Pós-Graduação em Ciência da Computação

Flávio da Silva Neves

Uma Abordagem de Segurança para os Dados Transmitidos por Dispositivos em Internet das Coisas

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Flávio da Silva Neves

Uma Abordagem de Segurança para os Dados Transmitidos por Dispositivos em Internet das Coisas

Trabalho apresentado ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: Dr. *Silvio Romero de Lemos Meira*

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

N513a Neves, Flávio da Silva
Uma abordagem de segurança para os dados transmitidos por dispositivos em internet das coisas / Flávio da Silva Neves. – 2017.
88 f.: il., fig., tab.

Orientador: Sílvio Romero de Lemos Meira.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências.

1. Engenharia de software. 2. Internet das coisas. I. Meira, Sílvio Romero de Lemos (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2017-88

Flávio da Silva Neves

Uma Abordagem de Segurança para os Dados Transmitidos por Dispositivos em Internet das Coisas

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 06/03/2017.

BANCA EXAMINADORA

Prof. Dr. Vinícius Cardoso Garcia
Centro de Informática / UFPE

Prof. Dr. João Henrique Correia Pimentel
Unidade Acadêmica de Cabo de Santo Agostinho/UFRPE

Prof. Dr. Sílvio Romero de Lemos Meira
Centro de Informática / UFPE
(Orientador)

À minha família, em especial aos meus pais
por sempre me proporcionarem totais
condições para a realização dos meus estudos,
me apoiarem e incentivarem.
Aos meus irmãos.
Dedico.

AGRADECIMENTOS

Primeiramente agradeço a Deus.

Aos meus pais **José Maria e Irani**, que apesar de muitas dificuldades me deram as condições e apoio necessários para que eu pudesse estudar e também a toda minha família que sempre me apoiaram e incentivaram a continuar estudando.

Ao meu orientador, professor Dr. Silvio Romero de Lemos Meira, que me apoiou, sempre mostrando as melhores soluções, com sua experiência e conhecimento.

Aos demais docentes do programa de Pós-Graduação do Centro de Informática da UFPE pela competência com que transmitiram os conteúdos e ensinamentos.

A Herbertt Diniz, por todas as orientações e conversas, que me tranquilizaram em momentos tão importantes, principalmente durante a elaboração desta dissertação.

Agradeço a todos os amigos que tive a oportunidade de conhecer no Centro de Informática, que por muitas vezes me ajudaram durante todo o meu mestrado, pois sem as trocas de ideias esta pesquisa não seria possível.

A todos os meus amigos que de alguma forma contribuíram e participaram na realização deste trabalho.

Ao CNPQ e ao CIn – UFPE pelo apoio financeiro para realização desta pesquisa.

A todos, os meus mais sinceros agradecimentos.

RESUMO

Internet das Coisas (*IoT*) é um paradigma em que vários dispositivos (sensores e atuadores) estão interligados. Esta interconectividade de diferentes tipos de dispositivos gera desafios, tais como: gerenciamento e segurança. Um dos grandes desafios consiste em desenvolver mecanismos que garantam segurança dos dados enviados por dispositivos com recursos limitados, em redes não confiáveis. Portanto, este trabalho tem por objetivo especificar e projetar uma Arquitetura de *Software,* que forneça segurança aos dados transmitidos em sistemas *IoT* usando o protocolo MQTT em redes não confiáveis. Geralmente, uma solução que usa *Hash* dos dados gerados pelo dispositivo**,** envia esses dados (Mensagem + *Hash*) juntos. Nesse processo, os dados podem ser interceptados e alterados. Desta forma, o consumidor não identificará que houve uma alteração. Para isso, foi desenvolvida uma técnica usando funções de *Hash* criptográficas, que visa garantir o requisito confidencialidade aos dados transmitidos pelos dispositivos. Esta técnica proposta gera o *Hash* do (valor + *salt* + contador), dessa maneira, é enviado apenas o *Hash*, a aplicação consumidora terá mecanismos para traduzi-lo para o valor original. Para avaliar a técnica projetada neste trabalho foram feitos alguns testes usando 8 ferramentas para tentar decifrar os valores *Hash*, que tiveram por objetivo verificar a eficácia da técnica no que diz respeito a garantia de confidencialidade dos dados transmitidos em redes não confiáveis. Outra métrica avaliada é, o quanto de recursos do dispositivo é consumido usando esta solução. Nos testes feitos não foi possível decifrar os valores *Hash* gerados usando a técnica desenvolvida neste trabalho. Tendo em vista as avaliações realizadas conclui-se que a solução aqui projetada atinge seus objetivos que é garantir a segurança dos dados transmitidos por dispositivos com recursos limitados.

Palavras-chave: Internet das Coisas. Arquitetura de *Software*. Segurança. Hash. MQTT.

ABSTRACT

Internet of Things (IoT) is a paradigm that interconnects several devices acting like sensors and actuators. This interconnectivity of different types of devices generates challenges, such as management and safety. One of the major challenges is to develop mechanisms that ensure the security of data sent by devices with limited resources in untrusted networks. Therefore, this work aims to specify and design a Software Architecture, which provides security to the data transmitted in IoT systems using the MQTT protocol in unreliable networks. Generally, a solution that uses Hash of data generated by the device sends these data (Message + Hash) together. In this process, the data can be intercepted and changed, so the consumer would not identify that it has been changed. For this, we develop a technique using cryptographic Hash functions, which aims to guarantee integrity, confidentiality and authenticity requirements of the data transmitted by these devices. Our technique generates the Hash based on (value + salt + counter) and, in this way, only the Hash is sent. The consumer application has mechanisms to translate this Hash to the original value. To evaluate our technique we make some tests using 8 tools to try to decipher the Hash values, which aim to verify the effectiveness of this technique regarding integrity, confidentiality and authenticity of the data transmitted in untrusted networks. We also evaluate the resources consumed by each device using our technique. The results show that it is not possible to decipher the Hash values generated using our technique. This way, we conclude that our solution achieves its objectives by guaranteeing the security of the data transmitted by devices with limited resources.

Keywords: Internet of Things. Software Architecture. Security. Hash. MQTT.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

AAC - *Authorization and Access Control*

ABE - *Attribute Based Encryption*

ACM - *Digital Library*

AMQP - *Advanced Message Queuing Protocol*

APC - *AVL Particle Counter*

API - (*Application Programming Interface*)

AS - Arquitetura de *Software*

AUPS - *AUthenticated Publish/Subscribe System*

BPM - *Business Process Management - Gerenciamento de Processos de Negócio*

CA - *Certificate Authority* (autoridade certificadora)

CoAP - *Constrained Application* Protocol

CP - *Ciphertext Policy*

CSBC – Congresso da sociedade brasileira de computação

DDS – *Data Distribution Service*

EPC - *Electronic product codes*

ETSI - *European Telecommunications Standards Institute*

GE - *Generic Enablers*

GQM - *Goal/Question/Metric*

HMAC - *Hash message authentication code*

IA – Inteligência Artificial

IEEE - *Institute of Electrical and Electronics Engineers*

IETF - *Internet Engineering Task Force*

IoT – *Internet of things* (Internet da coisas)

M2M – *Machine-To-Machine*

MQTT - *Message Queue Telemetry Transport*

MVC - *Model-View-Controller*

NIST - *National Institute of Standards and Technology*

OASIS - *Advancing Open Standards for the Information Society*

PA – Padrão Arquitetural

PAC - *Presentation-Abstraction-Control* (apresentação-abstração-controle)

PKG - *Public Key Generator*

SHA - *Secure Hash Algorithm*

SLR - *Systematic Literature Review* (Revisão Sistemática da Literatura)

SMQTT – *Secure* MQTT

SMQTT-SN – *Secure* MQTT *for Sensor Networks*

TCP / IP – *Transmission Control Protocol* (Protocolo de Controle de Transmissão) / **IP** - *Internet Protocol* (Protocolo de Internet)

TLS - *Transport Layer Security*

W3C - *World Wide Web Consortium*

XMPP - *Extensible Messaging and Presence Protocol*

SUMÁRIO

1. INTRODUÇÃO

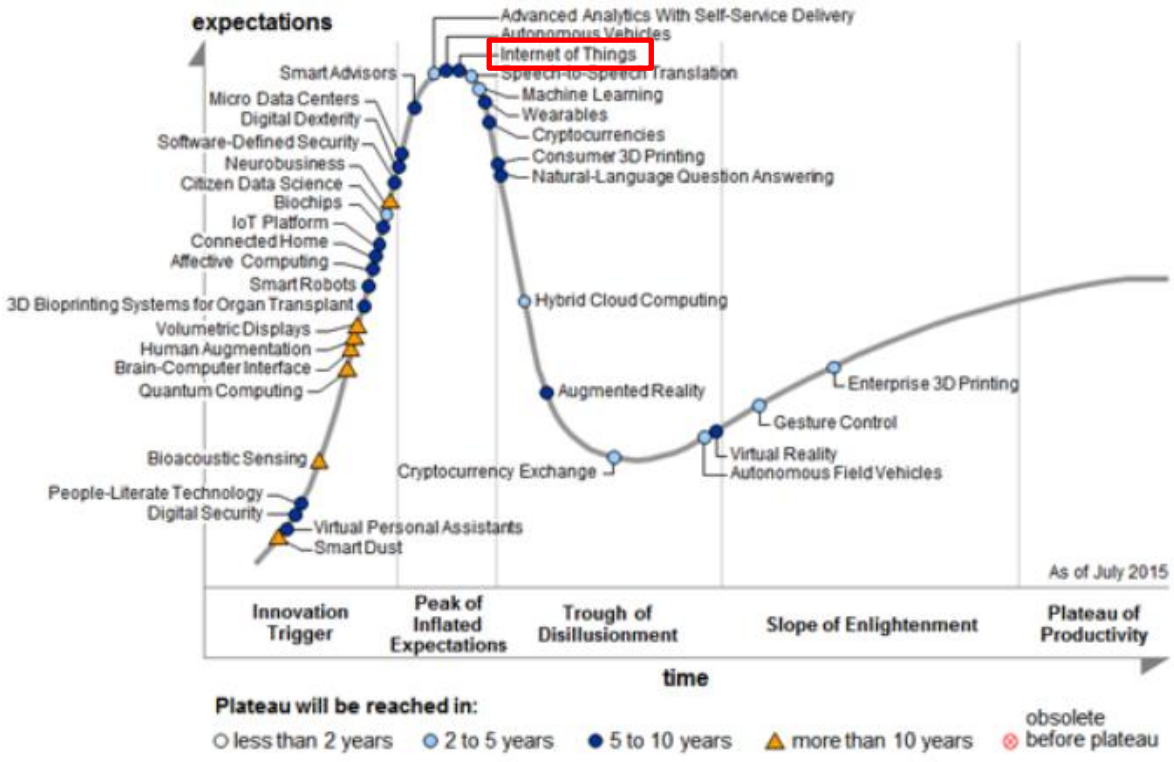
Este capítulo apresenta a motivação e justificativa para realização deste trabalho, bem como o problema e a definição da hipótese de pesquisa. A partir disso são apresentados os objetivos, geral e específicos desta pesquisa, a organização de todo o trabalho é apresentada no final deste capítulo.

1.1. MOTIVAÇÃO E JUSTIFICATIVA

A Internet das Coisas ou *IoT*, acrônimo do inglês *Internet of Things*, é uma área de pesquisa relativamente nova que está em crescente expansão. *IoT* possibilita a comunicação entre as "*coisas inteligentes*" e a interação dos seres humanos com essas "*coisas*" (ZHOU; ZHANG, 2014). Segundo o Gartner (2015a), a *IoT* foi considerada uma das dez melhores tendências estratégicas de 2016, dado que esse paradigma prevê um mundo onde tudo está conectado, desde os mais diversos sensores (ex. sensor de temperatura, de luminosidade ou até mesmo de presença) a objetos do dia a dia (cafeteira, geladeira, lâmpadas), possibilitando o monitoramento e o controle desses objetos remotamente (COLLINA et al., 2014).

A *IoT* está recebendo muitos investimentos, assim, como mostra Marek (2017). A SK Telecom investirá US$9 bilhões nos próximos três anos em Inteligência Artificial (AI), Internet das Coisas (*IoT*), veículos autônomos e 5G, o que incentiva novas pesquisas na área. Em um levantamento feito pelo Gartner (2015b), observou-se que entre as tecnologias emergentes para os próximos 10 anos, a *IoT* está no topo, assim, a partir da Figura 1, é possível visualizar essas informações.

Figura 1: Tecnologias emergentes em 2015.



Fonte: Gartner (2015b).

Com a vasta possibilidade de utilização para *IoT*, a quantidade de dispositivos e sensores está crescendo de maneira acelerada. Estudos preveem que em 2020 existirá mais de 50 bilhões de dispositivos conectados em todo o mundo com as mais diversas finalidades (CISCO, 2015; GARTNER, 2015a; NIRUNTASUKRAT; ISSARIYAPAT; PONGPAIBOOL, 2016). A comunicação entre esses bilhões de dispositivos torna-se difícil de ser realizada, pelo fato de não haver padrões de comunicação. Cada fabricante produz equipamentos com configurações específicas. Que, por sua vez, usam protocolos para comunicação restritos, o que torna complexa a comunicação entre dispositivos de fabricantes diferentes (SINGH et al., 2015). Vários problemas podem ser encontrados nessa área, dentre eles destacam-se a heterogeneidade, identidade, gerenciamento e segurança na comunicação dos dispositivos (SINGH et al., 2015).

Atualmente, existem vários protocolos para a comunicação de dados em *IoT*, tais quais: **AMQP**[1], **XMPP**[2], **CoAP**[3], **MQTT**[4], **DDS**[5]. Contudo, até o momento, não foi adotado

[1] https://www.amqp.org/
[2] https://xmpp.org/
[3] http://coap.technology/spec.html
[4] http://mqtt.org/
[5] http://www.omg.org/spec/DDS/

um protocolo padrão que permita a comunicação entre diferentes dispositivos (LI; XU; ZHAO, 2015). Em 2013, a OASIS (*Advancing Open Standards for the Information Society*) formou uma comissão técnica para propor a adoção do MQTT como protocolo padrão (HAMIDA et al., 2013). Diante disso, o protocolo MQTT vem sendo adotado em vários projetos de *IoT* nos últimos anos (NIRUNTASUKRAT; ISSARIYAPAT; PONGPAIBOOL, 2016). Para Hamida et al. (2013), embora o MQTT ainda não seja o protocolo padrão, ele está atraindo muito interesse em pesquisas nas universidades e na comunidade industrial. Existem várias implementações *Open Source* e comerciais deste protocolo que estão atualmente em execução em vários ambientes de produção da *IoT*.

1.2. PROBLEMA

Na pesquisa de Mineraud et al. (2016)**,** foram avaliadas 39 plataformas para Internet das Coisas (*IoT*), tanto de origem proprietária como *Open Source*, com base em sua capacidade de satisfazer as expectativas dos diferentes utilizadores de *IoT*. Os seguintes critérios foram usados para fazer essa pesquisa:

1. Suporte para dispositivos heterogêneos;
2. Tipo da plataforma;
3. Arquitetura;
4. Se é do tipo *Open Source*;
5. Se usa Rest;
6. Controle de acesso a dados; e
7. Serviço de descoberta.

Em sua pesquisa, Mineraud et al. (2016) fez um sumário das lacunas na *IoT*, dentre elas destaca-se a segurança dos dados transmitidos por dispositivos em sistemas para *IoT*, que ainda tem muitos problemas para serem resolvidos, pois como os dispositivos *IoT* têm processamento limitado, fornecer mecanismos de segurança é algo extremamente desafiador. A segurança na transmissão dos dados em dispositivos *IoT* é um dos grandes problemas desta área (JANG et al., 2016).

Embora existam muitas pesquisas em segurança nas últimas décadas, muitas das soluções existentes não são adequadas para os sistemas de *IoT*, devido a requisitos e restrições únicos desta área, tais como: recursos limitados e comunicações *over-the-air* (NTULI; ABU-MAHFOUZ, 2016).

Diante do exposto acima, o problema de pesquisa se evidência através da seguinte questão: Existe uma falta de segurança aos dados que são transmitidos por dispositivos com recursos limitados (dispositivos *IoT*) em redes não confiáveis usando o protocolo MQTT.

Diante do problema apresentado, o presente trabalho propõe o desenvolvimento de uma abordagem de segurança em uma arquitetura de referência para os dados transmitidos em sistemas *IoT* usando o protocolo MQTT.

1.3. HIPÓTESE

Considerando que a Internet das Coisas é uma área que está em crescente expansão, mas que ainda apresenta vários desafios a serem superados, como a heterogeneidade dos dispositivos, consumo de energia, dispositivos com recursos limitados e segurança. Desses desafios, um que se destaca bastante é a segurança dos dados transmitidos pelos dispositivos, então como garantir que invasores não tenham acesso às informações transmitidas em redes não confiáveis, que é um cenário comum desta área. Diante disto, a hipótese que conduz este trabalho é: Utilizando valores *Hash* para transmitir os dados gerados, por dispositivos, é possível atender ao requisito de confidencialidade das mensagens enviadas.

1.4. OBJETIVOS

Esta pesquisa visa compor e especificar uma técnica para fornecer segurança aos dados em *IoT*, diante disso, os objetivos da mesma são divididos em geral e específicos, listados a seguir:

1.4.1. Objetivo Geral

Este trabalho tem como objetivo geral compor e especificar uma técnica que forneça segurança a nível de confidencialidade aos dados transmitidos por dispositivos com recursos limitados em sistemas *IoT*, usando o protocolo MQTT.

1.4.2. Objetivos Específicos

1. Compor uma técnica que aperfeiçoe a confidencialidade dos dados que serão enviados por sensores com recursos limitados para servidores *Web*;
2. Desenvolver uma prova de conceito que demonstre a eficácia da técnica quanto à confidencialidade dos dados transmitidos usando a mesma;
3. Especificar e projetar uma Arquitetura de Software com a técnica criada embarcada para fornecer a confidencialidade dos dados transmitidos dos dispositivos para a *Web;*
4. Realizar testes para avaliar a eficácia da técnica proposta, no que diz respeito à confidencialidade dos dados e consumo de memória dos dispositivos.

1.5. ORGANIZAÇÃO DA DISSERTAÇÃO

O restante deste trabalho está organizado da seguinte maneira:

No capítulo 2, é apresentado o referencial teórico, que contém as principais definições que guiam esta dissertação e a revisão da literatura e os trabalhos relacionados utilizados para realizar o estudo, com seu quadro metodológico, as etapas da pesquisa e os processos envolvidos na seleção, extração, análise e síntese desta revisão da literatura. No capítulo 3, serão apresentadas a proposta da arquitetura e a técnica usada para fornecer a segurança. No capítulo 4, são apresentados os resultados e as avaliações do trabalho. Finalmente, o capítulo 5 é composto pelas limitações, dificuldades, os trabalhos futuros e as conclusões.

2. REFERENCIAL TEÓRICO

Neste capítulo, são descritos todos os conceitos e definições necessários para o entendimento deste trabalho e a revisão da literatura.

2.1. INTERNET DAS COISAS

Na literatura é possível encontrar várias definições para o termo Internet das Coisas (*IoT*), contudo ainda não existe um consenso geral a respeito deste conceito. No decorrer deste trabalho, serão apresentadas algumas das definições encontradas na literatura.

A *Internet das Coisas* (*IoT*) consiste em dispositivos inteligentes onipresentes e objetos com sensores/atuadores embutidos. Segundo Ashton (2009), o termo *"Internet das Coisas*" (*IoT*) foi mencionado pela primeira vez em 1999, como título de uma apresentação que ele fez na empresa ***Procter & Gamble (P & G)***.

O paradigma *Internet das Coisas,* prevê um mundo onde tudo está conectado e pode ser monitorado e controlado remotamente. É possível melhorar a eficiência e reduzir os custos, se os dados relevantes são coletados e analisados. Além disso, controlar remotamente casas pode dirigir a uma nova onda de ganhos de eficiência energética (COLLINA et al., 2014), proporcionando um ambiente inteligente, conectando coisas e pessoas (ZHOU; ZHANG, 2014).

Na Figura 2, é possível ter uma visão de alto nível da *IoT,* nela são apresentados vários dispositivos (Carros, objetos do dia a dia e até mesmo aviões) interconectados. Nesta pesquisa será adotada a definição de *IoT* apresentados por Santos et al. (2016), o autor diz que a Internet das Coisas "nada mais é que uma extensão da Internet atual", para proporcionar que objetos do dia-a-dia (quaisquer que sejam), se conectem a Internet" e permitir que os próprios objetos, sejam acessados como provedores de serviços (SANTOS et al., 2016).

Figura 2: Visão de IoT.

Fonte: Singh (2016)**.**

Umas das grandes características e desafios da *IoT* é a comunicação entre dispositivos heterogêneos (dispositivos com capacidade de processamento diferente), com diferentes formas de conexão (Wi-Fi, Bluetooth, 4G), ou seja a falta de padronização é uma marca da *IoT* atualmente (AMARAN et al., 2015; FRIGIERI; MAZZER; PARREIRA, 2015; JUNG et al., 2015). A Figura 3, mostra uma visão dos elementos que compõe a *IoT*..

Figura 3: Os elementos da IoT.

Fonte: Adaptado de Al-Fuqaha et al. (2015)**.**

A seguir são descritos cada um dos elementos que compõem a *IoT*, segundo a visão de Al-fuqaha et al. (2015):

1. **Identificação:** É crucial para a *IoT* nomear e combinar serviços com sua demanda. Muitos métodos de identificação estão disponíveis para o *IoT*, tais como códigos de produtos eletrônicos (EPC) e códigos ubíquos (*uCode*). Além disso, o endereçamento dos objetos *IoT* é crítico para diferenciar entre o ID do objeto e seu endereço. *Object*

ID refere-se ao seu nome como "T1" para um sensor de temperatura específico e o endereço do objeto refere-se ao seu endereço dentro de uma rede de comunicações.

2. **Detecção:** Significa reunir dados de objetos relacionados dentro da rede e enviá-los de volta para um *data warehouse*, banco de dados ou nuvem. Os dados coletados são analisados para tomar ações específicas com base nos serviços necessários.
3. **Comunicação:** As tecnologias de comunicação *IoT* conectam objetos heterogêneos juntos para fornecer serviços inteligentes específicos. Normalmente, os nós *IoT* devem operar com baixa potência na presença de *links* de comunicação com perdas e ruídos.
4. **Computação:** As unidades de processamento (por exemplo, microcontroladores, microprocessadores) e aplicações de *software* representam o "cérebro" e a capacidade computacional da *IoT*. Várias plataformas de *hardware* foram desenvolvidas para executar aplicações *IoT* como por exemplo o Arduino.
5. **Serviços:** Em geral os serviços *IoT* podem ser classificados em quatro classes: serviços relacionados à identidade, serviços de agregação de informações, serviços de colaboração e serviços *ubíquos*.
6. **Semântica:** Na *IoT* refere-se à capacidade de extrair conhecimento inteligentemente por diferentes máquinas para fornecer os serviços necessários. A extração de conhecimento inclui a descoberta e o uso de recursos e informações de modelagem. Além disso, ele inclui o reconhecimento e a análise de dados.

Um dos requisitos mais discutidos e estudados em projetos *IoT* é a interoperabilidade de objetos, na qual cada objeto é uma abstração de um sensor, atuador ou qualquer outro dispositivo capaz de realizar algum tipo de computação (TOMAS, 2014). De fato, este é um requisito crítico para a consolidação de qualquer plataforma que utilize uma vasta gama de objetos com diferentes especificações técnicas.

A interoperabilidade entre uma ampla variedade de sensores e dispositivos, a coleta e análise dos dados dos sensores e dispositivos, a detecção de contexto, a análise preditiva, a geração de saídas e atuação são alguns dos aspectos importantes de aplicações da *IoT*. Aplicações para *IoT* requerem o acesso ao sensor de dados em tempo real de acordo com as suas necessidades (BANDYOPADHYAY; BHATTACHARYYA, 2013).

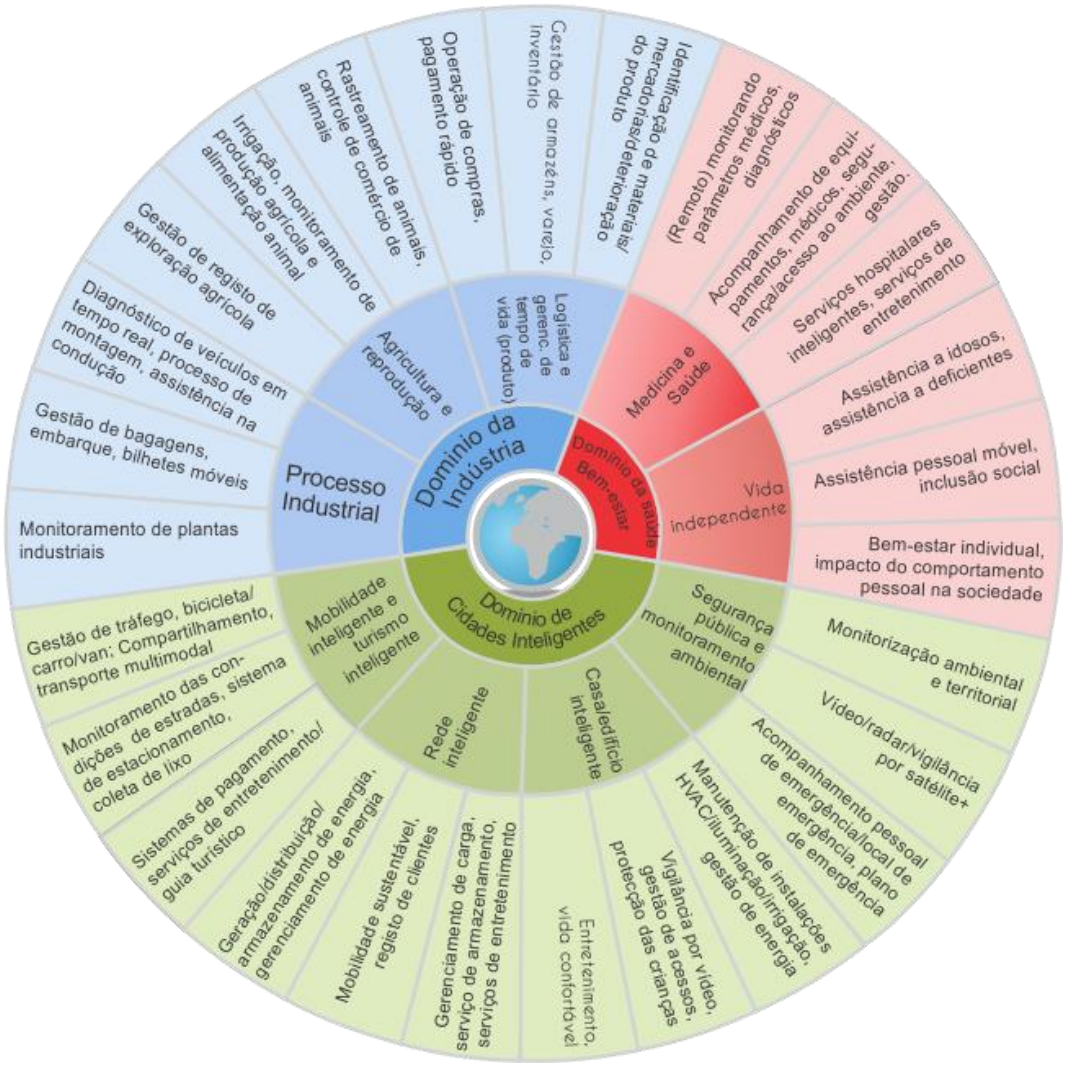
2.1.1. Aplicabilidade

Para Borgia (2014), a *IoT* tem enormes potencialidades para o desenvolvimento de novas aplicações inteligentes em quase todos os campos. Isto acontece principalmente devido à sua dupla capacidade de executar e detectar situações (como, por exemplo, coletar informações sobre fenômenos naturais, parâmetros médicos, ou os hábitos do usuário), e a partir dessas informações oferecer serviços personalizados.

Independente do campo de aplicação, a *IoT* destina-se a oferecer uma melhor qualidade de vida às pessoas, e terá um profundo impacto sobre a economia e a sociedade (BORGIA, 2014). A autora agrupa as aplicações em três grandes domínios: (A) domínio industrial, (B) domínio de cidades inteligentes, e (C) domínio da saúde e bem-estar, que serão descritos nas seções a seguir. A autora complementa afirmando que cada domínio não é isolado, mas são parcialmente sobrepostos, uma vez que algumas aplicações são compartilhadas.

Na Figura 4, é possível visualizar uma subdivisão dos domínios mencionados acima e uma lista de aplicações da *IoT* para cada um deles. Nem todas as aplicações da Internet das Coisas tem atualmente o mesmo nível de maturidade. Alguns aplicativos, normalmente os mais simples e mais intuitivos para o usuário, já fazem parte de nossas vidas diárias. Muitos outros ainda estão em fase experimental, pois eles exigem uma maior cooperação entre os vários intervenientes. Finalmente, outros são mais futuristas e estão em um estágio inicial (BORGIA, 2014). Resumindo, a Figura 4, apresenta uma visão de como ou onde a *IoT* vem sendo e vai ser aplicada.

Figura 4: Domínios de aplicação da Internet das Coisas e aplicações relacionadas.



Fonte: Adaptado de Borgia (2014).

A *IoT* pode ser aplicada nas mais diversas áreas como forma de melhora-las, no caso do domínio da indústria pode otimizar os processo de produção dos bens de consumo, já no caso do domínio de cidades inteligente e saúde e bem-estar pode ajudar a melhorar a qualidade de vida das pessoas. A seguir são apresentadas algumas aplicações importantes, baseado no trabalho de Borgia (2014):

2.1.1.1. Domínio Industrial

A *IoT* pode ser explorada em todas as atividades industriais que envolvem desde transações comerciais à financeiras. Alguns exemplos são: logística, manufatura, acompanhamento de

processos, setor de serviços, bancos, autoridades governamentais financeiras, intermediários, etc. (BORGIA, 2014).

Exemplos:

1. Gestão logística e tempo de vida do produto;
2. Agricultura e criação de animais;
3. Processos industriais.

2.1.1.2. Domínio de Cidades Inteligentes

A *IoT* pode ajudar a aumentar a sustentabilidade ambiental das cidades e a qualidade de vida das pessoas, com ênfase na energia e como controlá-la de forma eficiente, e na busca de soluções inteligentes para aproveitar a estadia pessoal (BORGIA, 2014).

Exemplos:

1. Mobilidade inteligente e turismo inteligente;
2. *Smart grid*;
3. Casas/edifícios inteligentes;
4. Segurança pública e monitoramento ambiental.

2.1.1.3. Domínio de Saúde e bem-estar

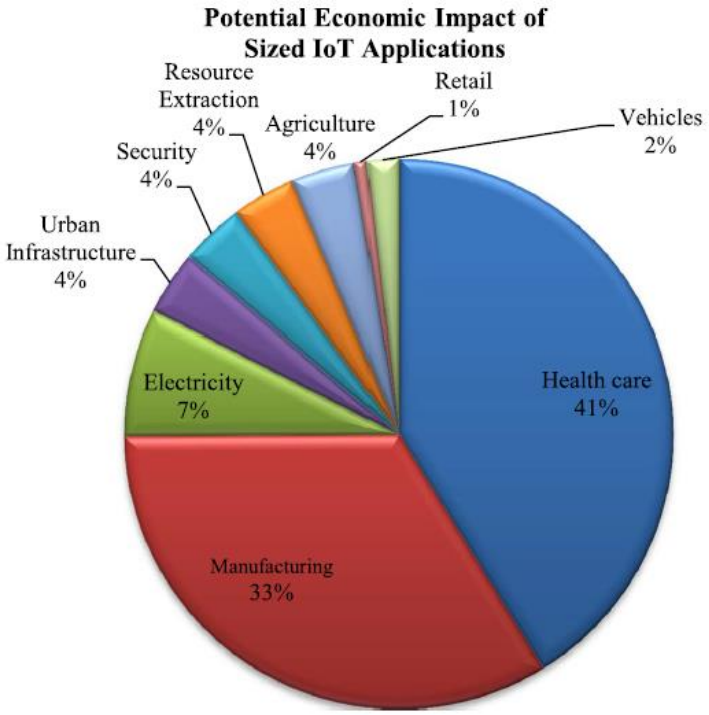
Para Borgia (2014), a *IoT* vai desempenhar um papel essencial no desenvolvimento de serviços inteligentes para apoiar e melhorar a saúde das pessoas e as atividades da sociedade. Estes vão desde os cidadãos e as comunidades que permitam envolver-se nas decisões da administração e do governo, permitindo que as pessoas vivam de forma independente ou para manter suas relações sociais, para melhorar a saúde e assistência social.

Exemplos:

1. Médico e cuidados de saúde;
2. Vida independente.

Na pesquisa de Al-Fuqaha et al. (2015), o autor apresenta um gráfico que demostra as projeções para aplicações da *IoT* que serão dominantes no mercado em 2025, é possível observar que segundo as previsões do autor, duas áreas (Industrial e Saúde) vão dominar o mercado da *IoT*. Essas informações são expostas na Figura 5.

Figura 5: Projeções para aplicações *IoT* em 2025.



Fonte: Al-Fuqaha et al. (2015)**.**

Al-Fuqaha et al. (2015), afirma que "a *IoT* oferece uma grande oportunidade de mercado para fabricantes de equipamentos, provedores de serviços de Internet e desenvolvedores de aplicativos". A Figura 6, ilustra as várias áreas que segundo o autor utilizam a *IoT*.

Figura 6: O quadro geral da *IoT* enfatizando os mercados verticais e a integração horizontal entre eles.

Fonte: Adaptado de Al-Fuqaha et al. (2015).

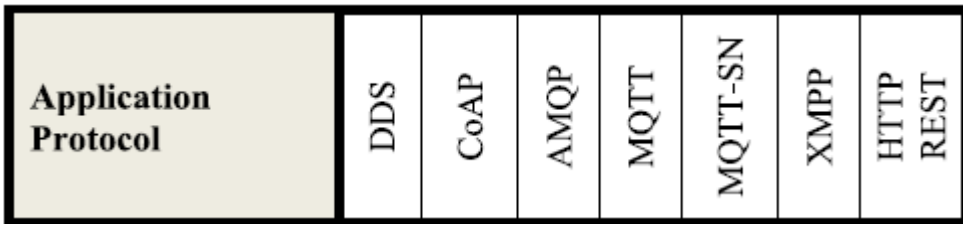
Na Figura 6, é ilustrado o conceito geral do *IoT* em que cada aplicação específica está interagindo com serviços independentes de domínio, enquanto que em cada domínio os sensores e atuadores se comunicam diretamente uns com os outros gerando informações que podem ser compartilhadas entre os mesmos.

2.2. PROTOCOLOS DE APLICAÇÃO PARA *IOT*

Atualmente, existem muitos protocolos para a *IoT*, objetivando facilitar e simplificar o trabalho dos programadores de aplicativos e prestadores de serviços. Vários grupos foram criados com o propósito de fornecer protocolos de apoio à *IoT*, dentre eles: o *World Wide Web Consortium (W3C), Internet Engineering Task Force (IETF), EPCglobal, Institute of Electrical and Electronics Engineers (IEEE), Advancing Open Standards for the Information Society* (OASIS) e o *European Telecommunications Standards Institute (ETSI)* (AL-FUQAHA et al., 2015). A Figura 7 fornece uma visão dos protocolos mais proeminentes definidos por esses grupos, que são os Seguintes: DDS, CoAP, AMQP, MQTT, MQTT-SN[6], XMPP E HTTP/REST.

[6] http://mqtt.org/2013/12/mqtt-for-sensor-networks-mqtt-sn

Figura 7: Principais protocolos de aplicação para IoT.



Fonte: adaptado de Al-Fuqaha et al. (2015)**.**

Até o momento, não adotou-se um protocolo como padrão para *IoT* (LI; XU; ZHAO, 2015). Em 2013, a OASIS (*Advancing Open Standards for the Information Society*) formou uma comissão técnica para propor a adoção do MQTT como protocolo padrão (HAMIDA et al., 2013), o que segundo os autores lhe dá grande chance de ser realmente o protocolo padrão para a *IoT*. Segundo Niruntasukrat; Issariyapat e Pongpaibool (2016), o referido protocolo ganhou bastante força nos últimos anos. Para Hamida et al. (2013), o MQTT ainda não foi adotado como um protocolo padrão, entretanto, ele está atraindo muito interesse em pesquisas nas universidades e na comunidade industrial. Existem várias implementações *Open Source* e comerciais deste protocolo que estão atualmente sendo executados em vários ambientes de produção da *IoT,* na seção 2.2.2 serão descritos alguns exemplos dessas implementações.

2.2.1. Protocolo MQTT

O *Message Queue Telemetry Transport* (MQTT) é um protocolo da camada de aplicação usado para fazer a comunicação entre dispositivos para *IoT*. O MQTT foi desenvolvido pela IBM em 1999 e, em seguida, liberado para a comunidade *Open Source*. O padrão mais recente para MQTT é a versão 3.1.1, definido pela OASIS em 2013 (HAMIDA et al., 2013). O protocolo MQTT foi projetado para dispositivos extremamente limitados tanto em termos de memória como processamento. Utiliza a estratégia de *publish/subscribe* (publicação/assinatura) para transmitir mensagens, objetivando minimizar o uso de largura de banda da rede e recursos do dispositivo (AMARAN et al., 2015; IBM; EUROTECH, 2010).

Além disso, o MQTT utiliza o TCP/IP como protocolos da camada de transporte e da camada de rede, respectivamente. O cabeçalho do protocolo MQTT pode assumir tamanho fixo ou variável, sendo que o tamanho fixo possui um cabeçalho com dois bytes. O protocolo pode operar com três qualidades de serviço distintas (IBM; EUROTECH, 2010), que são as seguintes:

1. **QoS 0**: No máximo uma vez, onde as mensagens são entregues de acordo com o melhor esforço das camadas TCP / IP subjacente;
2. **QoS 1**: Pelo menos uma vez, é assegurada a entrega da mensagem, mas com possíveis repetições;
3. **QoS 2**: Apenas uma vez, é assegurada a entrega da mensagem exatamente uma vez sem a possibilidade de repetições.

Muitas aplicações utilizam o MQTT, tais como: *health care*, monitoramento, medidor de energia. Um exemplo de aplicação muito conhecida que usa o MQTT é o *Facebook*, ele o usa em seu sistema de notificação. Portanto, o MQTT representa um protocolo de mensagens ideal para as comunicações da *IoT* e M2M (*Machine-To-Machine*) (AL-FUQAHA et al., 2015), sendo capaz de fornecer o roteamento, pequeno, barato e de baixa potência para dispositivos com pouca memória em redes com baixa largura de banda.

Estas características, o tornam ideal para uso em ambientes restritos, como por exemplo, quando recursos de rede são caros, com baixa largura de banda, ou seja, redes não confiáveis, ou quando ele é executado em um dispositivo incorporado com recursos de processador ou memória limitados (LUZURIAGA et al., 2015).

Na Figura 8, é detalhado o formato de cabeçalho fixo das mensagens MQTT. O cabeçalho de cada tipo de mensagem MQTT contém um tamanho fixo, composto por dois *bytes*, onde o primeiro *byte* contém o campo que identifica o tipo de mensagem e também campos marcadores (DUP, nível QoS e RETAIN), o segundo *byte* contém o campo de comprimento restante (IBM; EUROTECH, 2010).

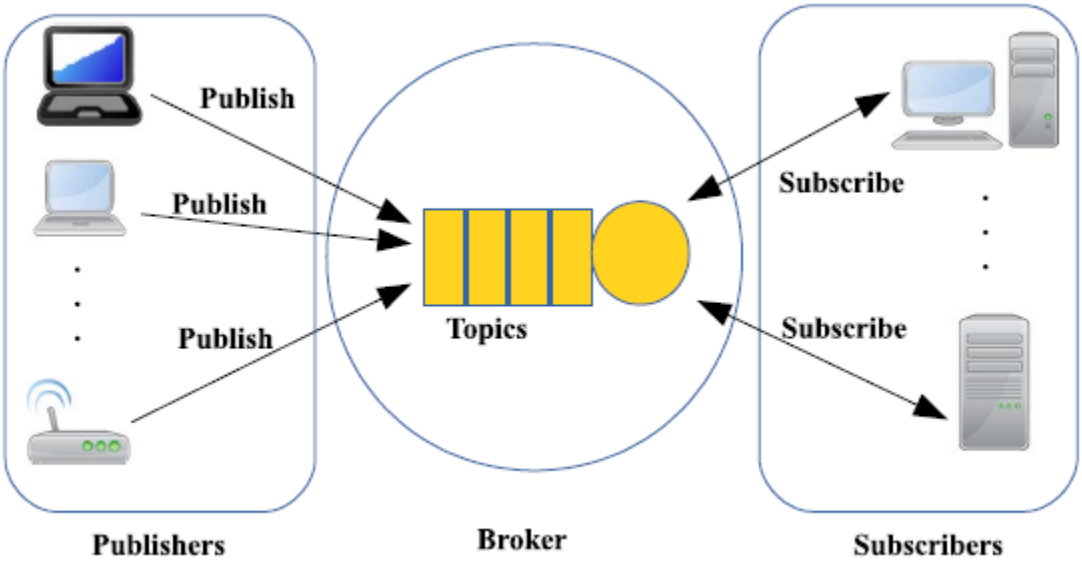
Figura 8: Cabeçalho MQTT.

bit	7	6	5	4	3	2	1	0
Byte 1	Tipo da Mensagem				Flag DUP	Nível QoS		RETAIN
Byte 2	Largura restante							

Fonte: Adaptado de IBM e Eurotech (2010).

Na Figura 9, é possível visualizar o funcionamento do protocolo MQTT. Para o funcionamento do MQTT, é necessário ter três componentes básicos: o *Subscriber*, o *Publisher* e o *Broker*. Inicialmente, o dispositivo se registra (*subscribe*) em um *Broker* para obter informações sobre dados específicos, para que o *Broker* os avise sempre que publicadores (*publishers*) publicarem os dados de interesse. Os dispositivos inteligentes (*publishers*) transmitem informações para os *subscriber* por meio do *Broker* (IBM, 2010).

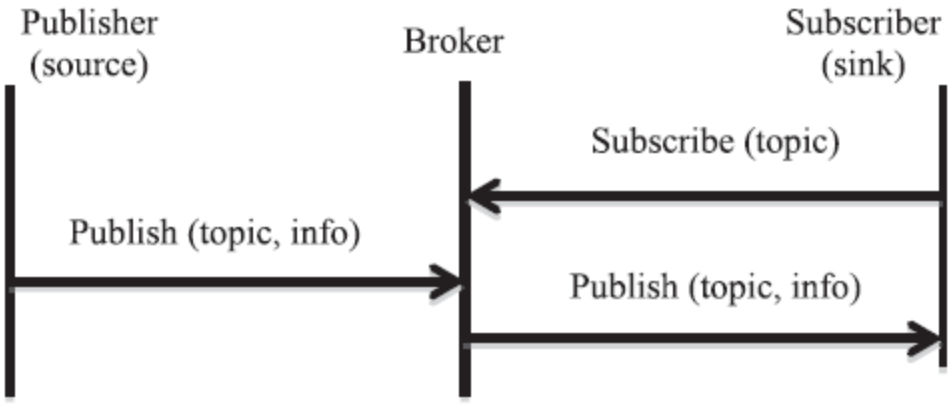
Figura 9: Arquitetura MQTT.



Fonte: Al-Fuqaha et al. (2015).

O MQTT foi projetado especificamente para dispositivos limitados, tais como, sensores e/ou atuadores. Portanto, ele tem espaço extremamente pequeno e exigência mínima de largura de banda, tornando-o adequado para aplicações da *IoT* (NIRUNTASUKRAT; ISSARIYAPAT; PONGPAIBOOL, 2016), a Figura 10, demostra o processo de publicação de informações deste protocolo, onde o *Publisher* publica um tópico para o *Broker* , o *Subscribe* se inscreve nesse tópico, então o Broker envia as informações para o *Subscribe*.

Figura 10: Processo de publicação e assinatura usado pelo MQTT.



Fonte: Adaptado de IBM e Eurotech (2010).

2.2.2. Brokers MQTT

Como mencionado anteriormente, uma solução baseada em MQTT possui três componentes: o *Broker* e os clientes (*subscriber* e *publisher*), que publicam e assinam tópicos. Há várias implementações em *software* livre para o *Broker* MQTT (*HiveMQ*[7], *ActiveMQ*[8], *RabbitMQ*[9], *CloudMQTT*[10], Mosca[11], *Mosquitto*[12], entre outros).

Uma implementação bastante popular de *Broker* MQTT é o *Mosquitto*. Segundo Kraijak e Tuwanut (2015), o *Broker Mosquitto* é uma implementação *Open Source* muito utilizada. O *Mosquitto* é um *Broker* leve e, em função disto, ele pode ser executado em sistemas relativamente limitados. Contudo, ele continua a ser poderoso o suficiente para uma ampla gama de aplicações, como por exemplo o *Facebook* (MOTWANI et al., 2016). Outra implementação *Open Source* de um *Broker* MQTT é o *RabbitMQ*, que foi escrito em *Erlang* e segundo seu site[13], é robusto, fácil de usar, roda nos principais sistemas operacionais e suporta enorme número de plataformas de desenvolvimento.

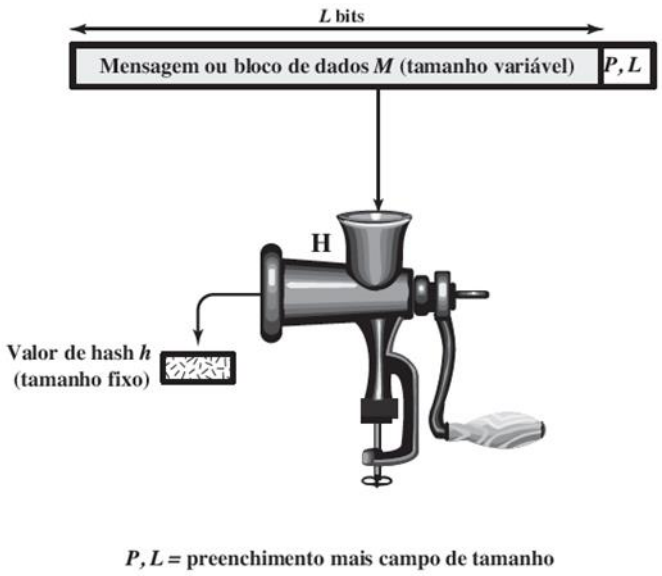
2.3. FUNÇÕES DE HASH CRIPTOGRÁFICAS

A função *Hash* é um resumo de tamanho fixo de uma mensagem ou arquivo, independente do tamanho deste último. É uma função matemática unidirecional, ou seja, não é possível obter o texto original a partir do *Hash* gerado pelo algoritmo (CONCEIÇÃO, 2014).

A Figura 11, mostra uma abstração de como funciona uma função de *Hash*. Uma função de *Hash H*, recebe uma mensagem de tamanho variável **M** como entrada e produz um valor de *Hash* de tamanho fixo h = H(M). Uma "boa" função de *Hash* tem a propriedade que leva os resultados da função a um grande conjunto de entradas que produzirão saídas distribuídas por igual. Em termos gerais, o objeto principal de uma função de *Hash* é a integridade de dados. Uma mudança em qualquer *bit* ou *bits* em **M,** resulta com alta probabilidade, em uma mudança no código de *Hash* (IETF, 2011; STALLINGS, 2014).

[7] http://www.hivemq.com/
[8] http://activemq.apache.org/
[9] https://www.rabbitmq.com/
[10] https://www.cloudmqtt.com/
[11] https://github.com/mcollina/mosca
[12] https://mosquitto.org/
[13] https://www.rabbitmq.com/

Figura 11: Função de Hash criptográfica; h = H(M).



Fonte: Stallings (2014).

O tipo de função de *Hash* necessária para aplicações de segurança é conhecido como função de *Hash* criptográfica, uma das vantagem desse algoritmo é a sua alta criptografia, o que torna computacionalmente inviável tentar quebrar a chave, por meio da técnica de força bruta (STALLINGS, 2014). As funções de *Hash* tem uma característica que é chamada de propriedade de mão única, ou seja, uma vez gerado *Hash* de um valor, não é possível fazer o caminho contrário, que é retornar ao valor original (IETF, 2011). Por conta dessa característica, as funções de *Hash*, são usadas com frequência para determinar se os dados mudaram ou não (STALLINGS, 2014). Um dos algoritmos criptográficos mais versáteis é a função de *Hash* criptográfica. Ela é usada em diversas aplicações de segurança (STALLINGS, 2014).

Segundo Stallings (2014), a essência do uso de uma função de *Hash* para autenticação de mensagem é o seguinte, o emissor calcula um valor de *Hash* da mensagem e transmite o valor de *Hash* e a mensagem juntos. O receptor realiza o mesmo cálculo de *Hash,* sobre a mensagem recebida e compara esse valor com o valor de *Hash* recebido. Se houver, uma divergência, o receptor saberá que a mensagem (ou possivelmente o valor de *Hash*) foi alterada. Na Figura 17, é possível visualizar a essência do funcionamento do Hash.

Uma função de *Hash* muito utilizada é o *Secure Hash Algorithm (*SHA*)* (STALLINGS, 2014). O SHA, foi desenvolvido pelo *National Institute of Standards and Technology* (NIST)[14] e publicado como um padrão de processamento de informações federais

[14] https://www.nist.gov/

dos Estados Unidos da América (FIPS 180) em 1993. As funções de *Hash* mais comuns são SHA-1 e SHA-2, onde função SHA-1 produz um valor de *Hash* de 160 bits (IETF, 2011). Por outro lado, o SHA-2 é composto pelas funções SHA-224, SHA-256, SHA-384 e SHA-512, onde os números representam o tamanho das saídas para cada um dos algoritmos, em *bits*. Essas novas versões tem a mesma estrutura básica e usam os mesmos tipos de aritmética modular e operações binárias lógicas do SHA-1(IETF, 2011). Na Figura 12, é possível visualizar uma comparação entre os tipos de *Hash* da família SHA.

Figura 12: Comparação de parâmetros do SHA.

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Tamanho do resumo da mensagem	160	224	256	384	512
Tamanho da mensagem	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Tamanho do bloco	512	512	512	1024	1024
Tamanho da palavra	32	32	32	64	64
Número de etapas	80	64	64	80	80

Nota: todos os tamanhos são medidos em bits.

Fonte: Adaptado de Stallings (2014).

Existe outra função de *Hash* criptográfica, conhecida como SHA-3, que foi a vencedora de uma competição criada pelo NIST em 2007, para escolher a próxima geração de *Hash*. O projeto vencedor para o SHA-3 foi anunciado pelo NIST em outubro de 2012. O SHA-3 é complementar ao SHA-2 como padrão aprovado para uma grande gama de aplicações (STALLINGS, 2014).

A estrutura básica do SHA-3, é um esquema denominado de construção em esponja. A construção em esponja tem a mesma estrutura geral de outras funções de *Hash*. A função de esponja recebe uma mensagem de entrada e a divide em blocos de tamanho fixo. Cada bloco é processado por sua vez com a saída de cada iteração alimentada na próxima, produzindo finalmente um bloco de saída (STALLINGS, 2014).

2.3.1. Técnicas Para Decifrar *Hash*

Para Kuppens (2012) as duas principais maneira de decifrar um valor de *Hash* são:

Ataque de força bruta

A recuperação por força-bruta é um método de busca exaustiva em que todas as combinações possíveis de determinado grupo de caracteres são testadas. Teoricamente esse método pode encontrar qualquer senha. Porém, por ser muito ineficiente, só funciona de forma satisfatória para senhas curtas. Para uma senha com 5 caracteres, a primeira tentativa desse método será “aaaaa”, depois “aaaab” até finalizar em “zzzzz” (KUPPENS, 2012), nesse exemplo considerando apenas letras minúsculas.

Ataque de dicionário

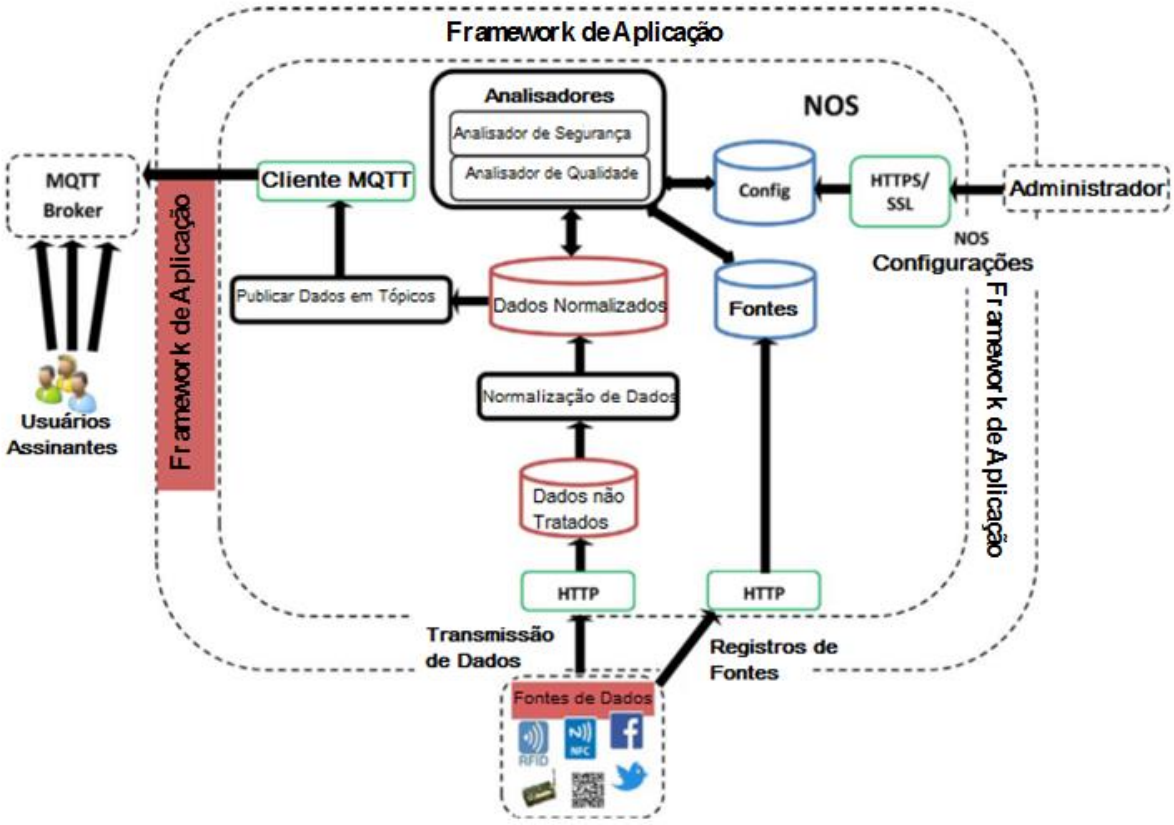
A recuperação utilizando palavras-chave ou dicionário é bastante eficaz. Nesse método, todas as ocorrências presentes em um determinado conjunto, são testadas como possíveis senhas. Esse ataque baseia-se no fato de muitas pessoas escolherem uma palavra simples de seu cotidiano como senha (KUPPENS, 2012).

2.4. ARQUITETURA DE *SOFTWARE*

Uma Arquitetura de *Software* (AS) é uma abstração de alto nível de um sistema de *Software*, que tem por objetivo, fornecer uma descrição de como o mesmo irá funcionar, dividindo-o em módulos e descrevendo como esses módulos estão conectados entre si, e como é feita a comunicação entre os módulos, bem como as tecnologias que são usadas no sistema.

AS funcionam como uma forma de mostrar aos envolvidos no processo de desenvolvimento do *Software* (Desenvolvedores, *Stakeholders* e os clientes) como o sistema irá funcionar, abstraindo a parte técnica para uma forma ilustrada. Na Figura 13, é mostrado um exemplo da arquitetura de um sistema seguro de Publicação / Assinatura.

Figura 13: Um exemplo de Arquitetura.



Fonte: Adaptado de Rizzardi et al. (2016).

Na Arquitetura apresentada na Figura 13, para cada dado de entrada, tanto a partir de fontes registrados como não registrados, as seguintes informações são extraída: (i) o tipo de fonte de dados (por exemplo, nó sensor, atuador, RFID, NFC, redes sociais); (ii) o modo de comunicação (por exemplo, comunicação discreta ou de fluxo contínuo); (iii) o esquema de dados, que descreve como o conteúdo dos dados é estruturado e o formato dos dados recebidos; (iv) os próprios dados; (v) o timestamp no qual os dados chegaram ao NOS.

Uma vez que os dados recebidos são de tipos e formatos diferentes, o NOS inicialmente os armazena na unidade de armazenamento “Dados não Tratados”. Isso inclui dois passos (Normalização de Dados e Analisadores), onde os resultados são colocados em uma representação de dados uniforme. Primeiro, a mensagem armazenada em “Dados não Tratados”, é colocada no formato especificado pelo módulo de “Normalização de Dados”, que os armazena em outra unidade de armazenamento denominada “Dados Normalizados”. Em seguida, um segundo módulo, constituído por um conjunto de “Analisadores”, periodicamente extrai os dados normalizados da unidade de armazenamento “Dados Normalizados”.

As regras, usadas para a avaliação das pontuações de segurança e qualidade, são armazenadas em Config. Que contém todos os parâmetros de configuração necessários para a administração do sistema *IoT*. Os Analisadores consultam a unidade de armazenamento Config. para saber quais ações devem ser executadas nos dados. Os dados tratados são utilizados para prestar serviços aos utilizadores interessados (ou aplicações externas) por meio de um mecanismo de *Publish/Subscribe*.

Para Fowler et al. (2006), "Arquitetura" é um termo que muitas pessoas, com pouca concordância entre si, tentam definir. Para os autores existem dois elementos comuns: um é a decomposição em alto nível de um sistema em suas partes, e o outro são as decisões difíceis de alterar. Eles complementam discorrendo que se percebe também, que cada vez mais, não há apenas um único modo de especificar a arquitetura de um sistema, existem diversas arquiteturas em um sistema, e a visão do que é significativo em termos de arquitetura pode mudar durante o ciclo de vida de um sistema.

AS lidam com abstração, decomposição e composição, estilo e estética. As AS também lidam com a concepção e implementação de estrutura de alto nível de *Software* (KRUCHTEN, 1995a).

Isto é o resultado de montar certo número de elementos de arquitetura em algumas formas bem escolhidas para satisfazer os grandes requisitos de funcionalidade e de desempenho do sistema, bem como, alguns outros requisitos não funcionais, tais como confiabilidade, escalabilidade, portabilidade e disponibilidade (KRUCHTEN, 1995b).

Segundo (KRUCHTEN, 1995b), as Arquiteturas de Software lidam com abstração, com decomposição e composição, com estilo e estética. Para descrever uma arquitetura de software, usa-se um modelo composto de vários pontos de vista ou perspectivas.

2.4.1. Padrões Arquiteturais

Os Padrões Arquiteturais (PAs) são modelos para o desenvolvimento de AS baseados em problemas comuns em determinadas áreas, ou seja, são soluções já comprovadas que podem ser usadas para resolver problemas semelhantes, assim agilizando o processo de resolução desses problemas.

PAs fornecem conhecimento de *design* reutilizável na forma de soluções comprovadas para problemas de software recorrentes em um contexto ou de domínio particular (ZDUN; KIRCHER; VOLTER, 2004). Os PAs representam a organização dos sistemas de software.

Eles definem, entre outros aspectos, os tipos de componentes e conectores utilizados em descrições arquiteturais.

Segundo Azevedo (2014), “Padrões Arquiteturais especificam componentes, responsabilidades e relacionamentos entre estes componentes que possam servir como solução para um problema recorrente em contextos específicos de software”.

Em Buschmann et al. (1996), os PAs são agrupados de acordo com as características dos sistemas em que eles são mais aplicáveis, com uma categoria tratando das questões gerais de estruturação. No Quadro 1 são listadas as categorias e os padrões nelas contidos, que serão descritos nas próximas subseções.

Quadro 1: Padrões Arquiteturais.

Categoria	**Padrão**
Estrutural	*Blackboard* *Layers* *Pipes and Filters*
Sistemas Distribuídos	*Brokers*
Sistemas Interativos	MVC PAC
Sistemas Adaptáveis	*Microkernel* *Reflection*

Fonte: Adaptado de Buschmann et al. (1996)**.**

2.4.1.1. Estrutural

Nesta categoria, antes de iniciar o projeto de um novo sistema, são coletados os requisitos do cliente e transformados em especificações, para então definir a arquitetura do sistema. Nesta fase, isto significa encontrar uma subdivisão de alto nível do sistema. Isso é geralmente o que se tem no começo, e deve ser transformado em uma estrutura mais organizada (BUSCHMANN et al., 1996).

2.4.1.1.1. Blackboard

O PA *Blackboard* é útil em problemas para os quais não são conhecidas estratégias determinísticas de solução. Em *Blackboard*, vários subsistemas especiais que montam seus

conhecimentos para construir uma solução possivelmente parcial ou aproximada (BUSCHMANN et al., 1996).

De acordo com Buschmann et al. (1996), o PA *Blackboard* vem da comunidade de Inteligência Artificial (IA). Segundo o autor, a ideia por trás dele merece ser vista em um contexto mais amplo. O autor complementa falando que "Em domínios pobremente estruturados ou simplesmente novos e imaturos, muitas vezes temos apenas conhecimentos incompletos sobre como lidar com problemas específicos".

O PA *Blackboard* mostra um método de combinar esse conhecimento desigual para chegar a soluções, mesmo que elas não sejam garantidas. Quando o domínio do aplicativo amadurece com o tempo, os *designers* frequentemente abandonam a arquitetura *Blackboard* e desenvolvem arquiteturas que suportam abordagens de soluções fechadas, nas quais as etapas de processamento são predefinidas pela estrutura do aplicativo (BUSCHMANN et al., 1996).

Uma arquitetura *Blackboard* ajuda na construção de sistemas *software* que devem resolver tarefas com base em incertezas, hipotéticas, ou conhecimento e dados incompletos. Também ajuda a descobrir e aperfeiçoar soluções fortemente determinísticas para tarefas que não tem tais soluções. Por outro lado, não há garantia que um sistema baseado em *Blackboard* produza um resultado útil (BUSCHMANN; HENNEY; SCHMIDT, 2007).

2.4.1.1.2. *Layers*

O PA *Layers* ajuda a estruturar aplicações que podem ser decompostas em grupos de subtarefas em que cada grupo está em um nível parcial de abstração (BUSCHMANN et al., 1996).

Para Buschmann et al. (1996), uma abordagem em camadas é considerada uma prática melhor do que implementar o protocolo como um bloco monolítico, uma vez que implementar conceitualmente diferentes questões separadas tem vários benefícios, por exemplo, ajudando o desenvolvimento por equipes e apoiando codificação incremental e testes. O uso de peças semi independentes, também facilita a troca de peças individuais em uma data posterior. Melhores tecnologias de implementação, como novas linguagens ou algoritmos, podem ser incorporadas simplesmente reescrevendo uma seção delimitada de código.

2.4.1.1.3. Pipes and Filters

O PA *Pipes and Filters,* fornece uma estrutura para sistemas que processam um fluxo de dados. Cada passo de processamento é encapsulado em um componente de filtro. Os dados, são passados através de tubos entre filtros adjacentes. A recombinação de filtros permite que sejam construídas famílias de sistemas relacionados (BUSCHMANN et al., 1996). O processamento de imagem é um excelente exemplo de um domínio que pode ser melhor modelado em software por meio de uma arquitetura *Pipes and Filters* (BUSCHMANN; HENNEY; SCHMIDT, 2007). Esse PA é atraente em áreas onde os fluxos de dados podem ser processados de forma incremental (BUSCHMANN et al., 1996).

2.4.1.2. Sistemas Distribuídos

Esta categoria é caractearizada por sistemas de computadores com várias CPUs, separadas remotamente ou não. Na maioria dos casos, uma máquina em uma rede ou sistema multiprocessador pode travar sem afetar o resto do sistema (BUSCHMANN et al., 1996).

2.4.1.2.1. *Broker*

O PA *Broker* pode ser usado para estruturar sistemas de software com componentes distribuídos que interagem com chamadas de serviços remotos. Um componente de corretagem (um *Broker*) é responsável por coordenar a comunicação, como por exemplo, encaminhar pedidos, bem como, para transmissão de resultados e exceções. Esse tipo de sistema funciona com três componentes, o Publicador, o *Broker* e Assinante.

Segundo Buschmann et al. (1996), o PA *Broker*, para aplicações distribuídas, pode acessar serviços de outras aplicações simplesmente pelo envio de mensagens a objetos mediadores, sem se preocupar com questões específicas relacionadas à comunicação entre processos, como a sua localização.

Segundo Levy e Losavio (1999), o PA *Broker* é usado para, estruturar sistemas de S*oftware* distribuídos com componentes dissociados que interagem por invocações de serviços remotos. Ele é responsável por coordenar as comunicações.

Um componente *Broker* pode ser introduzido para gerenciar a comunicação entre clientes e servidores, assim o padrão cliente-servidor muda para o padrão *Broker*. Neste caso, enquanto o padrão cliente-servidor ainda existe, é englobado pelo padrão *Broker*. (HARRISON; GUBLER; SKINNER, 2016).

2.4.1.3. Sistemas Interativos

A categoria Sistemas Interativos é caracterizada pelo alto grau de interação do usuário, alcançado principalmente com a ajuda de *interfaces* gráficas do usuário. O objetivo é melhorar a usabilidade de uma aplicação. Os sistemas de *Software* utilizáveis proporcionam acesso conveniente aos seus serviços e, portanto, permitem que os usuários aprendam a aplicação e produzam resultados rapidamente (BUSCHMANN et al., 1996).

2.4.1.3.1. MVC

O PA *Model-View-Controller* (MVC), divide um aplicativo em três componentes. O modelo contém a funcionalidade principal e os dados. As visões exibem informações ao usuário. Controladores lidam com entradas do usuário. As visões e os controladores em conjunto constituem a *interface* do utilizador. Um mecanismo de propagação de mudança garante consistência entre a *interface* do usuário e o modelo (BUSCHMANN et al., 1996).

MVC fornece provavelmente a organização de arquitetura mais conhecida para sistemas de *Software* interativos. Ele está subjacente a muitos sistemas interativos e *frameworks* de aplicação para sistemas de *Software* com *interfaces* gráficas de usuário (BUSCHMANN; HENNEY; SCHMIDT, 2007).

2.4.1.3.2. PAC

O PA *Presentation-Abstraction-Control* - apresentação-abstração-controle (PAC), enfatiza os níveis de abstração crescente. No entanto, a estrutura PAC global é uma árvore de nós PAC em vez de uma linha vertical de nós em camadas em cima uns dos outros (BUSCHMANN et al., 1996).

O Padrão Arquitetural PAC, define uma estrutura para sistemas de *Software* interativo na forma de uma hierarquia de agentes cooperantes. Cada agente é responsável por um aspecto específico da funcionalidade do aplicativo e consiste de três componentes: apresentação, abstração e controle. Esta subdivisão separa os aspectos de interação homem-computador do agente de seu núcleo funcional e sua comunicação com outros agentes (BUSCHMANN et al., 1996).

2.4.1.4. Sistemas Adaptáveis

A categoria Sistemas Adaptáveis é caracterizada por sistemas que evoluem ao longo do tempo, novas funcionalidades são adicionadas e os serviços existentes são alterados. Eles devem suportar novas versões de sistemas operacionais, plataformas de *interface* de usuário ou componentes de terceiros e bibliotecas. A adaptação a novos padrões ou plataformas de *hardware* também pode ser necessária (BUSCHMANN et al., 1996).

2.4.1.4.1. *Microkernel*

O PA *Microkernel,* propõe a separação de um conjunto de funcionalidades mínimas das funcionalidades estendidas e partes específicas de clientes. O encapsulamento dos serviços fundamentais da aplicação é realizado no componente "*Microkernel*". As funcionalidades estendidas e específicas devem ser distribuídas entre os componentes restantes da arquitetura (BUSCHMANN et al., 1996).

O PA *Microkernel* aplica-se a sistemas de *Software* que são capazes de se adaptar às mudanças nos requisitos do sistema. Ele separa um núcleo funcional mínimo da funcionalidade estendida e peças específicas do cliente. O *Microkernel* também serve como um soquete para conectar essas extensões e coordenar suas colaborações. Os sistemas *Microkernel* são empregados em arquiteturas Cliente-Servidor executados em cima do componente de *Microkernel* (BUSCHMANN et al., 1996).

2.4.1.4.2. *Reflection*

O PA *Reflection,* fornece um mecanismo para alterar dinamicamente a estrutura e o comportamento dos sistemas de *Software*. Suporta a modificação de aspectos fundamentais, tais como, estruturas de tipo e mecanismos de chamada de função. Neste padrão, uma aplicação é dividida em duas partes. Um nível básico inclui a lógica do aplicativo principal (BUSCHMANN; HENNEY; SCHMIDT, 2007).

O PA *Reflection,* define uma arquitetura que objetiva aspectos específicos da estrutura de um sistema e comportamento, que suporta flexibilidade em termos de como sua funcionalidade executa e/ou pode ser usado por seus clientes. Por isso, é frequentemente usado no contexto de cenários de integração de aplicativos e serviços, nos quais aplicativos cliente devem ser capazes de usar ou controlar as funcionalidades de outras aplicações sem ter

um conhecimento explícito e integrado de suas *interfaces* e comportamento interno (BUSCHMANN; HENNEY; SCHMIDT, 2007).

Para realizar uma arquitetura *Reflection*, primeiro especifique-se um *design* estável para a aplicação que não considera flexibilidade em tudo. A estabilidade é a chave para a flexibilidade. Normalmente, cada responsabilidade autocontida do aplicativo é encapsulada dentro de um objeto de domínio, que juntos formam o nível base da arquitetura *Reflection* (BUSCHMANN; HENNEY; SCHMIDT, 2007).

2.5. TRABALHOS RELACIONADOS

O processo de revisão da literatura é muito importante para uma pesquisa científica. Pois a partir dela é possível chegar a uma visão do que está sendo pesquisado e desenvolvido na atualidade. Esta seção tem por objetivo apresentar o estado da arte de soluções de segurança em *IoT* para isso foram extraídos 6 trabalhos de uma revisão da literatura descrita a seguir.

2.5.1. Revisão da Literatura

Inicialmente foi executada uma busca *ad hoc* para fazer um levantamento dos principais problemas e desafios a serem pesquisados/solucionados na área de *IoT*. Durante esta pesquisa inicial, foi possível constatar que dois dos grandes problemas da área são segurança e consumo energético. Diante disto, identificou-se a necessidade de desenvolver um estudo mais aprofundado nesses dois problemas da área.

De acordo com Kitchenham et al. (2009) uma Systematic Literature Review (Revisão Sistemática da Literatura) (SLR), tem como objetivo fornecer indícios para o desenvolvimento de pesquisas baseadas em evidências, a partir da seleção e síntese das pesquisas mais relevantes. Mas, como nesta revisão não foram seguidos todos os procedimentos propostos por Kitchenham et al. (2009), não pode ser considerada uma SLR e sim uma Revisão da Literatura. Ou seja, neste trabalho foi elaborada uma Revisão da Literatura seguindo alguns dos princípios de uma SLR. Foram feitos dois tipos de busca, uma busca manual e outra automática, a partir dos trabalhos retornados nas buscas foi feita uma filtragem, que tinha por objetivo chegar aos trabalhos que mais se aproximavam desta pesquisa.

2.5.2. Busca Automática

Para a busca automática, foram usadas as seguintes bases de artigos científicos (ACM Digital *Library, IEEExplore, Science Direct, Springer Link e Scopus*), a busca limitou-se a "*internet das coisas*" e o uso do protocolo MQTT com parâmetros de segurança na transmissão de dados e consumo energético. Outro critério usado foi para que a busca se limitasse a artigos mais recentes, a partir de 2012 até o ano 2016, pois como essa área é relativamente nova, a literatura é escassa. Na leitura *ad hoc* não foi encontrado nenhum trabalho que se abordasse uma solução de segurança especificamente para *IoT,* que se aproximasse desta pesquisa e que fosse anterior ao ano de 2012. As buscas foram executadas no dia 14 de julho de 2016, em todas as bases mencionadas. A *String* de busca, foi elaborada de acordo com os trabalhos *ad hoc* realizados, anteriormente. Como constatado na busca *ad hoc,* dois grandes problemas da área de *IoT* são segurança e consumo energético, então foi usada a seguinte *String* para a busca automática:

("Internet of things" OR "IoT") AND ("MQTT") AND ("security" OR "safety" OR "energy reduction" OR "reducing energy consumption")

É importante salientar que a *String* foi adaptada para cada base devido as diferentes sintaxes, mas todas as combinações estavam presentes. Uma observação importante é que as alterações foram mínimas devido à simplicidade da *String* elaborada para esta pesquisa. Após a busca automática, foram obtidos os seguintes resultados (foram eliminados os artigos repetidos, o critério pra eliminação foi, se um artigo já tinha sido encontrado nas bases buscadas anteriormente, então ele é descartado):

Tabela 1: Artigos encontrados na busca automática.

BASES PESQUISADAS	QTD. ARTIGOS RETORNADOS
ACM	2
IEEE	18
Science Direct	32
Scopus	14
SpringerLink	32
TOTAL	98

Fonte: Elaborado pelo autor.

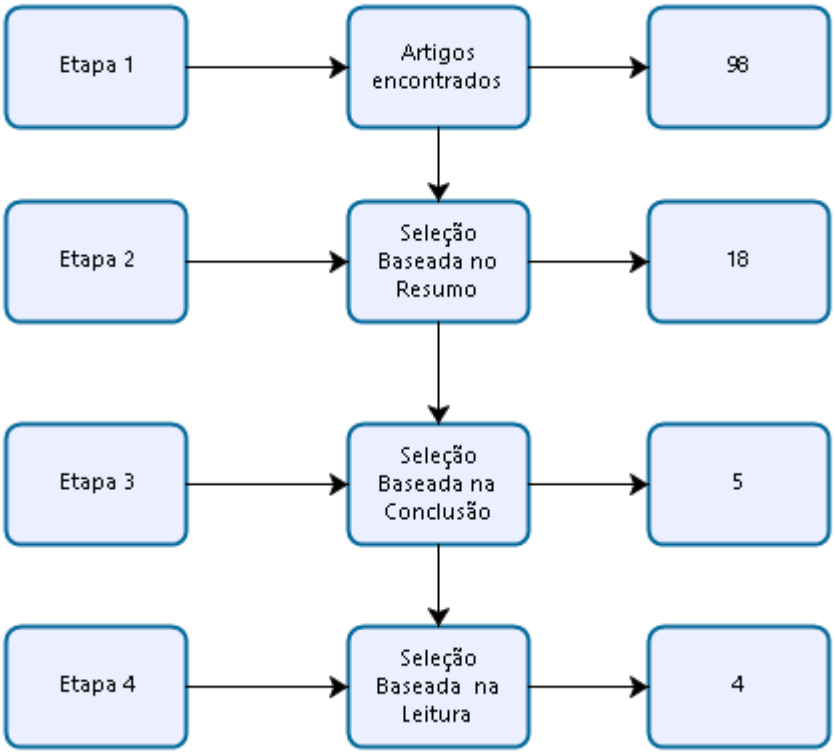
Nesta busca foram considerados apenas trabalhos escritos no idioma inglês. Após a busca em todas as bases de artigos, ao todo foram retornados 98 artigos, foi efetuado o

download de todos retornados pela busca. O processo de análise dos artigos encontrados na busca automática foi dividido em quatro etapas, mostrados a seguir na Figura 14:

Para a filtragem dos trabalhos foram usados os seguintes critérios de inclusão:

a) Soluções de segurança em *IoT*;
b) Uso do protocolo MQTT com soluções de segurança em *IoT*;
c) Redução do consumo energético de sensores em *IoT*.

Figura 14: Processo de seleção dos trabalhos relacionados na busca automática.



Fonte: Elaborado pelo autor.

Este processo teve como objetivo, selecionar os principais trabalhos que propõem soluções de segurança e redução de consumo energético em *IoT*. A primeira etapa teve por objetivo fazer a busca nas bases mencionadas acima. Na segunda etapa, foi realizada a leitura do título e do resumo de todos os artigos retornados na busca, a partir desta leitura foram selecionados 18 artigos, considerados relevantes para esta pesquisa seguindo os critérios estabelecidos.

Na Terceira etapa, foram lidas as conclusões e trabalhos futuros dos 18 artigos selecionados na etapa anterior, a partir desta leitura restaram 5 artigos que atendiam aos critérios para a filtragem. Na quarta e ultima etapa, foram lidos na íntegra os 5 artigos

selecionados na etapa anterior. Após a leitura dos artigos, então 4 artigos foram selecionados, ambos os artigos estão focados em soluções de segurança para *IoT*. Estes trabalhos, são descritos na seção de trabalhos relacionados (2.5.4).

2.5.3. Busca Manual

Na busca manual, foram utilizadas várias bases, inclusive foram feitas buscas no *Google* Acadêmico[15]. Para esta busca também foram considerados eventos brasileiros como o Congresso da Sociedade Brasileira de Computação (CSBC 2015 e 2016), também foram encontrados outros artigos internacionais considerados relevantes para a pesquisa e que não tinha retornado na busca automática.

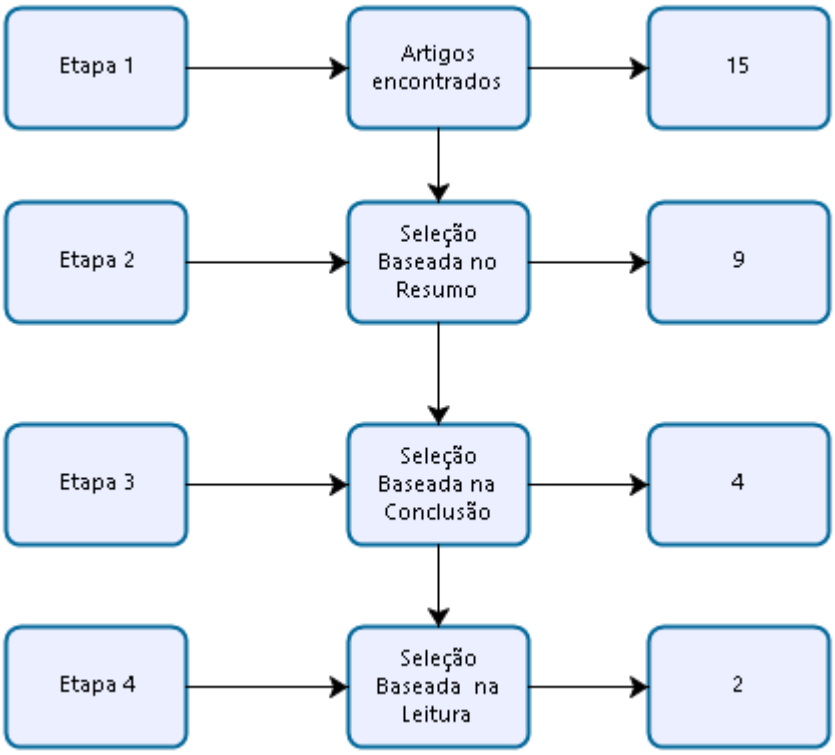
Essa pesquisa foi limitada a trabalhos entre 2012 a 2016. Para esta busca foi pesquisado apenas abordagens baseadas em segurança em *IoT*, descartando consumo energético, pois os artigos selecionados na busca automática são todos sobre segurança, então optou-se por seguir essa linha na busca manual. As principais palavras-chaves utilizadas para esta busca foram: Internet of Things, *IoT*, MQTT AND Security, basicamente seguindo esta String:

("Internet of things" OR "IoT") AND ("MQTT") AND ("security" OR "safety")

A partir dos resultados desta busca, alguns trabalhos foram selecionados de acordo com o título. A partir destas fontes, as principais referências destes trabalhos também foram analisadas, repetindo a abordagem até não restar nenhum trabalho potencialmente relevante para o escopo da pesquisa. Esta pesquisa resultou em 15 artigos pré-selecionados. O método de analise dos artigos selecionados na busca manual foram às mesmas etapas e critérios já mencionados na seção 2.5.2, dos 15 que foram analisados, apenas 2 foram selecionados. A Figura 15, demostra essas informações.

[15] https://scholar.google.com.br/

Figura 15: Processo de seleção dos trabalhos relacionados na busca manual.



Fonte: Elaborado pelo autor.

Após a análise dos trabalhos selecionados, esta pesquisa focou em soluções de segurança para *IoT*, até porque os trabalhos resultantes eram a respeito de segurança (uso de soluções de segurança), talvez devido a esse ser considerado um dos grandes problemas da área de *IoT*.

2.5.4. Trabalhos Selecionados

Nesta seção são descritas as principais características dos trabalhos selecionados da Revisão da Literatura.

2.5.4.1. An Efficient Device Authentication Protocol Without Certification Authority for Internet of Things

Em Jang et al. (2016), é proposto um protocolo de autenticação eficiente sem Autoridade de Certificação (*Certificate Authority* - CA) para a Internet das Coisas. Em comparação com os protocolos de aplicação restrita existentes, o protocolo proposto aumenta a eficiência,

minimizando o número de trocas de mensagens. O protocolo é baseado em um algoritmo de *Hash* com chave, em que uma CA não é necessária.

O trabalho de Jang et al. (2016) sugere um protocolo de autenticação que permite a autenticação entre dispositivos na ausência de um servidor de controle central baseado em um *Hash* com chave. A autenticação de *Hash-Tree* comumente usada foi analisada de vários ângulos, tem vantagens de ser capaz de autenticar com menos recursos, menos aperto de mão (*hand-shakes*) e vazamentos de informação reduzidos em autenticações esporádicas. Esse protocolo tem a vantagem de ser capaz de se destacar uma autenticação. Este é um processo complexo em que o desempenho e o consumo de energia são mais eficientes do que o método convencional em termos de número de peças e resolve os problemas de segurança que ocorrem quando uma quantidade significativa do peso é atribuída ao papel da CA.

Neste trabalho, foi proposto um novo protocolo, porem algumas informações a respeito do desempenho foram omitidas, testes de desempenho não foram realizados, testes usando redes muito lentas, comparando com outros protocolos que podem ser usados em redes com baixa qualidade, testes para medir o seu comportamento com um grande número de requisições, o que é um senário muito comum em *IoT*. Só foram empregados testes de consumo de energia, *delay* da autenticação e tamanho do código usado para ele funcionar, esses foram os únicos critérios usados nos testes, o que deixa muitas dúvidas a respeito da sua eficácia.

2.5.4.2. AUPS: An Open Source AUthenticated Publish/Subscribe System for the Internet of Things

Em Rizzardi et al. (2016), é apresentado um sistema seguro de Publicação/Assinatura que estende o MQTT por meio de uma estrutura de gerenciamento de chaves e uma aplicação de políticas. Desta forma, o fluxo de informação em sistemas da *IoT* com alimentação MQTT pode ser controlado de forma flexível por meio de políticas flexíveis.

Segundo Rizzardi et al. (2016), o protocolo MQTT está surgindo como um padrão de protocolos de mensagens, padrão para aplicativos *IoT*, mas na sua versão atual oferece pouco suporte de segurança. Neste trabalho é apresentado o AUPS, um sistema e métodos para adicionar segurança aos sistemas *IoT* baseados em MQTT. A AUPS inclui uma estrutura de aplicação de políticas, aliada a uma gestão-chave, e é capaz de gerenciar com eficácia as publicações e assinaturas através de interações MQTT. A solução proposta foi implementada e liberada, sob licença de código aberto, para a comunidade de pesquisa em geral.

Este trabalho apresenta algumas limitações, por exemplo, usa muito poder computacional, o que é um fator limitante para dispositivos *IoT*. Na configuração original, isto é, sem AUPS, a latência média medida é de aproximadamente 3,5 ms e 4 ms para 10 e 20 pacotes por segundo, respectivamente. Uma vez habilitada a extensão segura, esse tempo médio aumenta em 2 ms ou seja 57% e 50% respectivamente. O que surge é que as novas operações adicionam um aumento do atraso. Outro ponto que limita a solução é que segundo os autores os testes foram muito limitados.

2.5.4.3. Secure MQTT for Internet of Things (IoT)

Singh et al. (2015), propõe uma versão segura dos protocolos MQTT e MQTT-SN (SMQTT e SMQTT-SN) em que o recurso de segurança é aumentado para o protocolo MQTT existente com base na chave / *Ciphertext Policy-Attribute Based Encryption* (KP / CP- ABE), usando criptografia de curva elíptica leve. Além disso, esse trabalho demostra a viabilidade dos protocolos SMQTT e SMQTT-SN para vários requisitos da *IoT* através de simulações e avaliação do seu desempenho.

Um MQTT Seguro (SMQTT), que aumenta recurso de segurança para o protocolo MQTT existente e suas variantes com base na Criptografia Baseada em Atributo (*Attribute-Based Encryption* - ABE), sobre curvas elípticas. A vantagem de usar ABE é devido ao seu design inerente que suporta criptografia de transmissão (com uma criptografia, a mensagem é entregue a vários usuários pretendidos) e, portanto, adequado para aplicações da Internet das Coisas. ABE são de dois tipos: (i) baseado em políticas texto cifrado ABE (CP-ABE), e (ii) ABE baseado em políticas Key (KP- ABE) (SINGH et al., 2015).

Essa abordagem funciona da seguinte maneira, o *Publisher* envia mensagem criptografada para o *Broker* do *cluster* ao qual pertence. Em seguida, *Broker* desse *cluster* encaminha mensagem criptografada para todos os assinantes no mesmo *cluster*, bem como *Broker* de outros *clusters*. Todos os assinantes, descriptografam mensagem usando as chaves fornecidas pelo PKG Global (*Broker*) da nuvem, o que torna esse processo complexo, onde o assinante recebe uma chave para decifrar os dados em que está inscrito.

Essa proposta deixa alguns questionamentos, pois os testes foram realizados em computadores, ou seja, não tiveram uma preocupação com o consumo de memória, foi usado para teste uma máquina com 2 GB de memória. Foram realizados e repetidos os experimentos com três laptops conectados em rede padrão 802.3 e simulados como dispositivos *IoT* para

três iterações. As informações a respeito do consumo de memória e energético foram omitidas.

2.5.4.4. A Simple Security Architecture for Smart Water Management System

Nesta pesquisa é proposta uma arquitetura simples para sistemas de gestão de água em tempo quase real, usando MQTT. O sistema funciona da seguinte maneira: São introduzidos autorização e controle de acesso a servidores, do inglês *Authorization and Access Control* (AAC) na arquitetura. O papel da AAC é para autenticar dispositivos e proporcionar-lhes *tokens* de acesso. Esses *tokens* de acesso podem ser usados mais tarde para autenticação sem estado. A arquitetura conta com funções de *Hash* criptográficas e *tokens* de acesso (NTULI; ABU-MAHFOUZ, 2016).

Antes de o produtor publicar conteúdo, uma função de *Hash* criptográfica é utilizada para produzir um valor de *Hash*. Este valor *Hash* é enviado juntamente com o conteúdo original e o *token* de acesso. A arquitetura considera três metas de segurança: inicialização segura, comunicação segura e atualizações de *firmware* segura(NTULI; ABU-MAHFOUZ, 2016).

No entanto, essa pesquisa está limitada a uma contribuição teórica. O que de certa maneira a torna incipiente no que diz respeito à avaliação, porque não foi desenvolvido nenhum protótipo do sistema usando a arquitetura proposta para avaliar a sua eficácia. O desenvolvimento de um protótipo ficou para trabalhos futuros. Outro ponto fraco, é possível também que um invasor da rede capture a mensagem e o *Hash* e altere os dois, desta maneira o receptor não conseguirá identificar que ouve alteração nos dados recebidos.

2.5.4.5. Authorization Mechanism for MQTT-Based Internet of Things

A pesquisa de Niruntasukrat; Issariyapat e Pongpaibool (2016), desenvolveu um mecanismo de autorização para *IoT* baseado em MQTT. O projeto é baseado em *OAuth 1.0a*, que é um padrão de autorização aberto para aplicações *web*. Alguns redesenhos e alterações foram feitos para ajustá-lo dentro do ambiente MQTT. Várias considerações são levadas em conta, incluindo recursos limitados de nó, a falta de *interface* de usuário do nó e chave de distribuição secreta e gestão.

Durante a autorização, um conjunto de credenciais (*token* de acesso e *token* de acesso secreto) que passam através de um canal não criptografado entre o dispositivo e o ponto de

extremidade de autorização podem ser roubadas. No entanto o mecanismo de autorização proposto requer dois conjuntos de credenciais para o dispositivo acessar o *Broker* MQTT. Outro conjunto de credenciais (ID do dispositivo e dispositivo secreto), têm que ser enviados para outros usuários, através de seu dispositivo, como um telefone ou um computador que não tenha limitação de recursos e suporte ao protocolo HTTPS, mais tarde é incorporado *off-line* na memória do dispositivo. Enquanto, o dispositivo *IoT* é mantido a salvo de qualquer tipo de infiltração física após o seu ID ser incorporado, o método proposto fornece a melhor segurança possível no ambiente restrito (NIRUNTASUKRAT; ISSARIYAPAT; PONGPAIBOOL, 2016). O projeto foi implementado em uma plataforma de serviços *IoT*, com base em MQTT e demonstrou que ele funciona como pretendido, atraso de autorização e mensagens de sobrecarga são mínimas.

Segundo Niruntasukrat; Issariyapat e Pongpaibool (2016), existe uma complexidade no processo de autenticação, os mesmos acreditam que a função gargalo é HMAC-SHA1 para a criação de assinatura, o HMAC requer dispositivos com recursos pesados (JANG et al., 2016). O pedido de *token*, tem um atraso maior do que o atraso de acesso ao *token*, porque, o AuthServer tem que validar muitos mais campos antes de emitir o pedido de *token* com êxito.

2.5.4.6. Securing Smart Maintenance Services: Hardware-Security and TLS for MQTT

Neste projeto é construído um mecanismo, que permite a um dispositivo enviar informações de manutenção de um carro via MQTT para uma central, garantindo a segurança no envio dessas informações. O sistema monitora o carro e seu funcionamento para ver se o mesmo precisa de manutenção.

O projeto do sistema usa o *Transport Layer Security* (TLS), que adiciona uma camada de comunicação segura sob o protocolo MQTT. Além disso, o sistema usa um controlador de segurança de *hardware*, que realiza a autenticação do cliente TLS ao assinar digitalmente os respectivos desafios. Assim, a chave privada da APC (*AVL Particle Counter)* pode ser armazenada de forma segura dentro do controlador de segurança e nunca precisa deixar o armazenamento seguro (LESJAK et al., 2015).

A *Transport Layer Security* (TLS), que adiciona uma camada de comunicação segura sob o protocolo MQTT, é considerada pesada e tem uma quantidade considerável de complexidade computacional (NIRUNTASUKRAT; ISSARIYAPAT; PONGPAIBOOL,

2016). Para Rizzardi et al. (2016) e Jang et al. (2016) TLS nem sempre é uma escolha adequada para dispositivos com recursos limitados.

2.5.5. Análise Comparativa

No Quadro 2, é apresentada uma análise comparativa entre os trabalhos extraídos da revisão da literatura e a técnica composta neste trabalhos, para a analise são usados os 5 critérios apresentados a seguir. Por questão de estética do quadro 2, os trabalhos serão representados apenas pelo nome do primeiro autor e o ano.

Critério 1 - Alteração do protocolo (MQTT): Tem por objetivo analisar se o trabalho fez alguma alteração ou extensão do protocolo de aplicação usado para transmitir os dados dos dispositivos para o *Broker*. Este trabalho não faz nenhuma alteração na estrutura do protocolo MQTT, apenas o usa como protocolo para transmitir os dados gerados. Dos trabalhos relacionados apenas Ntuli; Abu-Mahfouz (2016) não fazem alteração ou extensão do MQTT, porem todos os demais trabalhos fizeram algum tipo de alteração no MQTT.

Critério 2 - Desenvolvimento de um protótipo: Tem por objetivo investigar se o trabalho fez algum protótipo de um sistema usando a abordagem proposta. Neste critério apenas o trabalho de Ntuli; Abu-Mahfouz (2016) não desenvolve algum tipo de protótipo usando sua abordagem.

Critério 3 - Avaliação do consumo de recursos de *Hardware*: Tem por objetivo analisar se o trabalho fez alguma avaliação para ver quanto de recursos do *Hardware* é consumido para utilizar a solução proposta no trabalho e se realmente consome poucos recursos do *Hardware*. Neste critério nenhum dos trabalhos extraídos da revisão da literatura relata que fez algum tipo de avaliação para verificar o quanto de recursos dos dispositivos sua abordagem consome.

Critério 4 - Confidencialidade dos dados: Tem por objetivo analisar se o trabalho fornece confidencialidade aos dados transmitidos. Neste critério apenas a abordagem proposta nesta pesquisa e os trabalhos de Lesjak et al. (2015) e Singh et al. (2015) fornecem confidencialidade aos dados transmitidos para o *Broker*.

Critério 5 - Avaliação (Segurança/confidencialidade): Tem por objetivo verificar se foi feita alguma avaliação para constatar se realmente é fornecida a segurnaça. Neste caso apenas os trabalhos de Ntuli; Abu-Mahfouz (2016) Singh et al. (2015) não fizeram avaliações para verificar se realmente eles forneciam confidencialidade aos dados transmitidos.

Quadro 2: Analise dos trabalhos relacionados.

Critérios	Trabalhos						
	Esta Pesquisa	Jang (2016)	Rizzardi (2016)	Singh (2015)	Ntuli (2016)	Niruntasukrat (2016)	Lesjak (2015)
Critério 1	Não	Sim	Sim	Sim	Não	Sim	Sim
Critério 2	Sim	Sim	Sim	Sim	Não	Sim	Sim
Critério 3	Sim	Não	Não	Não	Não	Não	Não
Critério 4	Sim	Não	Não	Sim	Não	Não	Sim
Critério 5	Sim	Sim	Não	Sim	Não	Sim	Sim

Fonte: Elaborado pelo autor.

2.6. CONSIDERAÇÕES FINAIS

Neste capítulo foram descritos e contextualizados os principais conceitos de Internet das Coisas, são descritas também as tecnologias usadas para o desenvolvimento desta dissertação, fornecendo uma visão geral das tecnologias que são usadas nesta área, são descritos quais os principais protocolos de aplicação usados, bem como também é detalhado como o MQTT funciona, já que o mesmo vem sendo bastante usado em *IoT*. Foi descrito também como funcionam as funções de *Hash* criptográficas, são descritas as funções da família SHA. Foram descritos os principais PAs encontrados na literatura.

Em seguida foi realizada uma revisão da literatura seguindo alguns princípios de uma Revisão Sistemática da Literatura sobre soluções de segurança em *IoT*. Em seguida, foi realizada uma consolidação das características mais relevantes de cada abordagem, descrevendo os trabalhos relacionados que foram selecionados, bem como as limitações de cada trabalho. Por fim foi feita uma análise comparativa entre os trabalhos extraídos na revisão da literatura e a abordagem desenvolvida neste trabalho.

3. SOLUÇÃO PROPOSTA

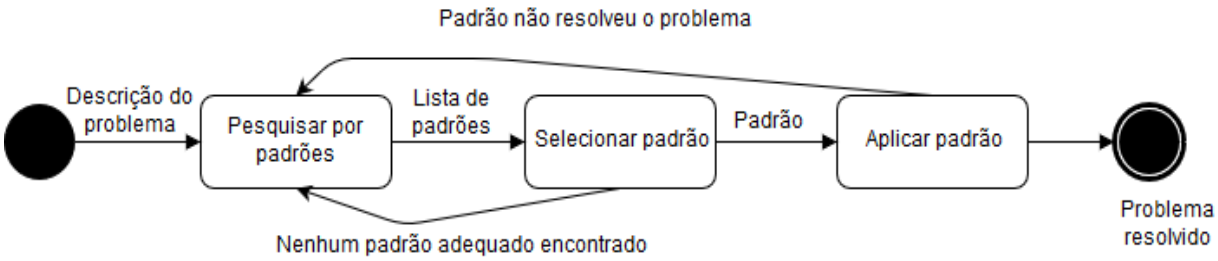
Neste capítulo, são descritos os principais componentes, restrições e as funcionalidades da técnica proposta com o uso da função de *Hash* criptográfica e do protocolo de comunicação usado para transmitir os dados dispositivos/sensores com recursos limitados.

3.1. PADRÃO ARQUITETURAL ADOTADO

Devido à flexibilidade requerida nesta arquitetura, que deve integrar dispositivos/sensores e *smartphones* à plataforma, ela deve ser modular e com baixo desacoplamento, permitindo reuso de código onde for possível, com intuito de reduzir retrabalho na adição de novas funcionalidades ou suporte de novos dispositivos pelo sistema.

Para a seleção do Padrão Arquitetural (PA) a ser adotado neste trabalho, foi seguido o processo de seleção de padrões apresentado por Birukou (2010). O processo é iniciado com a descrição do problema a ser solucionado e, em seguida, é feita uma investigação dos padrões existentes. O próximo passo é selecionar o padrão que melhor se adequa às características do problema a ser solucionado. Por fim, o padrão é aplicado para a solução do problema. A Figura 16, mostra o fluxo habitual da técnica de tomada de decisão de acordo com Birukou (2010).

Figura 16: O processo de aplicação de padrões para a resolução de problemas.



Fonte: Adaptado de Birukou (2010).

Após um levantamento na literatura dos principais PAs existentes e utilizados atualmente, concluiu-se que o PA *Broker* é o que se adequa mais aos objetivos e necessidades deste trabalho, pois ele foi idealizado para sistemas distribuídos, o que se assemelha com a arquitetura aqui proposta para sistemas de Internet das Coisas. A seguir, são descritas algumas das características que levaram a escolha deste padrão:

a) O padrão *Broker* desacopla o emissor e o receptor;
b) Possibilita integração de componentes heterogêneos;
c) O sensor não precisa fazer o processamento, quem faz é o *Broker;*

d) O protocolo adotado para esta pesquisa funciona usando *Broker*.

Este PA cria um intermediário entre um componente cliente e um componente servidor, esse intermediário é chamado de *Broker*. Um cliente envia uma mensagem para o *Broker* contendo todas as informações apropriadas para que a comunicação seja efetuada. O *Broker* é responsável por completar a conexão enviando a mensagem para o servidor.

O padrão *Broker* é adequado para aplicações em que os componentes trocam mensagens que exigem extensa transformação entre formatos de origem e de destino. O *Broker* desacopla o emissor e o receptor, permitindo-lhes produzir ou consumir o seu formato de mensagem nativa, e centraliza a definição da lógica de transformação no *Broker* para facilitar a compreensão e modificação (BUSCHMANN; HENNEY; SCHMIDT, 2007), o que se aplica perfeitamente aos objetivos desta pesquisa.

3.2. PROTOCOLO DE APLICAÇÃO ADOTADO

O protocolo de aplicação MQTT foi adotado para este trabalho devido ao fato de ser bastante utilizado em aplicações para *IoT*. Ele fornece mecanismos para garantia de entrega de mensagens em redes não confiáveis, além de ser adequado para dispositivos com pouco poder de processamento, que é o foco de aplicações para *IoT*. O MQTT deixa a responsabilidade de processar os dados para o *Broker* que está na nuvem, com isso é obtido uma redução no consumo de energia por parte do dispositivo.

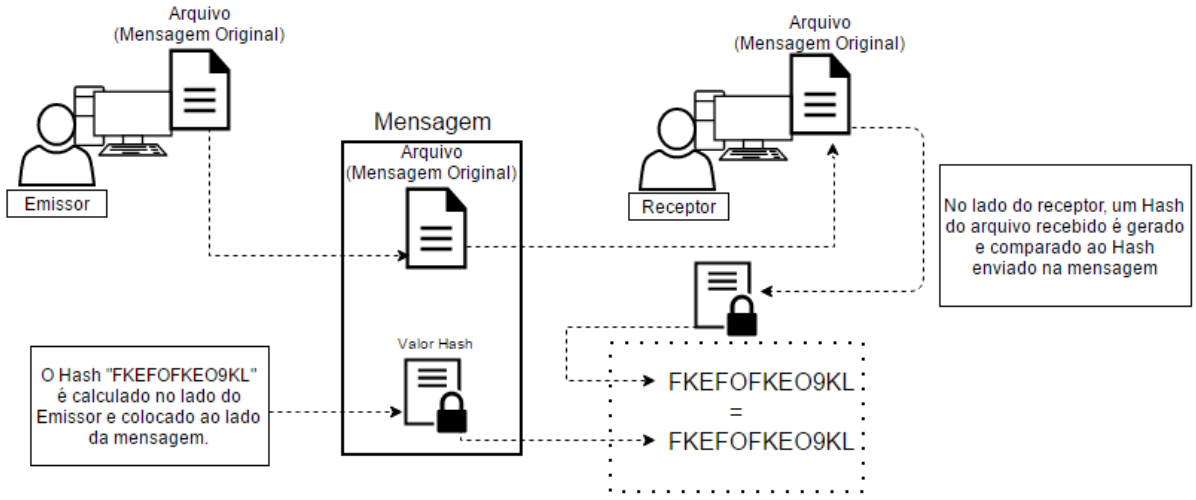
Devido à sua simplicidade e baixo *overhead*, o MQTT é adequado para ambientes de recursos limitados e tem aplicação em vários domínios, incluindo aplicativos de monitoramento e aplicativos com *feeds* (alimentação) ativa de dados em tempo real (RIZZARDI et al., 2016). Outro fator, que levou a escolha do MQTT para ser o protocolo de aplicação desta arquitetura está relacionado à sua maturidade, estabilidade e ao fato de que foi recentemente adotado pelo Consórcio OASIS como padrão oficial. Vários autores defendem seu uso, e até defendem que ele seja adotado como um padrão (JANG et al., 2016; RIZZARDI et al., 2016; SINGH et al., 2015).

Em particular, o MQTT foi implementado para conectar facilmente as "coisas" à web e suportar redes não confiáveis com pequena largura de banda e alta latência (RIZZARDI et al., 2016). No entanto, isso não é suficiente para garantir uma autenticação mútua entre clientes e servidores. A integridade e a confidencialidade das informações transmitidas também são deixadas em aberto.

3.3. MODELO DE SEGURANÇA ADOTADO

Para Ntuli e Abu-Mahfouz (2016), a criptografia de chave pública pode ser utilizada em momentos iniciais, os autores a consideram muito pesada para usar durante o funcionamento normal para aplicações *IoT*. Por esta razão, eles não usam criptografia de chave pública para proteger as comunicações. Em vez disso, o mesmo usa funções de *Hash* criptográficas. Antes de o produtor publicar conteúdo, uma função de *Hash* criptográfica é utilizada para produzir um valor de *Hash*. Então o valor de H*ash* é enviado juntamente com o conteúdo original e *token* de acesso. Quando o consumidor recebe o conteúdo, ele usa o valor de *Hash* e o *token* de acesso para validar a identidade, integridade e autenticidade da editora / conteúdo (NTULI; ABU-MAHFOUZ, 2016). A Figura 17, representa essa operação.

Figura 17: Forma tradicional de uso de Hash.



Fonte: Elaborado pelo autor.

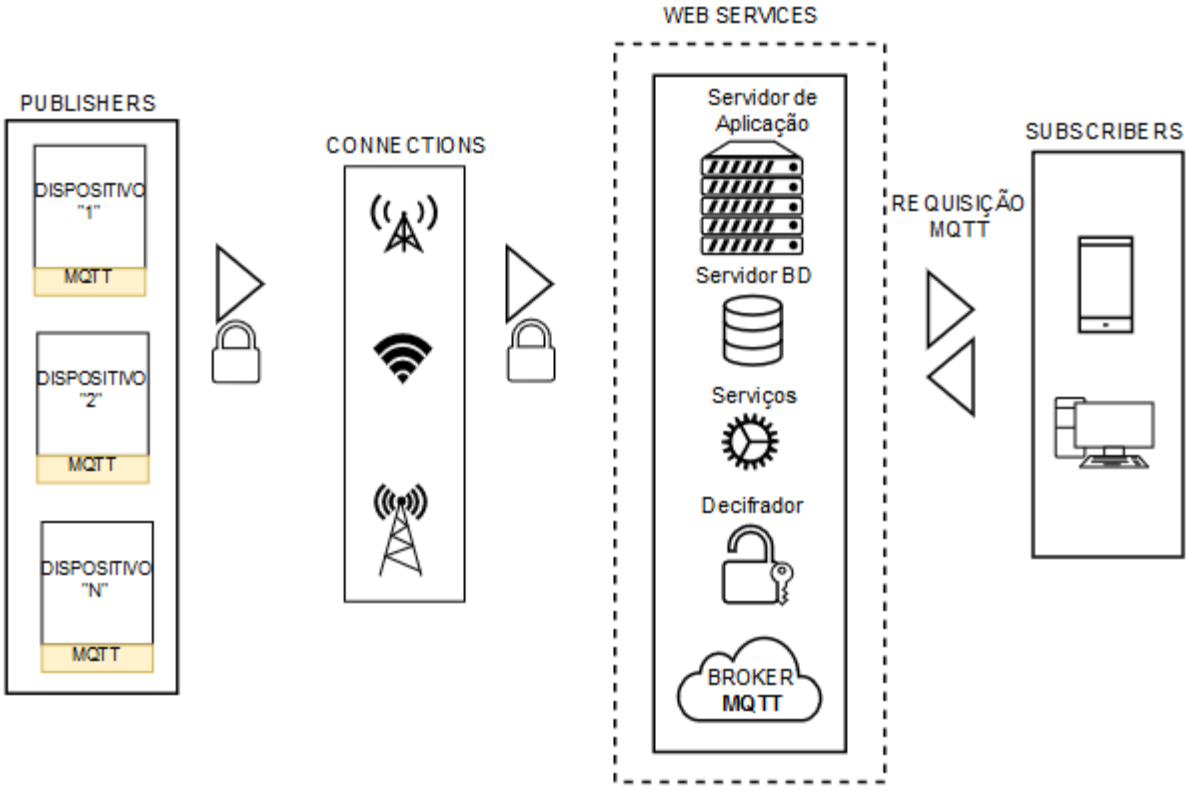
Segundo Ntuli e Abu-Mahfouz (2016), uma criptografia de chave pública é considerada pesada para a comunicação/transmissão dos dados principalmente em IoT, onde a preocupação com o consumo de recursos dos dispositivos é constante , em vez disso o uso de uma função de *Hash* criptográfica com *tokens* de acesso seria uma solução eficaz e leve para validar integridade e autenticidade dos dados e o dispositivo transmissor.

3.4. DESCRIÇÃO DA PROPOSTA (VISÃO GERAL)

Neste trabalho, foi projetada uma arquitetura de referência usando o MQTT como protocolo de aplicação, com o objetivo de fornecer confidencialidade aos dados que serão transmitidos

dos sensores para o servidor. Como pode ser visualizado na Figura 18, a arquitetura proposta funciona da seguinte maneira: Os dados serão transmitidos via o protocolo MQTT, dos dispositivos para o servidor, usando os mecanismos de segurança nativos do protocolo (usuário e senha) (GORGES et al., 2014). Além disso, uma função de *Hash* criptográfica é usada para garantir a confidencialidade dos dados, a função Hash é executada no dispositivo (descrita na seção seguinte). Esses dados serão armazenados no servidor de banco de dados. Por meio de uma aplicação será possível visualizar as informações geradas pelos dados armazenados.

Figura 18: Arquitetura Proposta.



Fonte: Elaborado pelo autor.

No cenário ilustrado na imagem anterior, os dados são gerados (valores numéricos) pelos dispositivos, para que então seja aplicada a técnica, onde é composta uma *String* contendo o valor gerado pelo dispositivo (valores de 0 a 99), o Salt (*String* com 10 caracteres) e um contador (de 0 a 99), para que então seja gerado o *Hash* desta *String*. Esse valor *Hash* é transmitido via protocolo MQTT para o *Web Service*, que por sua vez fica responsável por processar esses dados.

3.5. DESCRIÇÃO DA TÉCNICA DE SEGURANÇA PARA TRANSMISSÃO DOS DADOS

Para fornecer a segurança aos dados que serão enviados do dispositivo para o servidor, é utilizada uma função de *Hash* criptográfica. Essa função criptografa os dados que serão enviados do sensor para o *Broker* MQTT, para manter a confidencialidade dos mesmos.

Em sistemas que usam *Hash* como mecanismo de segurança para os dados que serão transmitidos por redes não seguras, a maneira tradicional de uso desta técnica seria enviar a mensagem junto com seu valor *Hash* correspondente (STALLINGS, 2014). Entretanto, essa abordagem pode apresentar alguns pontos fracos. Um exemplo seria que um invasor pode capturar a mensagem juntamente com o seu valor de *Hash e*, desta maneira, teria acesso à mensagem. Além disso, este invasor poderia alterar esta mensagem e gerar um novo *Hash* e enviá-los novamente para o seu destino. Neste caso, não seria possível identificar se ouve alguma alteração nos valores enviados pelos sensores.

Como forma de melhorar essa abordagem, este trabalho propõe enviar apenas o valor do *Hash* para o *Broker*. Desta maneira, mesmo que o invasor consiga ter acesso ao que foi transmitido, ele não conseguirá visualizar esses dados (o texto plano), pois os mesmos estão criptografados com o algoritmo de *Hash*. Neste projeto, optou-se por usar o SHAI-1[16], pois oferece uma "boa segurança", o que torna complexa a tarefa de quebrar essa criptografia com força bruta utilizando computadores domésticos (STALLINGS, 2014). Para Kuppens (2012) quebrar o *Hash* com força não é uma tarefa viável, seria necessário um alto poder de processamento, o que demandaria tempo, sendo uma tarefa inviável para aplicações que transmitem dados em tempo real. E ele é o mais leve dos algoritmos de *Hash* da família SHA.

Quando o *Broker* receber o valor *Hash*, ele os envia para o decifrador esse valor de *Hash* recebido, o decifrador tem armazenado uma tabela com todos os possíveis valores de *Hash* que poderiam ser gerados pelo dispositivo/sensor. Logo, é feita uma comparação do valor enviado com os valores da tabela para decifrar o *Hash*, chegando assim ao valor gerado originalmente pelo sensor. Neste caso são considerados apenas dados numéricos.

Um dos problemas do *Hash* é que ele está vulnerável a ataques de dicionário. Um ataque de dicionário é onde um atacante tem um banco de dados de senhas prováveis com seus respectivos *Hashes* (uma *WordList*) (KUPPENS, 2012). Como uma solução para este problema, tem-se o *Salt* que é uma porção aleatória de texto que é concatenado com valor original. Para a abordagem proposta aqui o *Salt* terá 10 variações pré-definidas no algoritmo,

[16] http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

a cada execução é escolhida aleatoriamente uma dessas variações para ser concatenada ao valor gerado pelo dispositivo.

É possível encontrar várias ferramentas disponíveis na web que já têm tabelas com valores *Hash* (WordLists de *Hash*), ver Tabela 2. Para decifrar um *Hash* usando essas ferramentas, basta digitar o seu valor e então o sistema consegue decifrar, fazendo a comparação do *Hash* que foi enviado com os armazenados na tabela. Isso no caso de senhas ou valores simples, se for uma sequência de caracteres com a mistura de letras, números e caracteres especiais, ou seja um uma *String* complexa, será muito improvável que a tabela contenha esses valores armazenados.

Como os sensores normalmente geram valores simples com poucos caracteres (por exemplo, números), seria muito simples decifrar seu valor *Hash*. Para evitar essa abordagem, neste trabalho é proposto adicionar uma sequência de caracteres (Salt) depois do valor que o sensor gerou, para que a partir desta nova *String* seja gerado o valor *Hash* a ser enviado.

Mesmo após o uso destas estratégias para aumentar a segurança do *Hash*, ainda pode ser encontrada uma falha, que é a seguinte: Dado que um invasor capturou os dados na rede, mesmo sem ele conseguir decifrar, ele pode ficar apenas repetindo o envio de um *Hash* que ele capturou (ataque de repetição). Portanto, o invasor não saberá qual informação está sendo enviada (valor original gerado pelo dispositivo/sensor), mas pode confundir o sistema repetindo o envio desse valor.

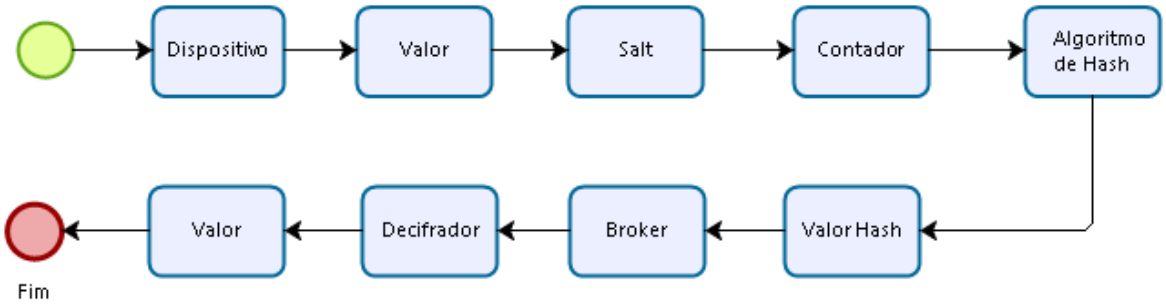
Para solucionar este problema, é acrescentado um contador ao valor concatenado e após isso é gerado o *Hash*. Logo, quando o valor for decifrado no servidor ele terá a seguinte sequência (valor + salt + contador), esse contador pode ir de 0 até 99 (é importante salientar que o contador poderia ser maior, isso poderia ser alterado de acordo com a necessidade do sistema), e ficar em um *loop*. Isso pode demostrar que não houve interceptação de valores *Hash*.

Como forma de limitar o escopo dos valores gerados, assume-se que serão gerados apenas números inteiros de 0 a 99 (é importante salientar que o esse valor poderia ser maior). Com essa sequencia (valor + *Salt* + contador), onde o valor iará de 0 a 99, o Salt tem 10 caracteres e o contador vai de 0 a 99, é possível gerar 100.000 *Strings* diferentes, para que possa ser criada uma *WordList* com as *Strings* e seus respectivos valores *Hash* e então armazenada no decifrador.

A técnica funciona da seguinte maneira: Um sensor X gera o valor “50” esse valor é concatenado com a *String* “DFERGFTJGR” e concatenado com o contador no valor de “25” formando então a seguinte String “50DFERGFTJGR25”, então é gerado o *Hash* da *String* que

resulta no seguinte valor de *Hash* “912B37EBBAFC05837EBD46CA76AB51177BC08834”, esse valor *Hash* é enviado para a um servidor *Web* via protocolo MQTT, esse valor é recebido pelo *Broker* que por sua vez o passa para o decifrador que tem uma *WordList* com todas as combinações de *Strings* que poderiam ser geradas, então o decifrador retorna com o valor enviado inicialmente que foi 50. O funcionamento da técnica pode ser visualizado na Figura 19.

Figura 19: Fluxo de execução do Hash na abordagem proposta.



Fonte: Elaborado pelo autor.

Com a técnica proposta neste trabalho (**String = “valor” + “salt” + “contador”**), é possível ter uma grande quantidade de possibilidades de valores *Hash* gerados. Assim, é possível visualizar a possibilidade de combinações usando a fórmula a seguir:

$$\boldsymbol{Fórmula = (P_o = V * A_{(m,p)} * N)}$$

P_o = Possibilidades de combinações
V = Vetor (0 a 99 = 100 valores)
$A_{(m,p)}$ = Arranjo (26^{10})
N = contador (0 a 99 = 100 posições)

$A(m,p) = m^p$
M = Tamanho do conjunto amostrado (26 = letras do alfabeto)
P = Número de elementos amostrado (10 = tamanho da String)
$100 \times 26^{10} \times 100$
*P_o: **1.411.670.956.533.760.000***

Essas possibilidades de combinações são alcançadas, usando apenas números inteiros, se forem usados números fracionários com apenas uma casa decimal, por exemplo (1.1; 1.2; ... 1.9) então se tem esse valor multiplicado por 10. É possível aumentar o tamanho da *String* do *Salt* de 10 caracteres para 26, assim contendo todas a letras do alfabeto, pois o *Hash* SHA-1 gerado independente do tamanho da entrada resulta em uma *String* com 40 bytes, então não faria diferença o Salt ter 10 caracteres ou 26. Seria possível também utilizar todos os símbolos e caracteres especiais do teclado, o que deixaria a técnica ainda mais robusta.

Para evitar a perda de dados, enviados pelos dispositivos, será usado o MQTT na Qos2 (qualidade de serviço 2), pois esta qualidade garante que os dados chegarão ao *Broker*, apesar de ser a mais pesada. É importante ressaltar que a técnica aqui proposta para garantir a segurança dos dados transferidos na rede não altera a estrutura do protocolo MQTT, e que esta técnica poderia ser usada em qualquer outro protocolo. Então o protocolo MQTT foi escolhido devido a sua eficiência e qualidade (devido a sua ampla adoção em sistemas *IoT*), já comprovada por diversas pesquisas (HAMIDA et al., 2013; NIRUNTASUKRAT; ISSARIYAPAT; PONGPAIBOOL, 2016; ZHOU; ZHANG, 2014).

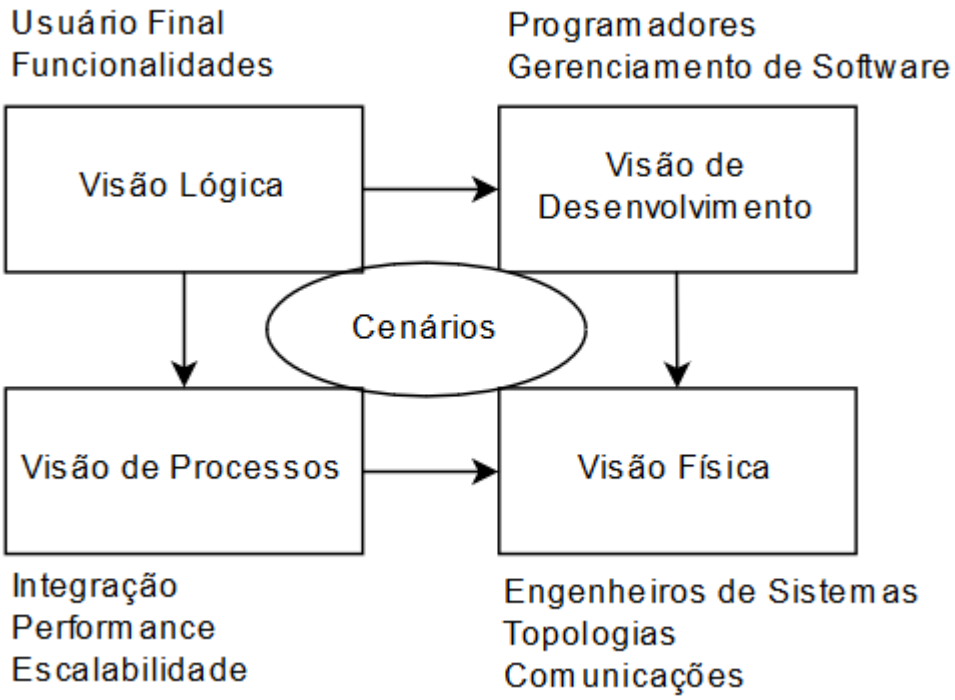
3.6. VISÕES DA ARQUITETURA (METODOLOGIA “4+1”)

Nesta seção, são apresentados subconjuntos de requisitos que serão abordados nesta proposta. A descrição da proposta é detalhada utilizando um método de descrição de Arquiteturas de *Software* (AS’s) baseado em visões, conhecido como “4+1” (*The 4+1 View Model of Architecture*) proposto por (KRUCHTEN, 1995a).

A descrição de uma arquitetura pode ser organizada em torno desses quatro pontos de vista e, em seguida, ilustrada por alguns casos de uso selecionados ou cenários que se tornam um quinto ponto de vista. A arquitetura é parcialmente evoluída a partir destes cenários (KRUCHTEN, 1995b).

Com o uso de múltiplas visões é possível descrever separadamente as preocupações de vários *stakeholders* da AS. Cada visão da arquitetura aborda um determinado conjunto de questões específicas dos envolvidos no processo de desenvolvimento: usuários finais, *designers*, gerentes, engenheiros de sistema, mantenedores, entre outros. Com o uso de múltiplas visões é possível ainda avaliar os requisitos funcionais e não funcionais separadamente (TOMAS, 2014). Estas visões abordam aspectos de relevância arquiteturais sob diferentes perspectivas (KRUCHTEN, 1995a). Na Figura 20, é mostrada metodologia 4+1 e suas visões:

Figura 20: Visões no Modelo “The 4+1 View Model”.



Fonte: Adaptado de kruchten (1995a).

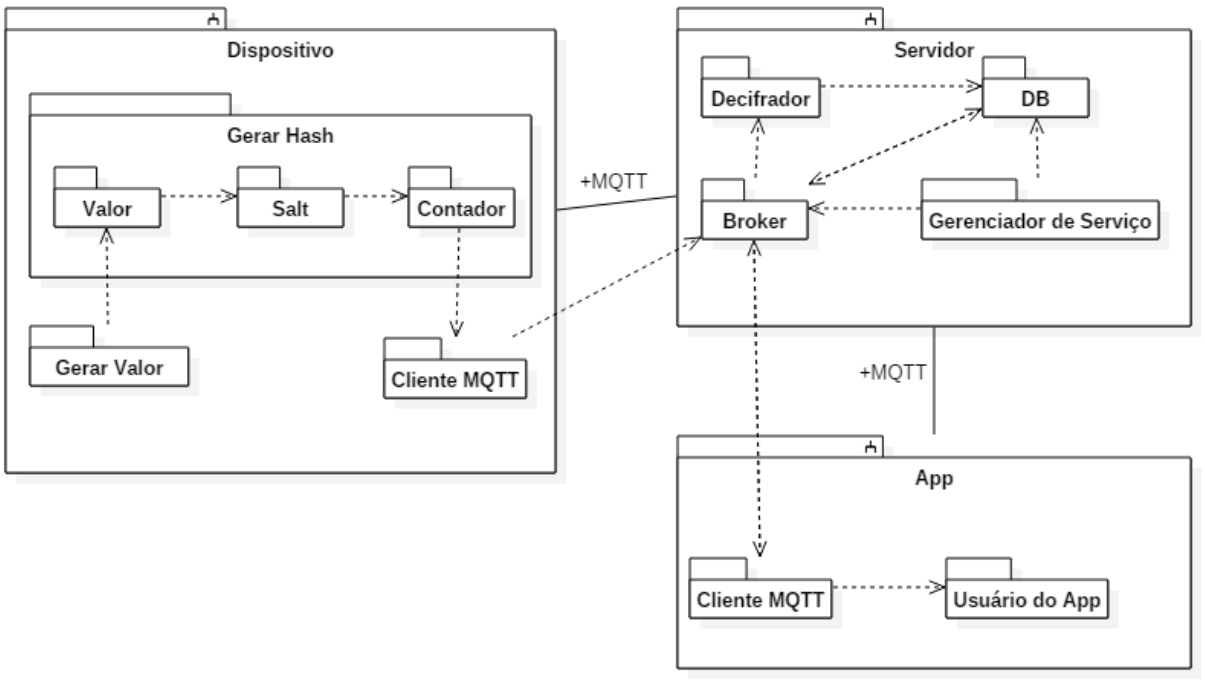
1. **A Visão Lógica:** Contém as classes de *design* mais importantes e sua organização em pacotes e subsistemas, descreve sistema em nível conceitual, demonstrando seus componentes e conectores. Trata o mapeamento dos requisitos funcionais para a arquitetura, refletindo abstrações-chave do domínio do problema;
2. **A Visão de Processos:** Contém a descrição dos diferentes processos do sistema, suas interações e configurações, e a alocação dos objetos e classes de design em tarefas;
3. **A Visão de Implementação/desenvolvimento:** Contém uma visão geral do modelo de implementação e sua organização em termos de módulos em pacotes e camadas. Descreve a organização estática do *software* no seu ambiente de implementação. A visão de desenvolvimento trata da possibilidade de reutilização, ou seja, do uso de componentes de prateleira, de bibliotecas do ambiente operacional do sistema ou de componentes de terceiros;
4. **A Visão Física:** Descreve o mapeamento do *software* para o *hardware* e reflete a natureza distribuída do sistema;
5. **A Visão de Cenários (Visão de Casos de Uso):** Utilizada para ilustrar o comportamento do sistema em sua arquitetura, conforme descrita pelas outras quatro visões. Para tal ilustração, alguns cenários de casos de uso devem ser selecionados.

Para Kruchten (1995a), nem toda arquitetura de *software* requer representações nas quatro visões deste modelo. Algumas visões que não são úteis podem ser omitidas. Por exemplo, a visão física poderia ser omitida se existisse apenas um processador ou a visão de processo poderia ser omitida se existisse apenas um processo no sistema.

3.6.1. Visão Lógica

Esta seção demonstra a organização da arquitetura proposta do ponto de vista lógico. Nela são especificados os principais elementos, tais como os módulos e os componentes principais. Também é especificada a *interface* entre estes elementos. Na Figura 21, é possível observar a visão lógica da arquitetura proposta para este trabalho.

Figura 21: Visão lógica.



Fonte: Elaborado pelo autor.

A arquitetura proposta é dividida em três subsistemas, são eles:

Subsistema 1 (Dispositivo): É dividido nos seguintes módulos:

1. Gera o valor
2. Gerar *Hash*
    - Recebe valor;
    - Concatena o valor recebido com o Salt;
    - Gera contador;
    - Concatena (Valor + *Salt* + Contador);

- Gera *Hash.*

3. Cliente MQTT (envia o valor de *Hash* para o servidor)

O dispositivo captura as informações do ambiente (por exemplo, sensor de temperatura, luminosidade, presença, ou qualidade da água). Em seguida, o dispositivo executa o algoritmo para concatenar o valor gerado com uma *String* e um contador e executa o algoritmo de *Hash*. Finalmente, esse valor será enviado para o servidor (Subsistema 2) via protocolo MQTT. Para tanto, é necessário que o dispositivo tenha um cliente MQTT instalado.

Subsistema 2 (Servidor): O servidor é dividido em cinco módulos, que são os seguintes:

1. *Broker*;
2. Decifrador;
3. Banco de Dados;
4. Gerenciador de Serviços;

O *Broker* é o intermediário entre as aplicações clientes MQTT. Ele recebe as mensagens dos clientes MQTT e as envia para o decifrador, que é responsável por decifrar o valor *Hash* para o valor originalmente gerado pelo sensor junto com o contador para então validar essa mensagem. Então o valor é armazenado no banco de dados, o gerenciador de serviços que controla os serviços da aplicação (controlador e regras de negócio).

Subsistema 3 (App): Aplicação é dividida em dois módulos, descritos a seguir:

1. Cliente MQTT;
2. App.

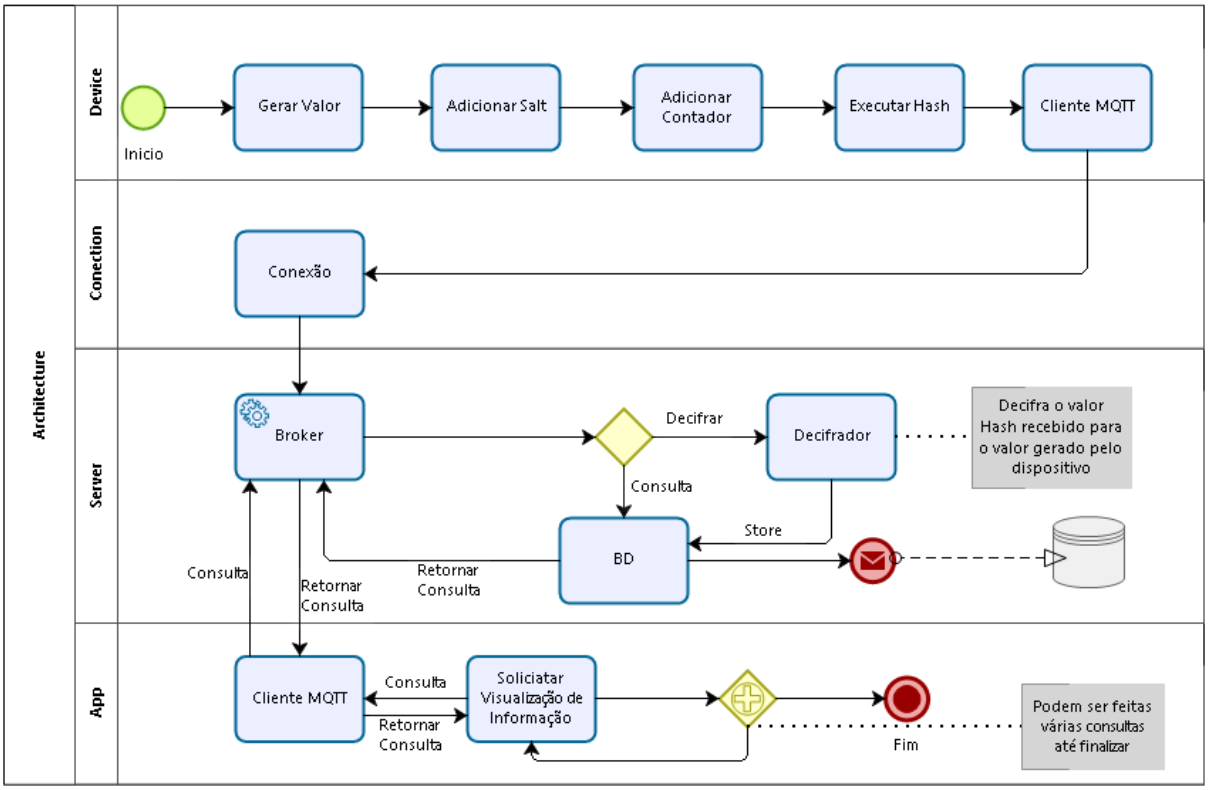
A aplicação do usuário envia uma requisição por meio do cliente MQTT para o banco de dados armazenado no servidor. Essa requisição chega até o *Broker*, que a encaminha para o banco de dados, que então retorna os dados solicitados para o *Broker*, que por sua vez, manda essas informações para o cliente MQTT na aplicação do usuário, que então consegue visualizar as informações que solicitou.

3.6.2. Visão de Processos

Nesta seção, é fornecida uma base que permitirá a compreensão da organização dos processos do sistema. Para tal, é ilustrada a decomposição dos processos do sistema, incluindo o mapeamento das classes e dos subsistemas para processos e *threads*. A visão de processos é

refinada durante cada iteração, para modelar essa visão foi usado a notação BPMN[17] (Business Process Management and Notation - Notação de Modelagem de Processos de Negócio). Na Figura 22 é possível observar a visão de processos.

Figura 22: Visão de Processos.



Fonte: Elaborado pelo autor.

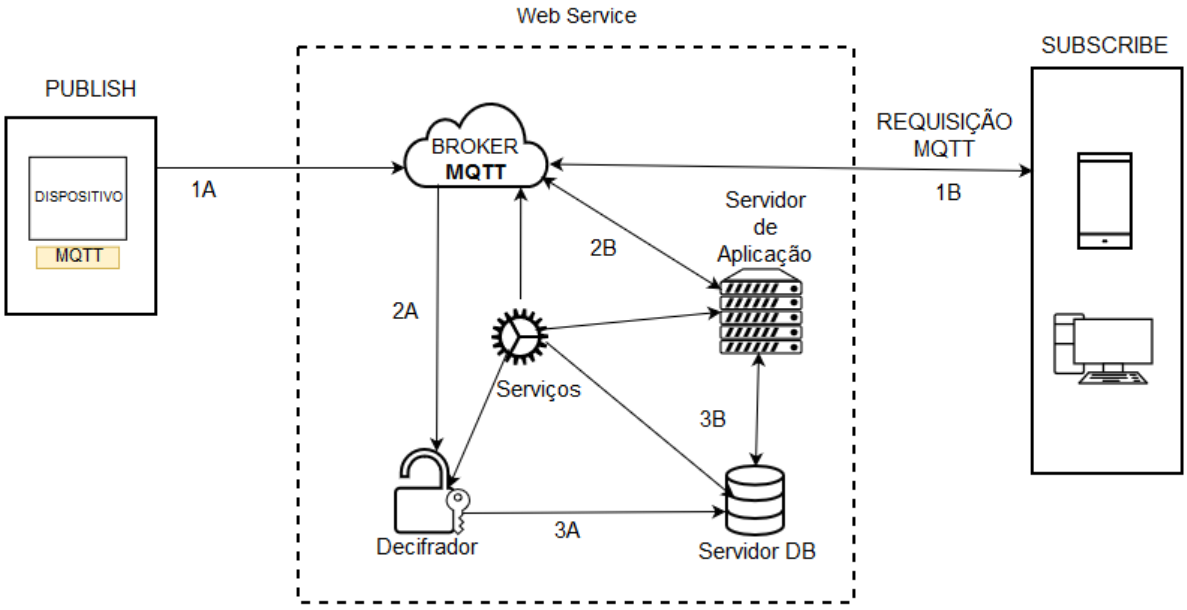
3.6.3. Visão de Implementação/Desenvolvimento

Esta seção descreve as estratégias utilizadas para o desenvolvimento de referência do sistema de acordo com a AS escolhida. São enumerados todos os subsistemas. Os diagramas de componentes ilustram como os subsistemas são organizados, suas camadas e hierarquias. Esta visão também ilustra as dependências de importação entre os subsistemas.

Com o intuito de melhorar a qualidade da arquitetura, foi adotado o Padrão Arquitetural (PA) *Broker*. Esse PA é o que se adequa mais aos objetivos e necessidades deste trabalho, pois ele foi idealizado para sistemas distribuídos, o que se assemelha com a arquitetura aqui proposta para sistemas de internet das coisas. A Figura 23, demostra a visão de implementação.

[17] http://www.bpmn.org/

Figura 23: Visão de implementação.



Fonte: Elaborado pelo autor

As informações demostradas na imagem acima são descritas a seguir:

Fluxo dos dados que são transmitidos dos dispositivos (PUBLISH - Web Services)

1A : Envia os dados já criptografados do dispositivo para o *Broker;*

2A : Envia os dados ainda criptografados do *Broker* para o decifrador, que os decifra;

3A : Envia os dados já decifrados para o banco de dados que é responsável por armazenar todos os dados gerados pelos dispositivos.

Fluxo da aplicação para consultar informações (SUBSCRIBE - Web Services)

1B : Solicita os dados armazenado no banco de dados para o *Broker*;

2B : O *Broker* então passa a solicitação da aplicação o "**SUBSCRIBE"** para o servidor de aplicação que que gerência o nível de acesso de cada usuário;

3B : O servidor de aplicação controla a comunicação entre os componentes.

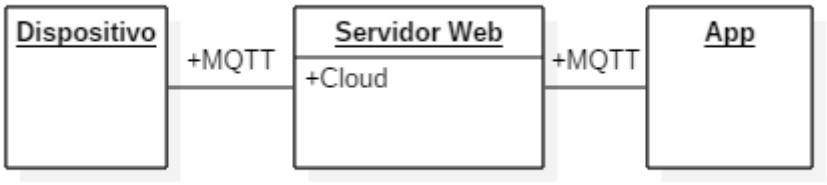
OBS: Esses fluxos fazem solicitações e retornam as informações solicitadas.

3.6.4. Visão Física

Esta seção permite compreender a distribuição física do sistema em um conjunto de nós de processamento, incluindo a distribuição física dos processos e *threads*, sendo refinada durante cada iteração. Como a arquitetura aqui proposta se trata de um ambiente distribuído, com vários acessos e a possibilidade de conectar vários dispositivos, isso demanda um certo grau

de escalabilidade. Segundo Tomas (2014), para este tipo de arquitetura é de fundamental importância à utilização de um ambiente *Cloud*, para manter esse serviço funcionando de acordo com a demanda exigida. Na Figura 24 é possível observar a visão física.

Figura 24: Visão Física da Arquitetura.



Fonte: Elaborado pelo autor.

3.6.5. Visão de Casos de Uso

Esta seção, demostra os casos de uso e cenários que englobam o comportamento da arquitetura, as classes ou os riscos técnicos significativos do ponto de vista da arquitetura. É descrito o que o sistema faz do ponto de vista dos atores/usuários, ou seja, descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os atores/usuários. A visão de casos de uso é refinada e considerada inicialmente em cada iteração.

Para os sistemas de *IoT*, os usuários não interagem diretamente com os sistemas, na arquitetura aqui proposta terá uma aplicação em que o usuário pode visualizar as informações geradas pelos dispositivos.

A arquitetura proposta tem como principal funcionalidade possibilitar que os usuários possam monitorar dados gerados por sensores (sensores de temperatura, de consumo de água, sensor de pressão em ambientes controlados, de luminosidade entre outros), então a interação do usuário com o sistema é mínima, ficando limitada apenas a solicitação para visualizar ás informações gerados pelos sensores.

3.7. CONSIDERAÇÕES FINAIS

Este capítulo apresentou a abordagem composta nesta pesquisa, é descrito o Padrão Arquitetural adotado nesta proposta. Para guiar a descrição desta proposta, foi apresentado também o protocolo selecionado para ser usado na abordagem, e detalhado porque o protocolo MQTT foi o selecionado. Foi descrito o modelo de segurança adotado para fornecer a confidencialidade. É possível também ter uma visão geral da abordagem composta e

descrita nesta dissertação. É apresentada a descrição de como funciona a técnica desenvolvida para adicionar a segurança aos dados transmitidos usando o MQTT em redes não seguras. Por fim é apresentado o método de descrição de AS's utilizado, conhecido como "4+1" (*The 4+1 View Model of Architecture*) proposto por (KRUCHTEN, 1995a), que se mostrou bastante efetivo para o contexto deste trabalho.

4. ESPECIFICAÇÃO, IMPLEMENTAÇÃO DO PROTÓTIPO E AVALIAÇÃO DA TÉCNICA

A avaliação da técnica proposta foi realizada da seguinte maneira: Inicialmente foi desenvolvido um protótipo de um sistema de monitoramento do consumo de água em tempo real. Posteriormente foi feita uma avaliação da técnica, testando seu funcionamento no sistema e eficácia no que diz respeito a garantir a confidencialidade dos dados que são transmitidos dos sensores para o *Broker* MQTT e quanto de recursos (memória e espaço de armazenamento) do dispositivo usado são consumidos. Para guiar a avaliação foi usada a abordagem GQM (Goal/Question/Metric – Objetivo/Questão/Métrica) (BASILI; CALDIERA; ROMBACH, 1994).

O Sistema de Monitoramento do consumo de água é integrado por quatro subsistemas: (I) medidor inteligente que coleta os dados de consumo (equipamento responsável por coletar as informações do consumo de água e enviar para o *Broker*), que será implantado na residência do cliente (II) o *Broker* que recebe, armazena e gerencia as informações enviadas pelo medidor, (III) o aplicativo de monitoramento para o cliente, permite que ele acompanhe seu consumo em tempo real, (IV) o aplicativo que coleta e centraliza os dados de todos os clientes, oferece a visualização e análise destes dados.

Como o objetivo do sistema é apenas para avaliar a eficácia da técnica proposta neste trabalho, foi desenvolvida apenas uma parte do sistema, os dados (valores numéricos) são gerados por um Arduino Uno, concatenados com o Salt e o contador, para que seja gerado o *Hash* desta *String* e então enviado por meio do protocolo MQTT para o *Broker*.

4.1. OBJETIVOS DO SISTEMA

Atualmente a água está ficando cada vez mais escassa, o Brasil, por exemplo, vem enfrentando uma falta de água por um período de tempo, o nordeste, sempre teve essa escassez de água (SILVA et al., 2016). Diante disso seria importante que as pessoas pudessem acompanhar seu consumo em tempo real, para que a partir disso possam controlar seu uso no dia a dia, desta maneira sendo uma ferramenta para redução do consumo de água.

Um sistema para monitorar o consumo de água em tempo real poderia ajudar a baratear os custos com o fornecimento de água, pois como o consumo será monitorado remotamente, não seria mais necessário gastos com a coleta do consumo manualmente, ou seja, o processo de coletar o consumo nas residências não necessitará de vários funcionários,

assim barateando os custos deste processo. Desta maneira o consumidor também pode ter uma redução em sua conta. Sem falar que a empresa pode ter uma visão em tempo real do consumo de água de cada residência.

Diante disto, esse sistema tem dois objetivos, o primeiro é proporcionar aos usuários o monitoramento em tempo real do consumo de água. O segundo objetivo é possibilitar que a fornecedora de água possa ter uma visão do consumo em cada residência, cada bairro, cada cidade, o sistema vai gerar dados do consumo dos clientes e a empresa vai monitorar esse consumo para que possa tomar decisões estratégicas com essas informações, desta maneira melhorando e otimizando a qualidade do serviço.

4.2. METODOLOGIA PARA AVALIAÇÃO

A metodologia usada para avaliação da técnica proposta será um estudo de caso, juntamente com alguns testes para verificar a eficácia da técnica para o uso do *Hash*. O sistema proposto será colocado em funcionamento para que seja testado se ele realmente faz o que se propõe.

Segundo Jang et al. (2016), dispositivos *IoT* requerem memória suficiente para reunir, enviar e receber dados de um dispositivo inteligente. O autor cita que as principais plataformas de dispositivos *IoT* foram projetadas por quatro empresas: *ARM mbed*[18], *ATMEL Arduino*[19], *Raspberry Pi*[20] e *Intel Edison*[21]. *ARM Mbed* tem especificamente um processador Cortex-M de 100 MHz e 32 bits. NXP, *Freescale*, TI, *STMicro* e *Nordic* são baseados na *Mbed*. O *Arduino* oferece Microprocessadores variando de 8 a 32 bits. A *Raspberry Pi* tem um ARM1176JZF 700 MHz, microprocessador de 32 bits. O *Intel Edison* tem um Quark de 400 MHz e 32 bits processador.

Para a avaliação, foram coletadas as informações de consumo de memória do algoritmo no dispositivo (Arduino UNO), usado para os testes de envio de dados. Além disso, foram capturados os valores *Hash* transmitidos do dispositivo para o *Broker* MQTT, para que sejam feitos testes que avaliem a eficácia da técnica. Também foram coletados dados de consumo de recursos do dispositivo.

Essas avaliações terão dois objetivos. O primeiro é comprovar que a técnica proposta é eficiente para fornecer a segurança (confidencialidade) aos dados transmitidos. O segundo é

[18] https://developer.mbed.org/
[19] https://www.arduino.cc/
[20] https://www.raspberrypi.org/
[21] http://www.intel.com.br/content/www/br/pt/support/boards-and-kits/intel-edison-boards/intel-edison-board-for-arduino.html

comprovar que a técnica funciona em dispositivos com recursos limitados, como por exemplo um Arduino UNO com 2KB de memória e 32KB de espaço de armazenamento, utilizado neste trabalho. No Arduino podem ser usados vários sensores como por exemplo sensor de temperatura, luminosidade, presença ou até mesmo sensores para mediar a qualidade da água.

4.2.1. Descrição do ambiente de testes

Para os testes e avaliações, foi utilizada uma infraestrutura de serviço da plataforma *FIWARE* e um Arduino UNO para gerar e enviar os valores para o servidor na *FIWARE*.

4.2.1.1. FIWARE

A Comissão Europeia lançou uma parceria público privada voltada para o futuro da Internet, a *Future Internet* PPP[22] (FI-PPP). Através da FI-PPP foi criado o projeto *Future Internet Ware* (FI-WARE) que tem por objetivo a construção de uma plataforma para a internet do futuro. Assim, surgiu a plataforma FI-WARE[23], ou simplesmente *FIWARE* (FIWARE, 2017).

A *FIWARE* é uma plataforma genérica e extensível para serviços da Internet do Futuro por meio de um conjunto de especificações abertas, disponibilizadas por interfaces de aplicações (*Application Programming Interfaces*, APIs) e implementadas em componentes denominados habilitadores genéricos (*Generic Enablers*, GEs) (BATISTA et al., 2016).

Segundo Batista et al. (2016) o desenvolvimento dos GEs objetiva compor o núcleo da plataforma *FIWARE*. Os GEs são as componentes fundamentais para a plataforma *FIWARE*. As implementações de cada GE são compostas por um conjunto de componentes, que juntos suportam as funcionalidades fornecidas através de APIs, desta maneira provendo interfaces interoperáveis de acordo com as especificações abertas definidas para cada GE.

Os GEs estão agregados nos capítulos técnicos (*Technical Chapters*) que estão relacionados a um conjunto de funcionalidades especificas fornecidas pela *FIWARE*, por exemplo, segurança, infraestrutura, etc. Atualmente existem sete capítulos técnicos referentes ao release 4 da especificação da *FIWARE* (BATISTA et al., 2016), que são os seguites: *Cloud Hosting, Data/Context Management; Internet of Things (IoT) Services Enablement; Applications Services and Data Delivery, Security, Interface to Networks and* Devices (I2ND) *Architecture e Advanced Web-based User Interface*.

Os componentes abertos e o alto investimento da FI-PPP têm proporcionado a *FIWARE* um grande e crescente uso a nível Internacional (BATISTA et al., 2016). Aliado a

[22] https://www.fi-ppp.eu/
[23] https://www.fiware.org/

isso a grande quantidade de GEs, bem como, o foco em fornecer elementos utilizáveis para vários domínios, foi o que levou a adoção da *FIWARE* aos níveis internacionais, sendo usado em mais de 75 cidades pelo mundo e por mais de 1000 *Startups* (FIWARE, 2017). Os campos de atuação das ferramentas são ainda mais diversos, indo desde agricultura, saúde, energia, meio ambiente, transporte, segurança urbana, tecnologia da informação e mídias sociais (BATISTA et al., 2016).

A nível de Brasil, a *FIWARE* vem sendo adotada em projetos que envolvem seu uso em *Smart City* e *IoT*, por pesquisadores de algumas instituições, dentre elas se destacam, o INES [24] (Instituto Nacional de Ciência e Tecnologia para Engenharia de Software) no Centro de Informática da Universidade Federal de Pernambuco, o Instituto Metrópole Digital[25] na Universidade Federal do Rio Grande do Norte e no *FIWARE Lab* São Paulo[26] da Escola Politécnica da Universidade de São Paulo.

4.2.1.2. Arduino UNO

Arduino nasceu no Ivrea Interaction Design Institute[27] como uma ferramenta fácil para prototipagem rápida, voltada para estudantes sem formação em eletrônica e programação. É uma plataforma de eletrônica de código aberto baseada em *hardware* e *software* fáceis de usar. As placas Arduino são capazes de ler entradas (luz em um sensor, um dedo em um botão ou uma mensagem no Twitter) e transformá-las em uma saída (ativar um motor, ligar um LED, publicar algo on-line). É possível dizer à placa o que fazer, enviando um conjunto de instruções para o microcontrolador na placa. Para isso, é usada a linguagem de programação Arduino. A linguagem de programação trabalhada no Arduino consiste em um conjunto de funções da linguagem C/C++ com pequenas alterações (ARDUINO, 2017).

Na placa é possível adicionar diversos tipos de componentes eletrônicos direcionados e programados para uma determinada atividade, permite que sejam criadas pequenas rotinas de programação, que tornam possível a aquisição automática de dados, possibilitando medidas de grandezas físicas como: tempo, temperatura, corrente elétrica, tensão etc. A plataforma e arquivos são licenciados pela *Creative Commons* que permite tanto uso pessoal, bem como comercial e obras derivadas, desde que seja dado crédito ao Arduino e liberação de seus projetos sob a mesma licença (ARDUINO, 2017).

[24] http://www.ines.org.br/?tag=ufpe
[25] http://portal.imd.ufrn.br/
[26] http://fiwarelabsp.org/fiware-lab/
[27] https://interactionivrea.org/en/index.asp

Ao longo dos anos Arduino tem sido o cérebro de milhares de projetos, desde objetos cotidianos até complexos instrumentos científicos. Uma comunidade mundial de criadores (estudantes, amadores, artistas, programadores e profissionais) se reuniu em torno desta plataforma de código aberto, suas contribuições somaram uma incrível quantidade de conhecimento acessível que pode ser de grande ajuda para novatos e especialistas (ARDUINO, 2017; SILVA; CAVALCANTE; CAMILO, 2016).

Segundo informações da Arduino (2017) esse *hardware* tem sido utilizado em milhares de projetos e aplicações diferentes. O software Arduino é fácil de usar para iniciantes, mas flexível o suficiente para usuários avançados. Ele é executado em Mac, Windows e Linux. Professores e alunos usam-no para construir instrumentos científicos de baixo custo, para provar princípios de química e física, ou para começar com programação e robótica. *Designers* e arquitetos constroem protótipos interativos, músicos e artistas usam-no para instalações e experimentam com novos instrumentos musicais. Os fabricantes, é claro, usam-no para construir muitos dos projetos exibidos na *Maker Faire*[28]. Qualquer um (crianças, amadores, artistas, programadores) pode começar a mexer apenas seguindo as instruções passo a passo de um kit ou compartilhando ideias on-line com outros membros da comunidade Arduino. Nesta pesquisa foi usado um Arduino UNO com as seguintes configurações, ver Figura 25.

Figura 25: Especificações técnicas do Arduino Uno usado nesta pesquisa

Microcontrolador	ATmega328P
Voltagem operacional	5V
Voltagem de alimentação (recomendada)	7-12V
Voltagem de alimentação (limites)	6-20V
Pinos I/O digitais	14 (dos quais 6 podem ser saídas PWM)
Pinos de entrada analógica	6
Corrente contínua por pino I/O	40 mA
Corrente contínua para o pino 3.3V	50 mA
Memória flash	32 KB (2KB usados para o bootloader)
SRAM	2 KB
EEPROM	1 KB
Velocidade de clock	16 MHz

Fonte: Arduino (2017)

4.2.2. Definição da Avaliação

Neste trabalho foi realizada uma avaliação com o objetivo de identificar se a abordagem proposta nesta pesquisa atende seus objetivos, que é fornecer confidencialidade aos dados que

[28] http://makerfaire.com/

são transmitidos por dispositivos com recursos limitados em *IoT*, ou seja a abordagem tem que fornecer confidencialidade e consumir poucos recursos (memória e espaço de armazenamento) do dispositivo . Para isso foi utilizada a abordagem Goal/Question/Metric (GQM) apresentada por Basili; Caldiera; Rombach (1994) para apoiar a organização e a elaboração da avaliação. Para facilitar o entendimento sobre a utilização da abordagem GQM neste trabalho, as fases de definição, de planejamento, de coleta de dados e de análise foram criadas com os dados e informações deste trabalho.

A definição dos objetivos (*Goals*) para a avaliação deste trabalho foi organizada da seguinte maneira:

Objetivo global

Identificar se a técnica composta neste trabalho realmente pode fornecer confidencialidade aos dados transmitidos por dispositivos *IoT*, e ao mesmo tempo é leve o suficiente para ser executada em dispositivos com pouca poder computacional (dispositivos *IoT*).

Questões e Métricas

A definição das questões e métricas (*Question/Metric*) para avaliação deste trabalho foi organizada da seguinte maneira:

Questão 1: A técnica composta nesta pesquisa consegue fornecer a confidencialidade aos dados transmitidos por dispositivos *IoT*?

Métrica: É feita a tentativa de decifrar os valores gerados pelos dispositivos usando a técnica composta neste trabalho. Esta métrica será detalhada a seguir neste trabalho na etapa 2 da avaliação.

Questão 2: Quanto de recursos (memória e espaço de armazenamento) dos dispositivos é consumido para executar a técnica composta nesta pesquisa?

Métrica: Análise do consumo de recurso dos dispositivos usando a técnica. Esta métrica será detalhada a seguir neste trabalho. Esta métrica será detalhada a seguir neste trabalho na etapa 3 da avaliação.

Para essa avaliação, foi desenvolvido um sistema que gera dados de um sensor (dados numéricos) e os envia para a *Internet* usando a técnica proposta nesta dissertação. Para testar a eficácia do mesmo foi feita uma prova de conceito, onde foi testado em um ambiente simulado para transmissão de dados de consumo de água, foram gerados valores para simular o consumo de água em uma residência, esses valores foram gerados aleatoriamente. Os dados

foram enviados de um Arduino UNO para o servidor *WEB* via protocolo MQTT, que está disponível na plataforma *FIWARE*.

Depois foi feita a captura dos dados que são transmitidos do dispositivo para o servidor, foram capturados os dados transmitidos pelo protocolo MQTT para que seja feita uma tentativa de decifrar os dados que estão criptografados com o algoritmo de *Hash* criptográfico**. Para validação da eficácia da técnica proposta para o uso do Hash, serão seguidas as seguintes etapas:**

Etapa 1: Esta etapa será chamada de avaliação 1. Nesta etapa serão gerados valores *Hash* de 0 a 99 e verificar se as ferramentas conseguem quebrar/decifrar o *Hash*, ou seja, será usado apenas os valores gerados pelo dispositivo. Esta avaliação tem como objetivo verificar a eficácia e eficiência das ferramentas para decifrar os valores *Hash*.

Etapa 2: Esta etapa será chamada de avaliação 2. Nesta etapa serão gerados valores de 0 a 99 usando a técnica proposta, para então testar se as ferramentas conseguem quebrar/decifrar o *Hash*. Essa avaliação tem como objetivo testar a eficácia da técnica proposta para o uso do *Hash*.

Etapa 3: Esta etapa será chamada de avaliação 3, e tem como objetivo desenvolver um sistema usando a técnica proposta e fazer um estudo de caso, para esses testes será usado um Arduino UNO, com os seguintes objetivos:

1. Verificar quanta memória e espaço de armazenamento o dispositivo gasta para receber o valor e concatenar tudo (valor + *Salt* + contador);
2. Será feito um teste para verificar quanto de memória e espaço de armazenamento a execução da função *Hash* consome no dispositivo;
3. Constatar quanto o protocolo MQTT consome de memória e espaço de armazenamento do dispositivo para transmitir dados;
4. Testar quanto à concatenação e o *Hash* consomem juntos de memória e espaço de armazenamento do dispositivo;
5. Verificar quanto os três consomem de memória e espaço de armazenamento do dispositivo para capturar dados de um dispositivo, gerar o valor *Hash* e enviar para o *Broker* MQTT.

4.2.3. Coleta e Análise dos Dados

Para testar a técnica proposta, onde a mesma tem que consumir poucos recursos do *Hardware*, por esta razão optou-se por usar um Arduino UNO, devido a suas baixas capacidades de processamento e memória. Na Figura 25 (seção 4.2.1.2) é possível ver a configuração do dispositivo. Foram capturados os dados gerados pelo dispositivo Arduino usando a técnica criada para adicionar segurança a arquitetura.

4.3. AVALIAÇÃO

Para avaliar a eficácia dos valores *Hash* gerados, foram usadas 8 ferramentas para tentar decifrá-los. Essas ferramentas estão disponíveis na *Web* de forma gratuita. Foram feitos três testes: no primeiro foram gerados valores *Hash* dos números de 0 a 99, no segundo foram gerados valores *Hash* usando a abordagem proposta neste trabalho (valor + *salt* + contador) e, no terceiro, foram analisados os consumos de memória e espaço de armazenamento gastos pelo dispositivo usando a técnica proposta neste trabalho. Na Tabela 2 é possível visualizar todas as ferramentas usadas:

Tabela 2: Ferramentas para decifrar Hash.

Ferramenta	Disponível em:	Tamanho da Base Usada (*WordList*)
Hashkiller	hashkiller.co.uk/sha1-decrypter.aspx	312.072.000.000
Hashtoolkit	https://hashtoolkit.com/reverse-sha1-hash/	6.761.351.755
Md5decrypt	http://md5decrypt.net/en/Sha1/	3.772.681.045
Insidepro	http://insidepro.com/	1.768.092.948
Crackstation	https://crackstation.net/	1.500.000.000
Sha1.gromweb	http://sha1.gromweb.com/	116.372.31
Isc.sans.edu/tools	https://isc.sans.edu/tools/reversehash.html	20.621.101
Hash Code Craker 1.2.1	http://crackerpassword.sourceforge.net/	18.138.922

Fonte: Elaborado pelo autor.

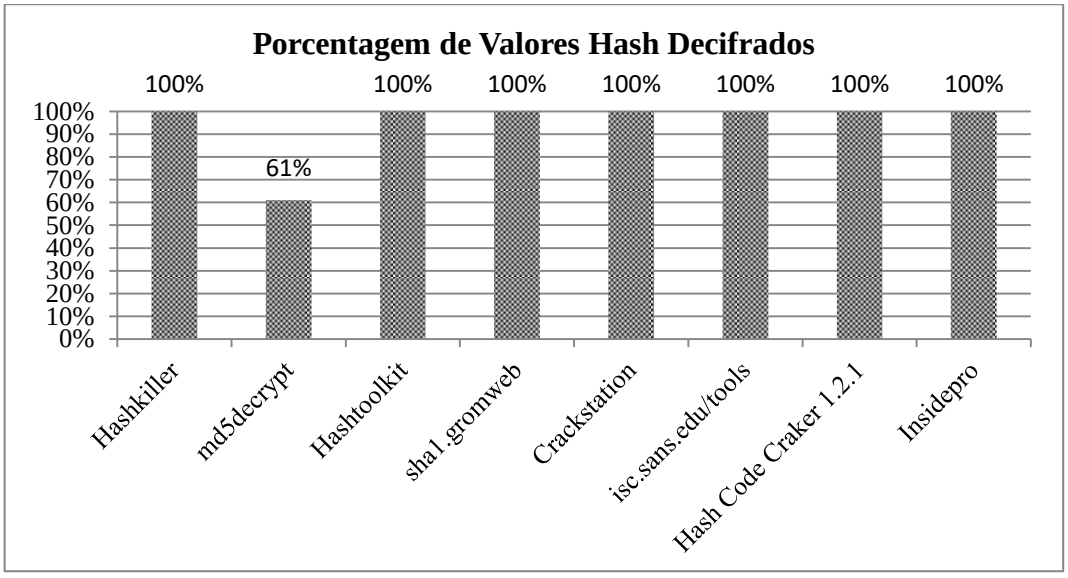
Avaliação 1 (valores de 0 a 99):

A avaliação 1 teve por objetivo demonstrar a eficácia das ferramentas mencionadas na tabela 2. Para avaliar as ferramentas foram gerados valores *Hash* dos números de 0 a 99,

posteriormente foram feitas tentativas de decifrar esses cem valores *Hash* usando as 8 ferramentas.

Após o teste foi possível observar que a ferramenta ***md5decrypt*** conseguiu decifrar apenas 61% dos valores *Hash* que foram testados. Entretanto, as outras ferramentas conseguiram decifrar todos os valores *Hash*, o que demostra a eficácia das ferramentas usadas neste teste para valores simples. Essas informações podem ser visualizadas na Figura 26.

Figura 26: Desempenho das ferramentas para decifrar os valores Hash.



Fonte: Elaborado pelo autor.

Outras duas ferramentas foram utilizadas para decifrar valores *Hash*, mas devido à baixa eficiência, os resultados foram descartados. Foram utilizadas as ferramentas *HashCat*[29] e *FindMyHash*[30] disponíveis no *KaliLinux*[31]. Ambas as ferramentas trabalham com *Wordlist*, nos testes efetuados com a *Wordlist* disponível na *Web*. Os resultados apresentados foram insatisfatórios, em exemplos preliminares, foram gerados valores *Hash* com nomes simples, em que todas as outras 8 ferramentas conseguiram decifrar, mas as duas ferramentas não obtiveram êxito.

Avaliação 2 (valor + salt + contador):

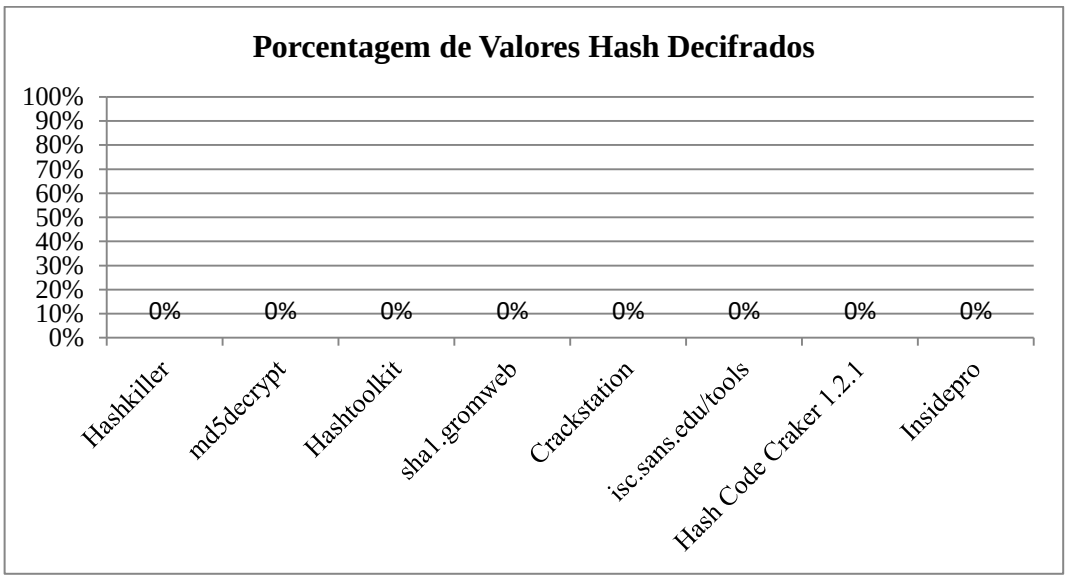
A avaliação 2 teve por objetivo demonstrar a eficácia da técnica proposta para fornecer confidencialidade aos dados transmitidos por dispositivos *IoT*. Para esta foram gerados valores *Hash* dos números de 0 a 99 usando a técnica citada acima na seção 3.5,

[29] https://hashcat.net/hashcat/
[30] http://tools.kali.org/password-attacks/findmyhash
[31] https://www.kali.org/

posteriormente foram feitas tentativas de decifrar esses cem valores *Hash* usando as 8 ferramentas.

Nesse teste foi possível constatar a eficácia da técnica proposta, das 8 ferramentas usadas para tentar decifrar os valores *Hash*, nenhuma conseguiu obter êxito, ou seja, a técnica teve um índice de 100% de aproveitamento, o que demostra sua eficiência. Como demonstrado na Figura 27.

Figura 27: Desempenho das ferramentas para não decifrar usando a técnica criada.



Fonte: Elaborado pelo autor.

Diante dos resultados obtidos nesta avaliação são obtidas evidências de que a abordagem realmente conseguiu atingir seu objetivo de fornecer confidencialidade aos dados transmitidos.

Avaliação 3 (Consumo de Recursos de Dispositivo):

Para avaliação da técnica, foi desenvolvido um algoritmo usando a abordagem proposta. O *hardware* usado para os testes foi um Arduino Uno, que tem espaço de armazenamento para programas de 32.256 bytes e 2.048 bytes de memória dinâmica (RAM). Com esse algoritmo foram obtidos os seguintes resultados em relação aos consumos de memória e espaço de armazenamento.

Na primeira etapa da avaliação 3, que tem por objetivo verificar quanto o algoritmo usado consome, apenas para receber o valor do sensor e adicionar a técnica (valor + String + contador), foram obtidos os seguintes resultados:

- Consome de 224 bytes (10%) da memoria RAM do dispositivo para executar o algoritmo;
- O Algoritmo ocupa 3.860 bytes (11%) de espaço de armazenamento do dispositivo quando é descarregado no mesmo.

A segunda etapa, que teve por objetivo verificar quanto o algoritmo usado apenas para gerar o *Hash* de uma *String* com 14 caracteres (cada *String* gerada usando a técnica desenvolvida neste trabalho terá 14 caracteres), consome de recursos do dispositivo. Para gerar o *Hash* na técnica, foi usada a biblioteca para Arduino chamada *SpritzCipher*[32], a partir dessa biblioteca foi desenvolvido o algoritmo para gerar a String e então gerar o *Hash* dessa *String* . Nesta etapa foram obtidos os seguintes resultados:

- Consome 298 bytes (14%) da memoria RAM do dispositivo para executar o algoritmo;
- O Algoritmo ocupa 3.760 bytes (11%) de espaço de armazenamento do dispositivo quando é executado.

Na terceira etapa que teve por objetivo verificar quanto o cliente MQTT consome de recursos do dispositivo, foram obtidos os seguintes resultados:

- Consome 731 bytes (35%) da memória RAM do dispositivo para executar o algoritmo;
- O Algoritmo ocupa 11.586 bytes (35%) de espaço de armazenamento do dispositivo quando é executado.

Na quarta etapa que teve por objetivo verificar quanto o algoritmo usado para gerar a String e o *Hash*, ou seja juntando o algoritmo da etapa 1 com o algoritmo da etapa 2 (criando apenas um algoritmo), foram obtidos os seguintes resultados:

- Consome 316 bytes (15%) da memoria RAM do dispositivo para executar o algoritmo;
- O Algoritmo ocupa 3.620 bytes (11%) de espaço de armazenamento do dispositivo quando é executado.

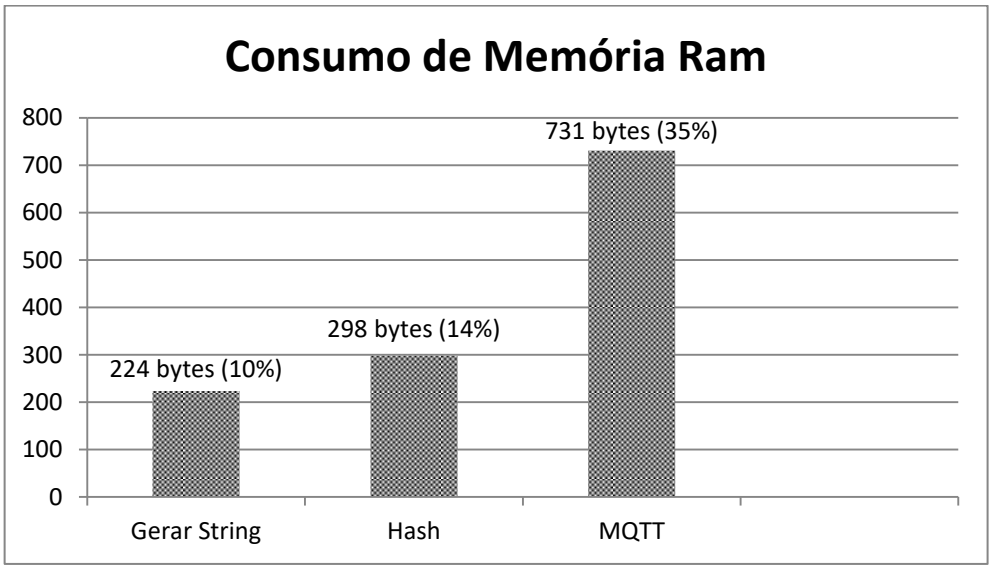
[32] https://github.com/abderraouf-adjal/ArduinoSpritzCipher

Na quinta etapa que teve por objetivo verificar quanto à técnica como um todo consome de recursos do dispositivo (juntando os três algoritmos, das etapas 1, 2 e 3), foram obtidos os seguintes resultados:

- Consome 1.077 bytes (50%) da memoria RAM do dispositivo para executar o algoritmo;
- O Algoritmo ocupa 15.206 bytes (46%) de espaço de armazenamento do dispositivo quando é executado.

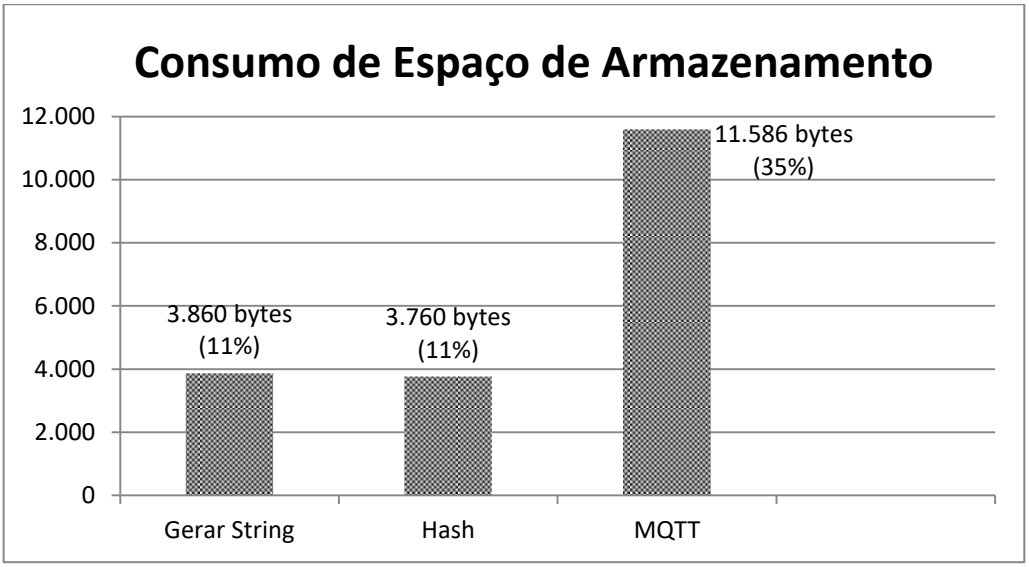
Os resultados da avaliação 3 podem ser visualizados nas Figuras 28 e 29:

Figura 28: Consumo de memória do Arduino pelos algoritmos.



Fonte: Elaborado pelo autor.

Figura 29: Consumo de espaço de armazenamento do Arduino pelos algoritmos.



Fonte: Elaborado pelo autor.

Com esta avaliação foi possível perceber que a abordagem composta neste trabalho pode ser usada em dispositivos com poucos recursos, pois a mesma consumiu apenas 316 bytes, o equivalente a 15% da memória RAM do Arduino usado para gerar os valeres. O código fonte desenvolvido para as avaliações usando a abordagem estão disponíveis no *GitHub*[33].

4.4. CONSIDERAÇÕES FINAIS

Este capítulo tem por objetivo descrever como foi avaliada a técnica proposta nesta dissertação e como foi desenvolvido o processo de avaliação da solução. Foi descrito como foram desenvolvidos os testes usados para avaliar a eficácia da técnica desenvolvida para garantir a segurança dos dados transmitidos pelos dispositivos em redes não seguras. Em seguida, foi descrito à metodologia e a definição das avaliações usadas para avaliar a arquitetura e a técnica criada para adicionar segurança, e como foi o processo de coleta dos dados, as ferramentas e os critérios usados nas avaliações, são demostrados também os resultados obtidos nas três avaliações feitas, esses resultados fornecem indícios sobre a eficácia da técnica proposta.

[33] https://github.com/flavioneves89/HashEmArduino-MQTT

5. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este capítulo tem por objetivo apresentar as conclusões da pesquisa descrita nessa dissertação, especificando os resultados, as dificuldades e limitações encontradas ao longo do desenvolvimento e por fim algumas sugestões para trabalhos futuros.

5.1. CONTRIBUIÇÕES DA PESQUISA

Do ponto de vista computacional este trabalho tem quatro contribuições principais, que são as seguintes:

1. Descrever e projetar uma arquitetura que forneça uma solução de segurança para os dados transmitidos em redes não seguras para sistemas de *IoT*;
2. Compor uma técnica para aumentar a segurança dos dados, essa técnica usa *Hash* para fornecer confidencialidade aos dados transmitidos em redes não confiáveis por dispositivos com pouco poder de processamento. A grande vantagem da técnica, é que a mesma pode ser usada em dispositivos com pouca capacidade;

De acordo com a avaliação, a solução, criada para fornecer segurança aos dados, apresentou-se eficaz no que diz respeito à confidencialidade dos dados transmitidos usando a mesma.

5.2. DIFICULDADES E LIMITAÇÕES DA PESQUISA

Ao longo do desenvolvimento desta pesquisa, foram encontradas várias dificuldades para garantir sua validade, uma delas era qual o PA mais adequado para esse tipo de sistema. Existem vários protocolos de aplicação usados para o desenvolvimento de sistemas para *IoT*. Diante disto, surgiu a dificuldade de escolher qual seria o mais apropriado para usar neste trabalho.

A seguir são listadas as principais limitações desta pesquisa:

1. A avaliação foi realizada em apenas um tipo de *Hardware*, o Arduino UNO.
2. A técnica desenvolvida neste trabalho é voltada apenas para fornecer confidencialidade a dados numéricos.
3. A avaliação que objetivou verificar a eficácia da técnica para apenas dois critérios, segurança e consumo de recursos do dispositivo (memória e armazenamento) são pontos que limitam a avaliação.
4. A técnica só pode ser usada em *Hardware* que aceita programação.

5.3. TRABALHOS FUTUROS

Além deste trabalho apresentar as contribuições citadas na seção 5.1, outras perspectivas são possíveis. Diante disso, a seguir são listadas algumas direções para trabalhos futuros desta pesquisa:

1. Realizar avaliações usando a técnica desenvolvida nesta pesquisa em outros protocolos para transmissão de dados;
2. Desenvolver estudos para analisar o quanto de largura de banda é consumido comparado com uma abordagem tradicional enviando o *Hash* e a mensagem juntos;
3. Realizar testes para verificar o quanto é consumido de energia comparado com uma abordagem tradicional enviando o *Hash* e a mensagem juntos.
4. Fazer testes para avaliar a Arquitetura projetada usando a técnica.

5.4. CONCLUSÃO

A segurança é um dos grandes problemas da Internet das Coisas, essa área que tem enormes potenciais para toda a sociedade, desde a indústria a agricultura, logo torna-se fundamental investir em novas abordagens que possam contribuir para a solução deste problema. Diante disso esta dissertação objetivou desenvolver uma solução de segurança para Internet das Coisas.

Conforme as avaliações discutidas no capítulo 4, é possível concluir que o principal objetivo deste trabalho, a especificação e projeto de uma técnica em uma Arquitetura de Software que forneça segurança aos dados transmitidos em sistemas *IoT*, usando o protocolo MQTT, foi alcançado nesta pesquisa, como demostrado pelos resultados das avaliações.

Foram feitos testes com objetivo de avaliar apenas eficácia da técnica no que diz respeito a manter a confidencialidade dos dados gerados pelos dispositivos e transmitidos usando a técnica projetada e implementada nesta dissertação.

A partir dos testes efetuados para verificar a eficácia da técnica proposta neste trabalho, é possível concluir que todos os objetivos tantos o geral quantos os específicos foram atingidos.

REFERENCIAS

(IETF), I. E. T. F. **US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF) Abstract**, 2011.

AL-FUQAHA, A. et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. v. 17, n. 4, p. 2347–2376, 2015.

AMARAN, M. H. et al. A Comparison of Lightweight Communication Protocols in Robotic Applications. **Procedia Computer Science**, v. 76, n. Iris, p. 400–405, 2015.

ARDUINO. **Technical specs**. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 11 jan. 2017.

ASHTON, K. That Internet of Things Thing. p. 4986, 2009.

AZEVEDO, R. P. M. **Seleção de Padrões para a Arquitetura de Software: Uma Abordagem Baseada em Procura de Termos e Sinônimos**. [s.l.] Universidade Federal de Viçosa, 2014.

BANDYOPADHYAY, S.; BHATTACHARYYA, A. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. **2013 International Conference on Computing, Networking and Communications, ICNC 2013**, p. 334–340, 2013.

BASILI, V.; CALDIERA, G.; ROMBACH, H. **Goal Question Metric ApproachEncyclopedia of Software Engineering**, 1994.

BATISTA, T. et al. SmartMetropolis - Plataformas e Aplicações para Cidades Inteligentes. In: Natal. Universidade Federal do Rio Grande do Norte - Instituto Metrópole Digital - SmartMetropolis. p. 47.

BIRUKOU, A. A survey of existing approaches for pattern search and selection. **Proceedings of the 15th European Conference on Pattern Languages of Programs - EuroPLoP '10**, p. 1, 2010.

BORGIA, E. The internet of things vision: Key features, applications and open issues. **Computer Communications**, v. 54, p. 1–31, 2014.

BUSCHMANN, F. et al. **Pattern-Oriented Software Architecture Volume 1: A System of**

Patterns. 1. ed. New York, USA: John. Wiley & Sons Ltd, 1996. v. Vol. 1

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. Pattern-oriented software architecture, Vol. 5: On Patterns and Pattern Languages. v. 5, p. 2007, 2007.

CISCO, T. Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update , 2010 – 2015. **Growth Lakeland**, v. 2011, n. 4, p. 2010–2015, 2011.

COLLINA, M. et al. Internet of Things application layer protocol analysis over error and delay prone links. **2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop, ASMS/SPSC 2014**, v. 2014–Janua, p. 398–404, 2014.

CONCEIÇÃO, R. A. DA. **Um Protocolo de Autenticação e Autorização Seguro para Arquiteturas Orientadas a Serviços**. Brasília. Universidade de Brasília, 2014.

FIWARE. **Fiware: ABOUT US**. Disponível em: <https://www.fiware.org/about-us/>.

FOWLER, M. et al. **Padrões de arquitetura de aplicações corporativas**. 1. ed. [s.l: s.n.].

FRIGIERI, E. P.; MAZZER, D.; PARREIRA, L. F. C. G. M2M Protocols for Constrained Environments in the Context of IoT : A Comparison of Approaches. **XXXIII Brazilian Telecommunications Symposium (SBrT 2015)**, p. 1–4, 2015.

GARTNER. **Gartner Identifies the Top 10 Strategic Technology Trends for 2016**. Disponível em: <http://www.gartner.com/newsroom/id/3143521>. Acesso em: 20 dez. 2016a.

GARTNER. **Gartner's 2015 Hype Cycle for Emerging Technologies Identifies the Computing Innovations That Organizations Should Monitor**. Disponível em: <http://www.gartner.com/newsroom/id/3114217>. Acesso em: 20 dez. 2016b.

GARTNER. Gartner Identifies The Top 10 Strategic Technology Trends for 2016. **Gartner**, p. 3143521, 2015c.

GORGES, M. et al. **Using machine-to-machine / "Internet of Things" communication to simplify medical device information exchange**. 2014 International Conference on the Internet of Things (IOT). **Anais**...IEEE, out. 2014Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84924232228&partnerID=tZOtx3y1>

HAMIDA, S. T. BEN et al. Towards efficient and secure in-home wearable insomnia monitoring and diagnosis system. **13th IEEE International Conference on BioInformatics and BioEngineering, IEEE BIBE 2013**, 2013.

HARRISON, N. B.; GUBLER, E.; SKINNER, D. Software Architecture Pattern Morphology in Open- Source Systems. 2016.

IBM; EUROTECH. MQTT V3.1 Protocol Specification. p. 1–42, 2010.

JANG, S. et al. An Efficient Device Authentication Protocol Without Certification Authority for Internet of Things. **Wireless Personal Communications**, 2016.

JUNG, M. et al. Things-to-cloud communication : technology overview and design considerations. **2015 5th International Conference on the Internet of Things (IoT)**, p. 177–178, 2015.

KITCHENHAM, B. et al. Systematic literature reviews in software engineering - A systematic literature review. **Information and Software Technology**, v. 51, n. 1, p. 7–15, 2009.

KRAIJAK, S.; TUWANUT, P. A survey on internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends. **2015 IEEE 16th International Conference on Communication Technology (ICCT)**, p. 26–31, 2015.

KRUCHTEN, P. The 4+ 1 view model of architecture. **Software, IEEE**, v. November 1, n. November, p. 9, 1995a.

KRUCHTEN, P. Architectural Blueprints—The "4+1" View Model of Software Architecture. **IEEE Software**, v. 12, n. 6, p. 42–50, 1995b.

KUPPENS, L. L. **Utilização de Dicionários Probabilísticos Personalizados para Decifrar Arquivos em Análises Periciais**. Brasília. Universidade de Brasília, 2012.

LESJAK, C. et al. Securing smart maintenance services: Hardware-security and TLS for MQTT. **Proceeding - 2015 IEEE International Conference on Industrial Informatics, INDIN 2015**, p. 1243–1250, 2015.

LEVY, N.; LOSAVIO, F. Analyzing and Comparing Architectural Styles. **SCCC ’99**

Proceedings of the 19th International Conference of the Chilean Computer Science Society, p. 87, 1999.

LI, S.; XU, L. DA; ZHAO, S. The internet of things: a survey. **Information Systems Frontiers**, v. 17, n. 2, p. 243–259, 26 abr. 2015.

LUZURIAGA, J. E. et al. **Handling mobility in IoT applications using the MQTT protocol**. 2015 Internet Technologies and Applications, ITA 2015 - Proceedings of the 6th International Conference. **Anais**...2015

MAREK, S. **SK Telecom to Invest $9B for IoT and 5G**. Disponível em: <https://www.sdxcentral.com/articles/news/sk-telecom-to-invest-9-billion-for-iot-and-5g/2017/01/?utm_source=sdnc_slider&utm_medium=link&utm_campaign=links&utm_source=SDxCentral.com+Mailing+List&utm_campaign=0ee908d0f4-SDxCentral+Newsletter+1%2F12%2F17&utm_>. Acesso em: 12 jan. 2017.

MINERAUD, J. et al. A gap analysis of Internet-of-Things platforms. **Computer Communications**, v. 89–90, p. 5–16, 2016.

MOTWANI, K. et al. Smart Nursing Home Patient Monitoring System. n. 6, p. 98–102, 2016.

NIRUNTASUKRAT, A.; ISSARIYAPAT, C.; PONGPAIBOOL, P. Authorization Mechanism for MQTT-based Internet of Things. v. 6, 2016.

NTULI, N.; ABU-MAHFOUZ, A. A Simple Security Architecture for Smart Water Management System. **Procedia Computer Science**, v. 83, p. 1164–1169, 2016.

RIZZARDI, A. et al. AUPS: An Open Source AUthenticated Publish/Subscribe system for the Internet of Things. **Information Systems**, v. 62, p. 29–41, 2016.

SANTOS, B. P. et al. Internet das Coisas: da Teoria à Prática. In: **Homepages.Dcc.Ufmg.Br**. Belo Horizonte. Minas Gerais. p. 52.

SILVA, J. L. D. S.; CAVALCANTE, M. M.; CAMILO, R. S. Plataforma Arduino integrado ao PLX-DAQ : Análise e aprimoramento de sensores com ênfase no LM35. v. 35, p. 1–12, 2016.

SILVA, E. M. S. DA et al. Sustentabilidade e responsabilidade socioambiental: O uso

indiscriminado de água. **Revista Maiêutica**, v. 4, n. 1, p. 57–66, 2016.

SINGH, B. **How Internet of Things (IoT) Are Going To Impact Your Business?** Disponível em: <http://www.business2community.com/big-data/internet-things-iot-going-impact-business-01572401#mv8PiX5RmVgIXfxh.97>. Acesso em: 20 dez. 2016.

SINGH, M. et al. Secure MQTT for Internet of Things (IoT). **Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015**, p. 746–751, 2015.

STALLINGS, W. **Criptografia e Segurança de Redes de Redes: Principios e Práticas**. 6. ed. Cambridge: Cambridge University Press, 2014. v. 1

TOMAS, G. H. R. P. **Uma Arquitetura Para Cidades Inteligentes Baseada Na Internet Das Coisas**. Recife. Universidade Federal de Pernambuco, 2014.

ZDUN, U.; KIRCHER, M.; VOLTER, M. Remoting patterns: design reuse of distributed object middleware solutions. **IEEE Internet Computing**, v. 8, n. 6, p. 60–68, nov. 2004.

ZHOU, C.; ZHANG, X. **Toward the Internet of Things application and management: A practical approach**. Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014. **Anais**...IEEE, jun. 2014Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84908876758&partnerID=tZOtx3y1>