante que a mensagem será entregue pelo menos uma vez. Ao enviar a primeira mensagem, o emissor aguarda uma resposta de que a mensagem foi entregue. Se essa resposta não foi recebida em um determinado espaço de tempo, ela é reenviada com uma sinalização de duplicada e mais de uma mensagem pode ser recebida pelo destinatário. Esse processo se repetirá até que o emissor receba o *acknowledged* de recebimento. [41]

Por fim, o método QoS 2 determina que a publicação seja entregue apenas uma vez, sendo esta a forma mais segura e demorada de comunicação, com 4 mensagens trocadas entre emissor e destinatário. O remetente envia uma mensagem e aguarda uma confirmação. O destinatário, ao receber a mensagem, realiza seu processamento, guarda todas suas referências e envia uma mensagem PUBREC ao remetente. Ao receber o *acknowledged*, o emissor sabe que a mensagem foi recebida, podendo descartá-la e enviando uma outra mensagem PUBREL ao destinatário, que responde com PUBCOMP e completa a comunicação.

3.1.3 Sessões

Quando um cliente se conecta ao *broker*, ele realiza toda sua inscrição nos tópicos. Ao reconectar, todas as inscrições devem ser refeitas. Isso gera uma quantidade elevada de dados para serem processados se a reconexão é realizada constantemente e, seria inviável para dispositivos com pouca capacidade lidar com essa quantidade de informações constantemente.

Portanto, o MQTT possibilita que toda a sessão de um usuário seja salva quando ele se desconectar. Ou seja, todas as informações da sessão, tópicos inscritos e mensagens com QoS 1 e 2, das quais o cliente não confirmou recebimento e não estava conectado para recebê-las são armazenadas pelo *broker* até que o cliente volte a se conectar.

3.1.4 Formato da mensagem

Os protocolos de mensagens utilizados pelo MQTT possuem um cabeçalho fixo e uma variável, que depende do tipo da mensagem. O cabeçalho fixo, com dois *bytes* segue o formato da figura 3.1.

Figura 3.1: Cabeçalho fixo MQTT.

bit	7	6	5	4	3	2	1	0
byte 1	Tipo da Mensagem				Duplicado	QoS		Retain
byte 2	Tamanho Restante							

Fonte: Autoria Própria

O *byte* 1 define o tipo da mensagem nos bits de 4 a 7. São disponíveis 16 tipos de mensagens, enumeradas de 0 à 16, conforme pode ser visto na tabela 3.1. No bit 3 há um sinalizador de mensagem duplicada, nos bits 2 e 1 é informado a qualidade do serviço e no bit 0, se ele for verdadeiro, o servidor mantém a mensagem mesmo após ter sido enviada aos *Subscribers*. O *byte* 2 indica o tamanho restante da mensagem, sendo que mensagens de até 127 *bytes* são indicados em apenas 1 *byte*, valores maiores são representados com 7 bits indicando valores de até 127 *bytes* e o último bit indicando a existência de mais 1 campo para o tamanho da mensagem. No máximo são 4 *bytes* permitidos para informar o Tamanho Restante, o que gera uma mensagem de até 127^4 *bytes* ou 256 MB [42].

O cabeçalho variável se localiza entre o cabeçalho e o corpo da mensagem e contém alguns componentes dependendo do comando enviado. Os campos com destaque neste trabalho seguem descritos abaixo.

Tabela 3.1: Tipos de Mensagem MQTT

Tipo	**Índice**	**Descrição**
reservado	0	Campo reservado
CONNECT	1	Requisição de conexão do cliente com o servidor
CONNACK	2	Reconhecimento de conexão
PUBLISH	3	Publicação de mensagem
PUBACK	4	Reconhecimento de mensagem publicada
PUBREC	5	Publicação recebida
PUBREL	6	Publicação enviada
PUBCOMP	7	Publicação completada
SUBSCRIBE	8	Requisição de inscrição pelo cliente
SUBACK	9	Reconhecimento de inscrição
UNSUBSCRIBE	10	Cancelamento de inscrição
UNSUBACK	11	Reconhecimento de cancelamento de inscrição
PINGREQ	12	Requisição de PING
PINGRESP	13	Resposta à PING
DISCONNECT	14	Desconexão do cliente
reservado	15	reservado

O *byte* que possui as *flags* de conexão utiliza os bits 7 e 6 para indicar a presença de usuário e senha, respectivamente. O bit 1, se não setado, armazena a sessão do *Subscriber* quando ele se desconectar. O cabeçalho com tempo de máximo de conexão mantida é especificado por 2 *bytes* que representa, em segundos, o limite do intervalo ocioso da comunicação, ou seja, o período do qual não há troca de mensagens, o que permite ao servidor detectar que a conexão com o cliente foi interrompida. Considerando os 16 bits que formam o tempo em segundo, é possível manter uma conexão de aproximadamente 18 horas sem troca de mensagens.

A mensagem do tipo CONNACK possui entre seus cabeçalhos variáveis o código de retorno de conexão que indica o sucesso ou a falha na tentativa de estabelecer uma comunicação. A tabela 3.2 apresenta as possíveis possibilidades. Caso a conexão seja rejeitada, alguns códigos indicam o motivo da falha [42].

Tabela 3.2: Código de retorno da conexão MQTT

Índice	**Valor Hexadecimal**	**Descrição**
0	0x00	Conexão aceita
1	0x01	Conexão rejeitada por versão do protocolo
2	0x02	Conexão rejeitada devido à identificação errada
3	0x03	Conexão rejeitada pela indisponibilidade do servidor
4	0x04	Conexão rejeitada por usuário ou senha inválidos
5	0x05	Conexão não autorizada
6-255		Reservado para uso futuro

3.1.5 Segurança

A preocupação com a segurança dos dados e com intrusos na comunicação fez com que o MQTT possua algumas ferramentas para autenticação de usuários e criptografia da comunicação. Porém, como já discutido anteriormente, devido à casos de recursos limitados tanto do cliente quanto do *broker*, devem ser analisadas as ocasiões e quais ferramentas devem ser implementadas ao serviço.

Autenticação de usuário

O processo de autenticação indica que o usuário é realmente quem ele indica ser. No MQTT é possível que o cliente utilize informações de usuário e senha para realizar a conexão com o *broker*. Essa informação é enviada junto ao pacote CONNECT, sendo eles opcionais e, podendo ser enviado apenas o usuário sem ser acompanhado de uma senha, porém o inverso não é permitido. Quando usados, o usuário e a senha enviados na autenticação são transportados em texto, permitindo que qualquer sequestrador tenha acesso à informação.

Outras informações providenciadas pelo cliente também podem ser usadas como forma de autenticação. O identificador do cliente é um valor de até 65535 caracteres que é único para cada cliente e pode ser usado junto aos dados de usuário e senha para estabelecer uma conexão. O *broker* pode impor prefixos à esse dado, o que ajuda a aumentar a segurança da conexão.

Autorização de acesso

Considerando que um cliente tenha as credenciais necessárias para se conectar ao servidor, não quer dizer que este usuário possa ter acesso a qualquer informação trafegada. Portanto, a autorização define os direitos e políticas de acesso aos recursos disponíveis. Este conceito deve estar sempre relacionado à autenticação do usuário, sendo primeiro identificado a identidade do usuário para em seguida definir e limitar as áreas que acessará dentro do serviço.

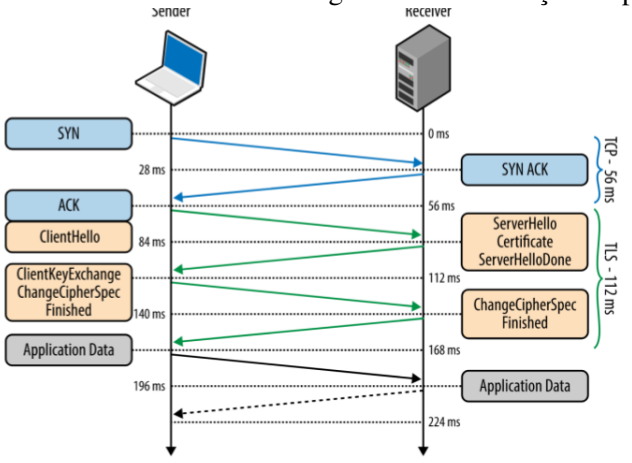
Uma forma de limitar o acesso do cliente é com a implementação de listas de controle de acesso (ACL, em inglês, *Access Control List*). Este tipo de autorização define uma lista de permissões para um recurso. Neste caso, é possível limitar quais usuários podem publicar ou se inscrever em determinados tópicos, ou seja, nem todo usuário poderá enviar ou receber informações de um tópico, apenas clientes associados a ele terão permissão para acessá-lo.

TLS

O MQTT possui suporte para comunicação utilizando o protocolo de Segurança na Camada de Transporte (TLS, em inglês, *Transport Layer Security*). Sendo derivado do *Security Socks Layer* (SSL), este protocolo foi desenvolvido para implementar segurança na comunicação da camada de transporte, garantido a integridade da conexão, mensagens criptografadas e a autenticidade da informação.

Para isso, inicialmente é realizado um mecanismo de negociação que estabelece vários parâmetros para realizar uma conexão segura, sendo eles: a versão do protocolo TLS, qual a criptografia utilizada e os certificados de autenticidade, conforme ilustrado na figura 3.2.

Figura 3.2: Estabelecimento de conexão segura com a utilização do protocolo TLS.



Fonte:https://hpbn.co/transport-layer-security-tls/ [43]

Neste caso, uma grande quantidade de dados, na casa dos kilobytes, são usados para a inicialização da comunicação. Uma forma para reduzir esse excesso de dados é utilizando formas de retomada de conexão, que permite utilizar certificados previamente negociados, mas não reduz a necessidade de um estabelecimento inicial de comunicação. Dois mecanismos podem ser utilizados nas retomadas de conexão, sendo o primeiro caso com o armazenamento dos dados da conexão associado com o ID da sessão, que é utilizado para realizar a reconexão. Já a segunda, utiliza um *ticket* de sessão, enviado criptografado do servidor ao cliente, na retomada da comunicação, este *ticket* é devolvido ao servidor que se conseguir decodificá-lo, reestabelece o contato seguro.

O certificado X509 pode adicionar uma maior segurança à rede. Ele é emitido por alguma autoridade e é utilizado pelo cliente para verificar a identidade do servidor, contendo sua chave pública e garantindo, portanto, que nenhum usuário possa ler os dados transmitidos nem alterar seu conteúdo. O cliente também pode emitir certificados que garantam sua validade e possuir chaves públicas e privadas. Os certificados dos clientes são enviados após garantir a conexão com o servidor e servem para evitar que clientes não autorizados realizem a conexão, sendo realizado uma autenticação na camada de transporte. Como esta verificação é feita antes do estabelecimento da comunicação, na camada de transporte, este procedimento pode evitar o uso desnecessário de recursos no *broker* para garantir a confiabilidade do cliente.

Porém, toda essa segurança ainda tem um custo, principalmente para dispositivos IoT, pois toda a criptografia usada exige um maior processamento do hardware e uma sobrecarga de dados transmitidos, sendo ambos grandes limitadores em sensores para Internet das Coisas. Outros grandes problemas que os certificados do cliente podem gerar é na forma de como enviar esses dados seguros ao usuário, uma vez que todos os clientes podem não estar acessíveis para receber esses dados de maneira confiável. Caso os certificados expirem ou sejam comprometidos e descartados, é importante que o *broker* saiba quando eles não são mais válidos, uma tarefa um pouco difícil quando há inúmeros clientes para serem gerenciados pelo *broker*.

Criptografia da Mensagem

Uma forma alternativa de segurança, mais leve, porém menos robusta que o TLS utiliza é a criptografia na camada de aplicação dos dados transmitidos. Esta abordagem não está definida nas especificações do MQTT, sendo de utilização exclusiva da aplicação. Não apenas a

mensagem pode ser criptografada, mas os dados, tais como, ID do usuário, senha de autenticação e tópico postado também podem ser enviados de forma codificada [44].

Dessa forma, todo um mecanismo de tradução deve estar disposto entre a comunicação do cliente com o *broker*, visto que o MQTT não suporta por padrão a criptografia, e realizar a conversão do dado puro em texto cifrado para transmitir a mensagem e a decodificação, novamente para texto, do dado recebido.

Diferentes abordagens podem ser utilizadas na comunicação dos dados codificados na camada de aplicação. Uma forma mais simples é a criptografia cliente-cliente, em que os dados não são descriptografados pelo *broker*, sendo apenas os usuários finais capazes de ler o conteúdo da mensagem. Este método permite que o *broker* não seja alterado, porém dados como o tópico e informações de autenticidade não poderão ser cifrados.

Utilizando uma criptografia cliente-servidor, três diferentes arquiteturas podem ser aplicadas para o caso da comunicação MQTT.

- Apenas os dados transmitidos pelo *publisher* serão criptografados, sendo que ao serem recebidos pelo servidor, eles são decodificados e transmitidos ao *subscriber* em texto puro.

- Apenas os dados transmitidos do *broker* para o *subscriber* são codificados, a comunicação com o *publisher* é feita em texto puro.

- Ambas as comunicações deverão ser criptografadas, sendo tudo o que é transmitido criptografado inicialmente pelo *publisher*, descriptografado pelo *broker* e criptografado novamente para ser enviado ao *subscriber*.

Como nesta abordagem o *broker* participa do processo de codificação, estas funcionalidades devem ser implementadas no código do servidor, o que permite que dados mais críticos, como informações de usuário e senha também possam ser enviados de forma segura.

Dependendo da utilização e das especificações dos dispositivos, métodos de criptografia simétrica, que é mais simples e apresenta uma chave única para codificar e decodificar a mensagem, ou assimétrica, em que um par de chaves pública e privada são utilizados para garantir mais segurança na comunicação, podem ser aplicados. Esta abordagem, mesmo ainda exigindo um pouco do processamento dos dispositivos, para criptografar e descriptografar a mensagem, pode ser vantajosa considerando que menor quantidade de *bytes* precisa ser usadas na transmissão.

Integridade da informação

Para garantir que os dados transmitidos não foram alterados por usuários indesejados, o MQTT possibilita a utilização de paridade para garantir a integridade da mensagem. As mensagens do tipo PUBLISH podem conter assinaturas digitais, Algoritmos de Autenticação de Mensagem (MAC) ou *checksum* do seu conteúdo, e o destinatário confirma se os dados estão de acordo com o esperado.

Os *checksum* agem de forma a garantir apenas que os dados não foram modificados de forma acidental. Casos de alteração intencional não serão identificados, visto que se um atacante souber qual o algoritmo de *checksum* utilizado, ele também poderá alterá-lo junto à mensagem.

Códigos de autenticação de mensagem (MAC) utilizam de funções *hash* e chaves de criptografia para realizar o cálculo do código de paridade para determinada mensagem. Seus algoritmos funcionam de forma rápida e garantem a segurança desde que a chave não seja corrompida por usuários indesejados.

As assinaturas digitais utilizam de chaves públicas e privadas para assinar uma mensagem. Apenas o remetente válido pode gerar um código de autenticidade utilizando-se da sua chave privada e sua assinatura, que será verificada pelo destinatário. Isso garante que outros usuários não sejam capazes de autenticar uma mensagem, mas qualquer um consegue verificar sua veracidade.

3.2 Mosquitto *Broker*

Como servidor foi utilizada a plataforma Mosquitto que é um serviço de código aberto para o *broker* e que implementa o protocolo MQTT nas versões 3.1 e 3.1.1, possuindo suporte para o protocolo TLS, autenticação de usuário com ou sem senha e lista de controle de acesso (ACL) [45].

Dentre suas funções normais, ele apresenta informações relevantes do seu status atual dentro de hierarquias do tópico $SYS, por exemplo:

- *$SYS/broker/bytes/received*: O total de *bytes* recebidos pelo *broker* desde que a conexão iniciou.
- *$SYS/broker/bytes/sent*: O total de *bytes* enviados pelo *broker* desde o ínicio.
- *$SYS/broker/heap/maximum*: O número máximo de memória usado pelo *broker*.

3.3 Eclipse Paho MQTT

Para o cliente foi implementado códigos em Python com a biblioteca Paho-MQTT, que é providenciada pela Eclipse e se baseia em código aberto. Ela suporta as versões 3.1 e 3.1.1 do MQTT, além de implementar conexão TLS. Suas principais funções permitem que o usuário se conecte ao *broker* utilizando um ID e autenticação de usuário e senha, e realize publicações, se inscreva nos tópicos e se desconecte, caso necessário [46] [47].

3.4 Dispositivos Cliente e Servidor

Para finalidade de simulação foram utilizados como clientes dois computadores de pequeno porte, considerados de recursos limitados. Uma Raspberry Pi 2 modelo B com processador baseado em Broadcom BCM2837 ARM7 Quad Core 32 bits, com frequência de 900MHz, 1 GB de memória RAM e Unidade de Processamento Gráfico (GPU) VideoCore IV, com 250MHz e API de renderização OpenGL ES 2.0 (24 GFLOPS) e decodificação de alto-perfil 1080p30 h.264/MPEG-4 AVC [48] [49].

Seu sistema operacional roda Raspbian Stretch Lite, lançado em 04 de abril de 2017, com versão de kernel 4.4, sendo este o sistema linux baseado em Debian e suportado oficialmente pela Raspberry Pi Foundation, sendo otimizado para a plataforma [50]. Esta versão mais leve do sistema operacional apresenta as mesmas funcionalidades da versão completa, incluindo suporte a Python e a suas bibliotecas. Sua diferença se dá apenas no suporte à parte gráfica.

O outro microcomputador é uma placa Intel Galileo Geração 2, com processador Quark X1000 de 32 bits da mesma marca, com frequência de 400 MHz e 16 KB cache L2, memória RAM de 256 MB. Esta placa roda também com sistema operacional linux Debian Wheezy, com kernel 3.8.7 lançado para esta placa [51].

O servidor roda em um computador Dell Inspiron N4110 (i14R-3240) com sistema operacional linux Ubuntu versão 14.04, possuindo como especificação processador Intel Core-i5-2430M com 2.40 GHz e 3MB de memória cache, e memória RAM de 4 GB. Este sendo considerado de recurso ilimitado para as necessidades do projeto [52].

Neste trabalho, as escolhas de cada um dos componentes ocorreram em consideração a existência de grandes comunidades presente na internet para cada item, que auxiliam tanto nas configurações dos sistemas quanto com problemas encontrados durante o desenvolvimento do projeto. Para as placas foram considerados também, além da facilidade em adquirir os dispositivos, a criação do ambiente com um cliente que possui recursos mais reduzidos,

no caso da Intel Galileo, para análise de seu comportamento em relação à solução. Para escolha da Raspberry Pi foi ponderado um cliente com um hardware mais robusto em relação ao anterior que não deverá apresentar problemas para rodar a solução e que servirá de comparação para análise de como os sistemas com e sem recursos de processamento reagem à criptografia.

Capítulo 4

Métodos

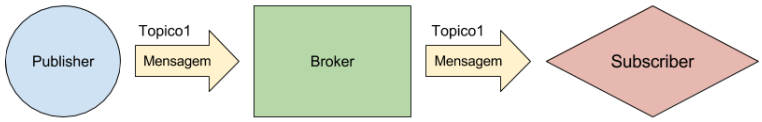
Este projeto pretende, de uma forma simples, proteger a conexão e a comunicação, por meio da criptografia de algumas informações das mensagens enviadas entre os clientes e o servidor, que se comunicam utilizando o protocolo MQTT, em um ambiente para Internet das Coisas utilizado em aplicações pessoais e na automação residencial. Considerando este cenário, foi estabelecido que os clientes não estão fisicamente acessíveis à um invasor, apenas a sua comunicação poderá ser interceptada.

O *broker* fica estabelecido na nuvem, não permitindo acesso físico por qualquer pessoa. É considerado que sua implementação na rede esteja de acordo com padrões de segurança para dificultar seu acesso de forma remota. Portando, apenas a conexão entre o cliente e o servidor, com a comunicação implementada sobre o protocolo MQTT, será analisada e estudada neste trabalho.

4.1 Conexão, Inscrição e Publicação

A arquitetura básica da transmissão implementada pelo MQTT é definida na figura 4.1. Através de uma comunicação *device-to-cloud* o *publisher* envia mensagens para o *broker* em um determinado tópico, que as repassam aos *subscribers* relacionados aos mesmos itens. Podendo, este processo, ser escalado para diversos *publishers* enviando mensagens para inúmeros tópicos e vários *subscribers* se inscrevendo para receber atualizações de um ou mais tópicos.

Figura 4.1: Comunicação entre *Publisher* e *Subscriber*.



Fonte: Autoria Própria

O fluxo do envio de pacotes para realizar a conexão e a comunicação está descrito abaixo:

1. Cliente envia uma mensagem do tipo CONNECT ao *broker* com os dados obrigatórios de ID, o bit que indica que a sessão deve ser encerrada ao desconectar e o tempo de ociosidade, que indica o período máximo sem envios de pacotes que a comunicação permitirá. Caso exigido, as informações de usuário e senha também são enviados juntos nesta mensagem, porém eles não são obrigatórios.

2. O *broker* responde ao cliente com uma mensagem de CONNACK, que determina se a conexão foi aceita ou recusada, especificando o motivo da recusa, conforme tabela 3.2.

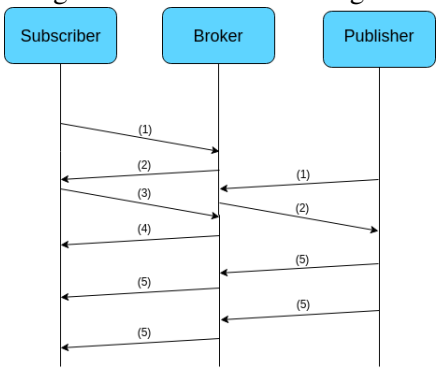
3. No caso do *Subscriber*, uma mensagem do tipo SUBSCRIBE deve ser enviada para o *broker* informando a lista de tópicos no qual ele deseja se inscrever e qual a QoS desejada (0, 1 ou 2).

4. Um pacote SUBACK, é retornado ao requerente com um código de sucesso ou falha da inscrição para cada um dos tópicos enviados. Os códigos 0, 1 e 2 indicam, respectivamente, sucesso na conexão para cada um dos níveis de QoS. O valor 128, representa uma falha na inscrição.

5. Para o *publisher*, apenas uma mensagem do tipo PUBLISH é necessário para o envio da informação, considerando o QoS de nível 0. Neste pacote, dados como o tópico, o QoS e a informação de pacotes duplicados são adicionados junto ao conteúdo publicado.

A figura 4.2 exemplifica todo o processo descrito:

Figura 4.2: Fluxo de mensagens.



Fonte: Autoria Própria

4.2 Criptografia da informação

A solução proposta neste trabalho sugere que os dados críticos, sendo eles a identidade do cliente, usuário, senha, tópico e o conteúdo da transmissão sejam criptografados entre o cliente e o servidor. Neste caso, serão utilizadas duas chaves simétricas, compartilhadas entre o *broker* e seus clientes. A primeira chave, denominada de comum, é de caráter idêntica entre todos os usuários e deve ser utilizada para codificar a identidade, o usuário e a senha. A segunda chave, chamada de única, é exclusiva para cada cliente e compartilhada apenas com o *broker*, sendo sua função realizar a criptografia do tópico e da informação enviada.

Ao introduzir um novo cliente à rede, o usuário ou o administrador do sistema irá solicitar por meio de uma plataforma de conexão com o *broker* um novo acesso, informando quem irá se conectar. Em seguida, o *broker* irá gerar os dados de chave única, o número aleatório para indicar a quantidade de vezes que essa chave será usada, o ID único, o nome do usuário, a senha e os tópicos que o cliente poderá postar e se inscrever. Os dados de ID, usuário e senha são *strings* de 16 caracteres gerados de forma aleatória, já as chaves, também criadas de forma aleatória, possuem 64 *bytes*. Junto a esses dados é adicionado a senha comum e o identificador da versão dessa senha. Todas essas informações são enviadas de forma segura para o cliente ser configurado.

A plataforma de interação do usuário com o *broker*, o modo de gerenciamento dos tópicos e a forma em que esses dados são enviados para o cliente não serão cobertos neste trabalho. Porém, nos casos em que o dispositivo está protegido pelo acesso físico dentro de uma casa, comunicações como o NFC presente em celulares ou a criação de uma rede WIFI direta com o dispositivo, poderiam ser utilizados para realizar a configuração inicial desses aparelhos e

enviar os dados a eles. Aqui é considerado que esses dados gerados são enviados via um meio seguro até os clientes, e só a partir do recebimento dessas informações é que o sistema do cliente conseguirá se conectar à rede e iniciar a comunicação.

4.3 Chaves

Para este projeto, foram utilizados dois tipos de chaves. Uma, de caráter comum, para realizar a criptografia dos dados da conexão e a outra utilizada para codificar as publicações enviadas. Ambas as chaves são geradas e administradas pelo *broker*, sendo elas uma sequência aleatória de 64 *bytes*, limitando cada *byte*, neste trabalho, entre os valores 33 e 126 da tabela ASCII, que representam caracteres conhecidos e presentes em um teclado.

4.3.1 Chave comum

A chave comum é compartilhada igualmente entre o *broker* e todos os usuários da rede. Ela é utilizada para criptografar o ID, o usuário e a senha. Seu caráter comum é devido ao fato de que como o ID é enviado criptografado, não é possível que o *broker* identifique qual usuário está realizando a conexão para buscar uma chave que seja exclusiva deste cliente. A figura 4.3 ilustra o caráter comunitário desta chave.

Figura 4.3: Chave Comum.

Fonte: Adaptado de https://canaldoensino.com.br/blog/wp-content/uploads/2012/08/redes-de-computadores.jpeg [53]

Como a chave é simétrica, portanto, de conhecimento de todos os usuários, foi implementado um sistema que realiza constantemente a troca de sua sequência de caracteres, dificultando que clientes indesejados consigam descobrir o seu valor. Sendo assim, quando uma chave é criada, também é gerado um indicador único para ela, que, para este dado, foi utilizado a hora de sua criação, sendo este o retorno da função time, presente na biblioteca de mesmo nome da linguagem Python. Este valor representado em segundos o tempo atual

desde o dia 01/01/1970. Este número real retornado é convertido para seu inteiro mais próximo e então transformado em *string* com, atualmente, 10 caracteres.

Esta identidade possui duas funções, sendo a primeira auxiliar o *broker* a definir quando uma nova chave deve ser criada e, a segunda, para que o *broker*, ao receber uma conexão, saiba qual a codificação que o cliente está utilizando, pois como este valor é alterado com o tempo, um cliente que ficou muito tempo desconectado poderá não ter sido atualizado com a nova senha comum e realizado a conexão com uma criptografia antiga. Por isso, o *broker* deverá armazenar um histórico de chaves para permitir que mesmo usuários desatualizados se conectem e, verificada sua autenticidade, eles sejam atualizados com os novos valores da chave comum.

4.3.2 Chave única

A segunda chave, com caráter único para cada cliente, é utilizada para codificar o tópico e a informação da mensagem. Como as publicações costumam ser mais frequentes que as conexões, separar as chaves utilizadas na conexão dos pacotes e na publicação das mensagens, permite uma maior flexibilidade com a troca constate desta senha, visto que quanto mais ela é utilizada, maior é a necessidade de que ela seja alterada para reduzir as chances dela ser descoberta por um usuário indesejado. A figura 4.4 demonstra a característica de exclusividade desta senha.

Figura 4.4: Chave Única.

Fonte: Adaptado de https://canaldoensino.com.br/blog/wp-content/uploads/2012/08/redes-de-computadores.jpeg [53]

Para que o *broker* saiba qual chave pertence a cada usuário, uma base de dados dentro do servidor é empregada para relacionar a senha com o ID do cliente em questão. Portanto, ao receber uma nova mensagem, uma busca é realizada dentro do *broker* utilizando a identidade do remetente para encontrar sua chave única.

Outra característica pertencente a este item é a presença de um índice de validade, que

é gerado junto a sua criação e que serve para determinar a quantidade de vezes que uma mensagem deverá ser criptografada utilizado esta sequência de caracteres. Quando este valor zerar, uma nova chave é produzida pelo *broker* e enviada ao cliente.

4.4 A criptografia

Como os algoritmos de criptografia são o grande limitante para dispositivos com pouca capacidade de processamento, foram definidas formas mais simples de alterar o conteúdo de uma mensagem de maneira a protegê-lo.

A lógica digital ou-exclusivo apresenta características que podem suprir as necessidades de criptografia para esta comunicação. Sua funcionalidade define que a operação entre 2 números binários idênticos resultem em uma saída zero, enquanto a operação de dois números diferentes o resultado é um. Isso garante que possuindo apenas os dados da saída, não seja possível encontrar os valores dos operandos. A tabela 4.1 mostra o resultado da operação de ou-exclusivo entre os valores a e b.

Tabela 4.1: Exemplo de Operação Ou-Exclusivo.

a	0	0	1	0	1	1	0	1
b	1	0	1	0	0	1	1	1
Saída	1	0	0	0	1	0	1	0

O valor da saída apenas indica que os bits de mesma ordem dos operandos são iguais ou diferentes, não sendo possível definir os bits de a nem de b. Porém, possuindo conhecimento sobre um dos operando e o resultado da operação, o outro valor é facilmente deduzido. Isso inviabilizaria toda sua utilização, pois muitos dos dispositivos enviam apenas dados simples, como 1 ou 0 para indicar, por exemplo, um estado de ligado ou desligado. Então, mesmo não sabendo a senha, um invasor pode possuir o conhecimento do valor inicial e do seu resultado com a operação de ou-exclusivo com uma chave e, realizando a operação inversa, descobrir qual é esta chave.

Uma forma de dificultar o processo de deciframento da senha, é utilizando mais de uma operação de ou-exclusivo em sequência, utilizando diferentes chaves intermediárias. Portando, um invasor com o conhecimento dos possíveis dados gerados pelo cliente e o resultado da criptografia, sem saber as chaves intermediárias usadas, não conseguiria, de maneira simples, chegar ao dado real transportado.

Tabela 4.2: Exemplo de Operação Ou-Exclusivo.

Dado	1	0	1	0	1	1	0	1
Senha1	1	1	0	1	0	1	1	1
Resultado1	0	1	1	1	0	0	0	1
Senha2	0	0	1	1	0	1	1	0
Resultado2	0	1	0	0	0	1	0	0
Senha3	1	1	1	0	1	0	0	1
Dado Criptografado	1	0	1	0	1	1	0	1

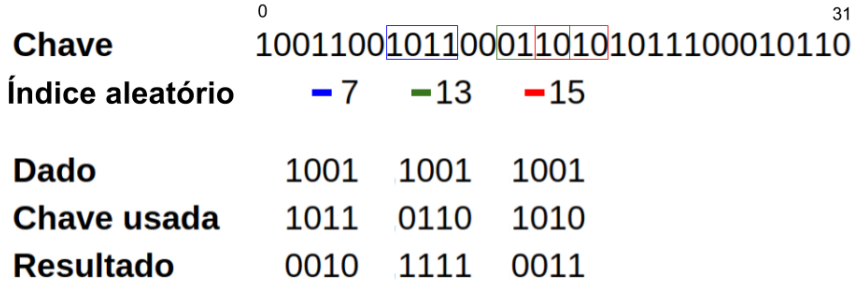
No exemplo da tabela 4.2, o invasor conhecendo os possíveis valores do dado e do resultado criptografado, não conseguiria de forma simples definir quais as senhas utilizadas. Ele deverá testar todas as possibilidades para as senhas 1, 2 e 3 para encontrar a combinação entre o dado e seu correspondente criptografado. É por este motivo que chaves de 64 *bytes* são utilizadas. Para dados pequenos de um *byte*, 16 caracteres da senha poderão ser selecionados para realizar operações de ou-exclusivo em sequência de forma vertical. Um ataque de força bruta que tente descobrir esses 16 *bytes* utilizados entre a mensagem e a cifra, deverá testar 93^{16} possibilidades de chaves. O valor 93 é definido do total de possibilidade de caracteres entre os valores 33 e 126 da tabela ASCII.

Para mensagens de 2 *bytes*, a mesma ideia pode ser utilizada, porém realizando 8 operações de ou-exclusivo com 2 caracteres cada. Essa forma também utiliza 16 *bytes* no total de toda a operação, exigindo do ataque de força bruta a realização de $93^8 * 93^8 = 93^{16}$ operações para descobrir as senhas. A mesma abordagem é obedecida para valores maiores de dados, sempre respeitando a quantidade mínima de 16 *bytes* por operação. Para valores de mensagem maiores que 8 *bytes* e que possuam dados pré-estabelecidos e de conhecimento do invasor, ciclos de 32 *bytes* para operações de ou-exclusivos passam a ser utilizados. Com mensagens de 16 *bytes*, toda a sequência da chave será utilizada na criptografia, considerando 4 ciclos de 16 *bytes* cada. Para mensagens maiores de 16 *bytes*, a chave deverá ser rotacionada, de forma que ao atingir o total de 64 *bytes*, os valores utilizados no início serão reutilizados, mas considerando o envio constante de mensagens grandes e padronizadas, é indicado que chaves de tamanhos maiores sejam consideradas.

Uma maneira de aleatorizar a solução para que nunca a mesma sequência de *bytes* da chave seja utilizada para criptografar uma mensagem é por meio da utilização de valores

randômicos que indicam o índice da chave no qual será iniciado a operação. A figura 4.5 indica, de forma simplificada, esta operação.

Figura 4.5: Índices Randômicos.

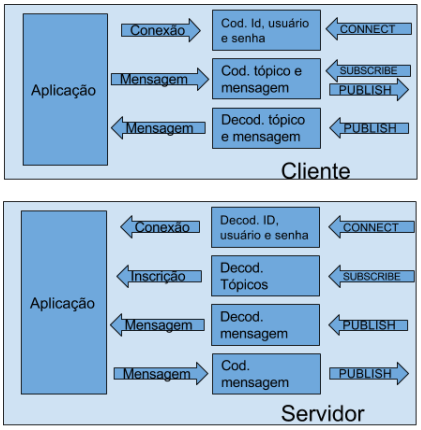


Fonte: Autoria Própria

4.5 Arquitetura

A arquitetura da solução proposta se baseia na camada de aplicação, sendo todo o processo de criptografia feito nos dados antes de serem enviados à camada de transporte, ou no momento em que a informação é recebida vindo desta mesma camada. A estrutura da solução é ilustrada na figura 4.6.

Figura 4.6: Arquitetura Cliente e Servidor.



Fonte: Autoria Própria

No cliente, a aplicação representa o programa original que iria gerar dados e mensagens para transmiti-las no sistema, ou seja, a aplicação de um sensor de temperatura é o algoritmo que irá receber os dados do dispositivo e enviá-los para a rede. No *broker*, a aplicação, é o local que será feito o gerenciamento dos usuários da rede e a transmissão de uma mensagem enviada do *publisher* para o *subscriber*.

O fluxo de funcionamento da conexão segue descrito abaixo:

1. O cliente, envia os dados de conexão ao bloco codificador de senha comum que irá criptografar ID, usuário e senha e enviar para o *broker* um pacote CONNECT cifrado.

2. O servidor, recebe os dados de conexão e no bloco decodificador extrai os dados reais de ID, usuário e senha. Nesse momento já é feita a validação do ID com o valor do usuário. Se eles não corresponderem, um retorno de CONACK com recusa de conexão é enviado. Caso os dados sejam válidos, a informação é enviada ao bloco de aplicação que faz a autenticação do usuário com sua senha e estabelece a conexão.

3. Para fazer uma inscrição, o cliente envia o tópico para um outro codificador, que utiliza a senha única para cifrar os dados e, então, a mensagem de SUBSCRIBER é enviada.

4. O *broker*, ao receber uma mensagem de SUBSCRIBER, a envia para o bloco decodificador de inscrição, que apenas descriptografa o dado e o direciona para o bloco de aplicação realizar o seu tratamento, sem a necessidade de realizar nenhum outro processamento da informação.

5. As mensagens também são criptografadas pelo mesmo bloco de senha única e então, enviados junto ao tópico, também cifrado em um pacote PUBLISH. Sendo o processo inverso realizado ao receber este mesmo tipo de mensagem.

6. Ao receber uma mensagem de PUBLISH, o *broker* realiza a descriptografia em um bloco diferente do bloco de inscrição, pois caso a mensagem precise ser traduzida, seu processamento é feito neste local. O processo inverso também vale ao realizar uma publicação, sendo que para cada um dos clientes inscritos no tópico uma nova criptografia é realizada.

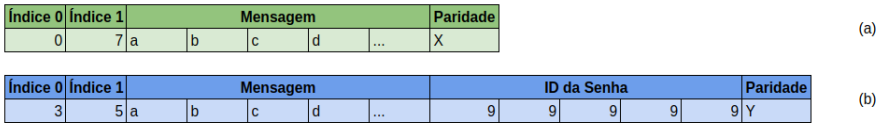
4.6 Estrutura dos dados criptografados

Cada um dos dados devem enviar qual o índice da chave utilizado para o início da criptografia. Considerando que a chave é de 64 *bytes*, valores entre 0 e 63 podem ser selecionados, portando, dois valores inteiros serão adicionados antes de cada mensagem cifrada. O ID, além do índice, carrega a identidade da chave comum, que como dito na seção 4.3.1, é uma *string* de pelo menos 10 *bytes*, que deverá ser adicionado no final do campo do ID. Ambos os tipos carregam no final um *byte* de paridade. A figura 4.7 ilustra a estrutura das mensagens.

No final, uma sobrecarga de 3 *bytes* são adicionado por dado criptografado, e outros 10 *bytes* a mais caso seja o ID. Sendo estes, valores pouco significativos, comparados a utilização do protocolo TLS na camada de transporte ou de outros algoritmos de criptografia.

Para cada pacote, um mesmo índice de chave será usado para criptografar os dados, ou seja, cada novo pacote CONNECT, PUBLISH ou SUBSCRIBE os dados codificados usarão o mesmo índice de chave e este valor nunca deverá se repetir. Para chaves de 64 *bytes*, há 64 pontos de inicialização, porém a chave deve ser trocada pelo menos a cada 64 vezes em que ela for utilizada. Este valor pode ser dobrado, considerando que a sequência possa ser formada em ordem decrescente. Este procedimento impede que um invasor replique uma mensagem criptografada para ganhar acesso ao sistema.

Figura 4.7: (a) Estrutura para os campos usuário, senha, tópico e mensagem. (b) Estrutura para o campo de ID do usuário.



Fonte: Autoria Própria

4.7 Gerador de Dados Iniciais Randômicos

Uma das partes fundamentais deste trabalho é a utilização de chaves criadas de formas aleatórias. Toda vez que um usuário é criado no servidor, sua chave é gerada e enviada ao cliente. A estrutura das chaves definida para este trabalho é de uma *string* de 64 *bytes*.

Para os tópicos, considerando que a comunicação é feita entre máquinas e há um *broker* que realiza a intermediação dos dados, todos os tópicos podem ser criados de forma randômica, não necessitando de uma ordem lógica de palavras para defini-los. Por exemplo, para o tópico *´casa/sensores/quarto/temperatura´*, um código criado pelo *broker* e repassado ao cliente com a sintaxe *´xnj/uuz/ade/xx1´* teria o mesmo significado. Quem precisa de uma sintaxe lógica para entendimento é o usuário que irá interagir com a plataforma. Um *broker* robusto conseguiria interagir com seus clientes de forma codificada.

Este modo ajudaria a reduzir também o tamanho do dado enviado. No exemplo acima, enviando o tópico em forma de palavras seriam necessários 32 *bytes*, o modo codificado enviaria menos da metade deste valor. Caberia ao *broker*, portanto, criar tabelas de codificação para os tópicos disponíveis, além de restringir quais clientes postem em quais tópicos.

Dessa mesma forma, os dados transmitidos entre o *broker* e o cliente também podem, além da criptografia, serem codificados. Esta propriedade vale principalmente para dados binários ou informações de status, que já são pré-definidos. Por exemplo, uma tomada inteligente que envia seu estado, ligado ou desligado, como sendo ´1´ e ´0´, respectivamente, poderiam enviar ´j´ para ligado e ´9´ para desligado. Essa codificação deve ser combinada entre o cliente e o *broker*, ao realizar a configuração inicial e pode ser diferente entre 2 clientes distintos, cabendo ao *broker* realizar a tradução.

4.8 A troca de chaves

Para evitar que usuários invasores tentem exaustivamente encontrar padrões nas mensagens a fim de decifrar o conteúdo transmitido, fica estabelecido que tanto a senha comum quanto a única devem ser constantemente modificadas. Esta troca ocorrerá na comunicação de PUBLISH, uma vez que uma transmissão segura e criptografada já esteja estabelecida inicialmente. Para isso, é estabelecido na configuração inicial um tópico para que o usuário se inscreva e receba as novas senhas.

O procedimento de troca de senha segue em detalhes abaixo:

1. Ao identificar que a senha atual está vencida, o cliente envia um pedido de senha por meio de uma publicação em um tópico determinado estabelecido pelo *broker*.

2. O *broker* irá gerar uma nova senha, com os mesmos padrões já estabelecidos de 64 *bytes*.

3. Para realizar o envio, esta senha será dividida em pacotes de 16 *bytes*.

4. Cada pacote é submetido à três operações de ou-exclusivo. A primeira é com o ID do usuário que irá recebê-la, seguido de uma operação com a senha e, por fim, outra com o dado de usuário. Como estes valores já são de 16 *bytes*, nenhum ajuste deverá ser realizado.

5. Um número randômico é gerado para indicar um índice da outra chave, que não será trocada.

6. Quatro operações de ou-exclusivas são feitas entre o resultado do item 3 e todos os *bytes* da chave não alterada.

7. O resultado de cada um desses pacotes junto do índice de chave é associado e enviado como uma mensagem PUBLISH em um tópico específico para troca de senhas.

8. O cliente, ao receber essa mensagem, retira o índice e realiza sua divisão, novamente, em 4 pacotes de 16 *bytes* cada.

9. Em seguida, é realizada a decodificação utilizando a chave que não será alterada, seguida do dado de usuário, da senha e do ID do cliente.

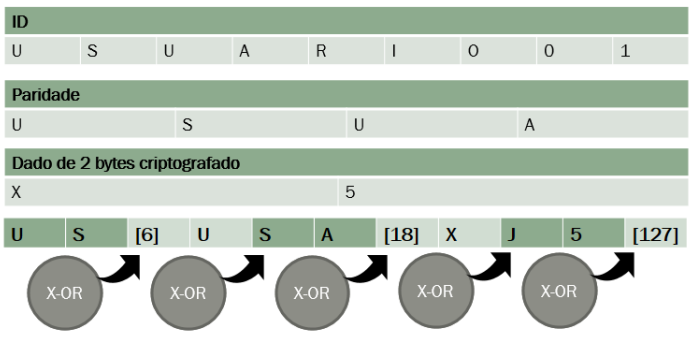
10. No final, uma mensagem tipo PUBLISH com o valor da senha antiga sendo criptografada pela senha nova utilizando os métodos de criptografia de mensagem é enviada ao *broker*.

11. O *broker* valida o recebimento correto da nova senha enviando uma publicação de confirmação para o usuário, caso os dados não estejam corretos, uma nova senha é gerada e todo o processo é reinicializado.

4.9 Paridade

Para evitar que aconteçam erros ou mudanças no conteúdo das mensagens quando elas são transmitidas, um *byte* de paridade é adicionado para cada item criptografado. A paridade é calculada de seguinte forma: os 4 primeiros *bytes* do ID do usuário são utilizados no seu cálculo, sendo inicialmente realizado operações de ou-exclusivo entre esses *bytes* em ordem do primeiro ao último e, em seguida, este resultado é utilizado também em operações de ou-exclusivo com os dados cifrados, incluindo os índices de aleatoriedade da chave. A figura 4.8 ilustra essa operação.

Figura 4.8: Operação de Paridade. Obs.: Os dados em conchetes representam o valor em decimal de um item da tabela ASCII.



Fonte: Autoria Própria

No exemplo da figura 4.8, considerando que o ID seja 'USUARIO01', são utilizados na paridade os 4 caracteres iniciais, 'USUA', que são mesclados, um a um, da esquerda para a direita por meio da operação de ou-exclusivo. O resultado gerado com essa operação é utilizado para iniciar novas operações de ou-exclusivo entre os caracteres do dado criptografado, incluindo o índice de chave utilizado. No final da operação é resultado um byte, que é enviado para o destinatário. Este, ao receber a mensagem, analisa o byte de paridade com a realização do processo inverso para obter o primeiro caractere do ID do usuário e confirmar que a mensagem não sofreu alterações.

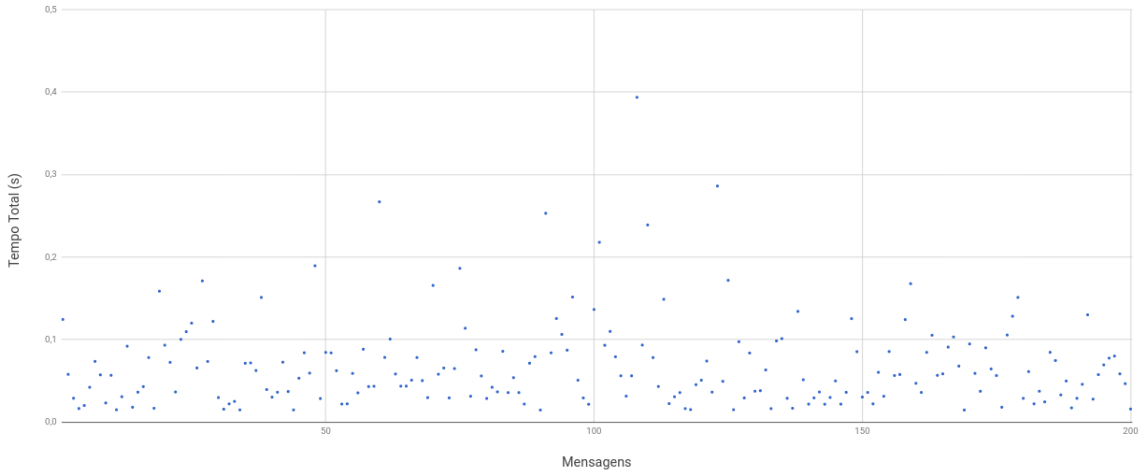
Capítulo 5

Resultados e Discussões

Por questões de teste foram instalados os clientes e o *broker* em uma rede fechada, sem que houvesse tráfegos de dados, a não ser o da comunicação MQTT. A placa Intel Galileo atuou como o *publisher*, gerando um valor aleatório entre 10 e 99, criptografando os dados e enviando-os ao *broker*. Este realizava a decodificação da informação e criptografava novamente para enviar à placa Raspberry Pi, que é o *subscriber* e que, ao receber os dados, realizava a descriptografia da informação.

Para testar a eficiência da solução foi medido o tempo desde a geração do valor aleatório, pelo *publisher*, até sua decodificação no *subscriber*, portanto, o resultado final ficou definido por $t_{total} = t_{pub} + t_{p2b} + t_{broker} + t_{b2s} + t_{sub}$, sendo t_{p2b} o tempo de envio da mensagem do *publisher* para o *broker* e t_{b2s} o tempo do *broker* ao *subscriber*. Para que o tempo total convergisse em um valor médio foi gerado uma amostra de dados, que para este caso totalizaram com o envio de 200 pacotes. O gráfico da figura 5.1 ilustra os tempos calculados. Durante todo o processo foi considerada apenas a realização da criptografia do dado no *publisher*, o recebimento, tradução e envio pelo *broker*, recebimento e decodificação no *subscriber*, além de contar com a validação da paridade em todos os pontos. Apenas um *publisher* e um *subscriber* foram utilizados. A média do tempo total foi de 68,80 ms.

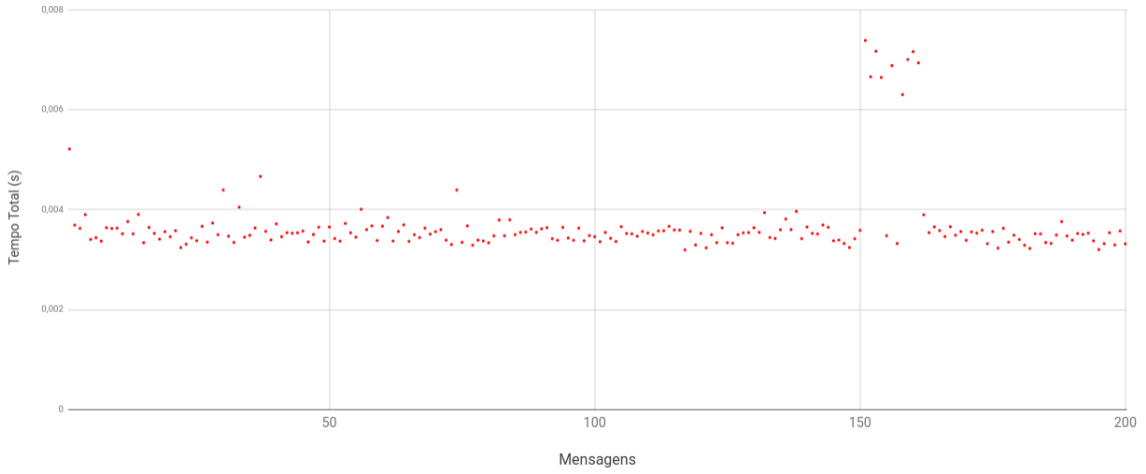
Figura 5.1: Tempo de envio de mensagens da solução proposta.



Fonte: Autoria Própria

Separando apenas os tempos do *publisher* e do *subscriber*, são obtidos os valores de 2,43 ms e 0,29 ms, respectivamente, demonstrando que a criptografia incluída nos clientes tem pouca influência no tempo total da transmissão da mensagem, sendo, portanto, o maior processamento ocorrido no *broker*. Para fins de comparação, o mesmo teste foi realizado sem a criptografia e obteve uma média de publicação de 3,67 ms. O gráfico da figura 5.2 demonstra os tempos obtidos.

Figura 5.2: Tempo de envio de mensagens sem criptografia.



Fonte: Autoria Própria

A conexão, que é o processo em que os clientes inicializam a comunicação, também foi medida. Neste caso, como ela ocorre apenas uma vez, sua medição foi separada do envio de mensagens. O tempo de estabelecimento de uma conexão foi calculado utilizando os

tempos de criptografia do ID, nome de usuário e senha, do envio de pacote para o *broker*, da autenticação dos dados e envio da mensagem CONNACK para o cliente.

Para a conexão, também foram amostrados 200 pacotes, sendo calculado a média aritmética do tempo necessário para o estabelecimento do canal de comunicação. Para a solução proposta nesta dissertação foi utilizado o tempo de conexão do *publisher*, implementado na placa Intel Galileo, que possui menor capacidade de processamento, com o *broker*, até receber a mensagem de ack. O resultado obtido com o *Publisher* é de 273,07 ms.

Separadamente, foram testados os tempos de criptografia dos dados com tamanhos entre 1 e 32 *bytes* e os tempos da geração do *byte* de paridade. Os testes foram realizados tanto na placa Intel Galileo, que possui menores recursos de processamento, quanto na placa Raspberry Pi, com um melhor hardware disponível. No total foram realizados 200 processos de criptografia e geração de paridade para cada um dos tamanhos. A tabela 5.1 apresenta a média dos resultados da Galileo e a 5.2, da Raspberry. Notavelmente, é possível observar a diferença nos tempos entre as placas, sendo que as operações ocorrem mais rapidamente nos dispositivos com maiores recursos computacionais.

Foram avaliados também as quantidades de *bytes* para cada caso durante a conexão e o envio de uma mensagem. Ficou estabelecido que os dados de ID do cliente, usuário e senha possuam 16 *bytes*, o tópico tem 35 caracteres e a mensagem apenas 2. Para enviar uma mensagem PUBLISH são utilizados 107 *bytes* sem criptografia e 113 *bytes* criptografados, enquanto que para estabelecer uma conexão são necessários 569 para esta solução e no modelo original são gastos 550 *bytes*.

Estas informações apresentam uma coerência entre os resultados da solução original MQTT e do trabalho desenvolvido neste projeto, uma vez que no pacote PUBLISH existem apenas o item de tópico e de mensagem que adicionam 3 bytes cada, devido a criptografia, somando 6 bytes. Em relação à mensagem CONNECT, são enviados os dados de usuário, senha e identidade do cliente, dos quais os dois primeiros adicionam 3 bytes cada e o último acrescenta 13 bytes ao pacote, totalizando 19 bytes a mais. Portanto, esta análise apresenta uma vantagem à solução considerando que poucos bytes são incluídos nos pacotes e, independentemente do tamanho de cada um dos dados, a criptografia adiciona apenas 3 bytes por item e 13 para o ID.

Em termos de tamanho de código, o Mosquitto *Broker* possui compilado 7.740 KB de código. Já adicionando os códigos da solução proposta, o *broker* terá compilado 12.252 KB, que, em termos de armazenamento não inviabiliza a máquina na qual o *broker* foi testado.

Para o *publisher* e o *subscriber* compilado, eles apresentam com criptografía respectivamente 8,8 KB e 8,9 KB. Enquanto os códigos sem a utilização de criptografia apresentam ambos compilados apenas 2,1 KB. Apesar da diferença de mais de 4 vezes dos códigos compilados, eles não são uma limitação para o processamento dos clientes utilizados neste trabalho.

Os códigos utilizados para testar a solução proposta podem ser encontrados em [54].

Tabela 5.1: Tempos de criptografia e geração do *byte* de paridade pela placa Intel Galileo.

Tamanho do Dado	Tempo Criptografia (s*10^{-3})	Tempo Paridade (s*10^{-6})
1	1,32	168,94
2	1,59	182,65
3	1,71	187,38
4	1,79	219,42
5	2,08	220,77
6	1,73	251,07
7	1,96	238,13
8	3,28	252,27
9	3,99	261,53
10	4,26	276,64
11	5,06	298,64
12	6,01	293,91
13	7,33	304,51
14	7,95	331,74
15	8,73	330,33
16	8,48	342,82
17	12,82	355,27
18	13,30	379,73
19	13,02	389,05
20	17,26	382,70
21	18,21	400,90
22	20,23	402,22
23	23,86	441,26
24	26,61	463,91
25	28,10	464,30
26	32,21	466,99
27	30,80	469,94
28	32,97	476,24
29	37,71	491,86
30	56,72	508,96
31	44,36	514,71
32	34,17	523,67

Tabela 5.2: Tempos de criptografia e geração do *byte* de paridade pela placa Raspberry Pi.

Tamanho do Dado	Tempo Criptografia (s*10^{-3})	Tempo Paridade (s*10^{-6})
1	0,26	22,62
2	0,28	25,75
3	0,34	29,12
4	0,31	32,03
5	0,36	35,31
6	0,32	38,37
7	0,40	41,62
8	0,58	46,22
9	0,74	48,14
10	0,92	51,27
11	0,96	54,52
12	1,12	57,34
13	1,25	60,59
14	1,57	63,78
15	1,74	67,10
16	2,06	70,30
17	2,33	73,26
18	2,91	76,57
19	2,60	79,52
20	3,17	82,61
21	4,03	86,06
22	4,05	88,87
23	5,31	91,94
24	5,10	95,82
25	5,36	98,53
26	6,76	101,87
27	6,70	104,71
28	9,17	108,94
29	8,87	110,95
30	10,01	114,35
31	9,47	117,89
32	9,10	120,89

Capítulo 6

Conclusão

Este capítulo conclui a atividade e adiciona trabalhos para realização futura.

6.1 Conclusão do trabalho

Considerando as soluções presentes relacionadas à Internet das Coisas, foi proposto neste trabalho uma forma de comunicação que, de maneira leve, transmitisse informações seguras por meio do protocolo MQTT entre os clientes e o *broker*. Esta solução define diversos serviços que combinados provêm uma maneira codificada, criptografada e randômica para a transmissão de dados, dificultando que usuários indesejados sequestrem os dados da conexão ou transmitam informações duvidosas.

Esta implementação intenta reduzir a exigência de processamento para criptografia e transmissão dos dados por parte de clientes com recursos limitados. A exigência maior fica por conta do servidor que deverá administrar todos os usuários e seus dados específicos, assim como chaves únicas de criptografia e tabelas de códigos de mensagens, e garantir que as informações recebidas e enviadas por cada cliente estejam corretas e traduzidas. Formas randômicas de se comunicar prejudicam o mapeamento dos dados e o deciframento do conteúdo.

Com as funcionalidades propostas adicionadas à comunicação MQTT é possível garantir:

- Confidencialidade: Com dados codificados e criptografados apenas o cliente e o *broker* poderão ter acesso à informação, evitando espionagem por entidades não autorizadas.

- Integridade: A paridade, utilizando de dados que apenas os clientes e o *broker* conheçam, garante que as informações originais enviadas não sejam manipuladas por tercei-

ros, além de afirmar a coerência dos dados.

- Autenticidade: Esta propriedade é garantida com a utilização de dados de identidade, usuário e senha que são únicos para cada cliente e que apenas eles e o *broker* têm conhecimento sobre o seu conteúdo.

O protótipo apresentado, realiza a criptografia da mensagem enviada pelo *publisher*, a tradução realizada pelo *broker* e a decriptografia final executada pelo *subscriber*. Nesta simulação foi confirmada a baixa exigência de banda, visto os poucos *bytes* a mais adicionados à comunicação, porém obteve um aumento significativo nos tempos de transmissão de mensagens devido à adição da criptografia, mas que não foi um impedimento para que a criptografia fosse executada e, que se esse aumento no tempo da comunicação não for uma adversidade crítica ao projeto, esta criptografia garantirá um maior nível de segurança ao protocolo MQTT.

6.2 Trabalhos futuros

Para continuação deste trabalho ficam definidas algumas possibilidades:

- Criação da interface do *broker* com o usuário para realizar a administração dos clientes
- Definição das formas de estabelecer a comunicação segura para realizar a configuração inicial dos clientes.

Referências

[1] Cisco visual networking index predicts near-tripling of ip traffic by 2020. https://newsroom.cisco.com/press-release-content?type=press-release&articleId=1771211, Acesso em: 27 de outubro de 2017.

[2] Mahmud Hossain, Ragib Hasan, and Anthony Skjellum. Securing the internet of things: A meta-study of challenges, approaches, and open problems. In *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*, pages 220–225. IEEE, 2017.

[3] Jim Chase. The evolution of the internet of things. *Texas Instruments*, 2013.

[4] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE, 2012.

[5] Tiago C de França, Paulo F Pires, Luci Pirmez, Flávia C Delicato, and Claudio Farias. Web das coisas: conectando dispositivos físicos ao mundo digital, 2011.

[6] ITU Strategy and Policy Unit. Itu internet reports 2005: The internet of things. *Geneva: International Telecommunication Union (ITU)*, 2005.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[8] Nest thermostat. https://nest.com/thermostats/nest-learning-thermostat/overview/, Acesso em: 07 de outubro de 2017.

[9] Google home. https://store.google.com/us/product/google_home?hl=en-US, Acesso em: 07 de outubro de 2017.

[10] Apple watch. https://www.apple.com/br/apple-watch-series-3/, Acesso em: 07 de outubro de 2017.

[11] R van der Meulen. Analysts to explore the value and impact of iot on business at gartner symposium/itxpo 2015, november 8-12 in barcelona, spain.

[12] The internet of things and the enterprise. http://www.gartner.com/smarterwithgartner/the-internet-of-things-and-the-enterprise/, Acesso em: 08 de outubro de 2017.

[13] H Tschofenig et al. Architectural considerations in smart object networking, tech. Technical report, No. RFC 7452, Internet Architecture Board, 2015. https://www.rfc-editor.org/rfc/rfc7452.txt, 2015.

[14] Figura 3 - device-to-device. https://www.embarcados.com.br/modelos-de-comunicacao-para-iot/, Acesso em: 08 de outubro de 2017.

[15] Figura 2 - device-to-cloud. http://www.thewhir.com/web-hosting-news/the-four-internet-of-things-connectivity-models-explained, Acesso em: 08 de outubro de 2017.

[16] Device-to-gateway. http://internetofthingsagenda.techtarget.com/feature/Using-an-IoT-gateway-to-connect-the-Things-to-the-cloud, Acesso em: 08 de outubro de 2017.

[17] Back-end data-sharing model. http://www.innvonix.com/blog/iot/iot-internet-of-things/, Acesso em: 08 de outubro de 2017.

[18] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015.

[19] Mosquitto broker. https://mosquitto.org/man/mqtt-7.html, Acesso em: 08 de outubro de 2017.

[20] Mqtt architecture. https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html, Acesso em: 08 de outubro de 2017.

[21] Z Shelby, K Hartke, C Bormann, and B Frank. Rfc 7252. *Constrained Application Protocol (CoAP). Available online: http://tools. ietf. org/html/rfc7252 (accessed on 6 August 2014)*, 2014.

[22] H Xuan. Coap protocol binding. Technical report, Hitachi (China) RD Corp., 2014.

[23] Peter Saint-Andre, Kevin Smith, and Remko Tronçon. *XMPP: the definitive guide*. "O'Reilly Media, Inc.", 2009.

[24] Figure 7-1: Xmpp network. https://www.safaribooksonline.com/library/view/beautiful-testing/9780596806934/ch07s02.html, Acesso em: 09 de outubro de 2017.

[25] Amqp. http://blog.locaweb.com.br/artigos/tecnologia/amqp-bsico-ilustrado/, Acesso em: 10 de outubro de 2017.

[26] Amqp architecture. https://blogs.vmware.com/vfabric/2013/01/messaging-architecture-using-rabbitmq-at-the-worlds-8th-largest-retailer.html, Acesso em: 10 de outubro de 2017.

[27] Rodrigo Roman, Javier Lopez, and Pablo Najera. A cross-layer approach for integrating security mechanisms in sensor networks architectures. *Wireless Communications and Mobile Computing*, 11(2):267–276, 2011.

[28] James F Kurose and Keith W Ross. *Computer networking: a top-down approach*, volume 5. Addison-Wesley Reading, 2010.

[29] Simón Singh. The code book: The science of secrecy from ancient egypt to quantum cryptography anchor books. *New York*, 1999.

[30] Marcelo C Carlos, JeandrÃ© M Sutil, Cristian Thiago Moecke, and Jonathan G Kohler. *Introdução a Infraestrutura de Chaves Públicas e Aplicações*, volume 1. Escola Superior de Redes RNP, 2010.

[31] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.

[32] Mirai source code. https://github.com/jgamblin/Mirai-Source-Code, Acesso em: 17 de outubro de 2017.

[33] Ddos attacks. https://www.incapsula.com/ddos/ddos-attacks/, Acesso em: 16 de outubro de 2017.

[34] Ddos attack against dyn. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet, Acesso em: 16 de outubro de 2017.

[35] Dyn analyze about ddos attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, Acesso em: 16 de outubro de 2017.

[36] About man-in-the-middle attack. https://www.kaspersky.com.br/blog/what-is-a-man-in-the-middle-attack/462/, Acesso em: 16 de outubro de 2017.

[37] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys & Tutorials*, 18(3):2027–2051, 2016.

[38] What is a mitm attack. https://www.incapsula.com/web-application-security/man-in-the-middle-mitm.html, Acesso em: 16 de outubro de 2017.

[39] Hackers remotely kill a jeep on the highway with me in it. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, Acesso em: 04 de dezembro de 2017.

[40] Introducing mqtt. https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt, Acesso em: 17 de outubro de 2017.

[41] Mqtt qos levels. http://www.steves-internet-guide.com/understanding-mqtt-qos-levels-part-1/, Acesso em: 17 de outubro de 2017.

[42] Mqtt messages. http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html, Acesso em: 17 de outubro de 2017.

[43] Tls. https://hpbn.co/transport-layer-security-tls/, Acesso em: 17 de outubro de 2017.

[44] Payload encryption. https://www.hivemq.com/blog/mqtt-security-fundamentals-payload-encryption, Acesso em: 17 de outubro de 2017.

[45] DocumentaÃ§Ã£o mosquitto broker. https://mosquitto.org/documentation/, Acesso em: 18 de outubro de 2017.

[46] Python paho-mqtt client. http://www.eclipse.org/paho/clients/python/, Acesso em: 18 de outubro de 2017.

[47] Diretório github paho-mqtt. https://github.com/eclipse/paho.mqtt.python, Acesso em: 18 de outubro de 2017.

[48] Raspberry pi model 2 b. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, Acesso em: 18 de outubro de 2017.

[49] Especificações raspberry pi model 2 b. http://www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-2-model-b/rpi2-model-b-hardware-general-specifications, Acesso em: 18 de outubro de 2017.

[50] Raspian. https://www.raspberrypi.org/downloads/raspbian/, Acesso em: 18 de outubro de 2017.

[51] Galileo debian. https://sourceforge.net/p/galileodebian/wiki/Home/, Acesso em: 18 de outubro de 2017.

[52] Processador intel core i5 2430m. https://ark.intel.com/pt-br/products/53450/Intel-Core-i5-2430M-Processor-3M-Cache-up-to-3_00-GHz, Acesso em: 18 de outubro de 2017.

[53] Redes de computadores. Acesso em: 05 de dezembro de 2017.

[54] Fernando Camargo de Andrade. Código da solução. https://github.com/frndcandrade/TCC-Fernando/tree/master/cipher_MQTT, Acesso em: 23 de novembro de 2017.