Leandro Loffi

PROTOCOLO OTIMIZADO PARA AUTENTICAÇÃO MÚTUA EM IOT NO CONTEXTO DE *FOG COMPUTING*

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Carlos Becker Westphall.

Coorientadora: Prof.ª Dra. Carla Merkle Westphall.

Florianópolis
2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Loffi, Leandro
Protocolo otimizado para autenticação mútua em IOT no contexto de fog computing / Leandro Loffi ; orientador, Carlos Becker Westphall, coorientadora, Carla Merkle Westphall, 2019.
88 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Ciência da Computação, Florianópolis, 2019.

Inclui referências.

1. Ciência da Computação. 2. Autenticação Mútua. 3. Fog Computing. 4. Fatores de Autenticação. I. Westphall, Carlos Becker. II. Westphall, Carla Merkle. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. IV. Titulo.

Leandro Loffi

PROTOCOLO OTIMIZADO PARA AUTENTICAÇÃO MÚTUA EM IOT NO CONTEXTO DE *FOG COMPUTING*

Esta Dissertação foi julgada aprovada para obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 27 de fevereiro de 2019.

_____

Prof. José Luís Almada Güntzel, Dr.
Coordenador do Curso

_____

Prof. Carlos Becker Westphall, Dr.
Universidade Federal de Santa Catarina – UFSC
Orientador

_____

Prof.ª Carla Merkle Westphall, Dra.
Universidade Federal de Santa Catarina - UFSC
Coorientadora

Banca Examinadora:

_____

Prof. Bruno Richard Schulze, Dr.
Laboratório Nacional de Computação Científica - LNCC

_____

Prof. Eros Comunello, Dr.
Universidade do Vale do Itajaí - Univali

_____

Prof. Jean Everson Martina, Dr.
Universidade Federal de Santa Catarina - UFSC

_____

Profa. Michelle Silva Wangham, Dra. Eng.
Universidade do Vale do Itajaí - Univali

Este trabalho é dedicado a todos que,
de alguma forma, ajudaram no
crescimento da computação no Brasil

AGRADECIMENTOS

Primeiramente, a Deus, que me concede a vida, saúde, inteligência e a capacidade de desenvolver ciência para o Brasil.

Aos meus pais, Francisco Loffi e Inês Wernke Loffi, e ao meu irmão, André Fernando Loffi, pelo incentivo e motivação a mim dedicados.

À minha namorada, Chazana Talita Jasper, pelo apoio emocional e contribuições à realização deste trabalho.

Agradecimentos especiais à equipe do Laboratório de Redes e Gerência (LRG), em especial aos Professores Dr. Carlos B. Westphall e Dra. Carla M. Westphall, além dos amigos e colegas, Dr. Jorge Werner, Me. Cristiano Souza, Me. Hugo V. Sampaio, Bel. Eduardo de M. Koneski, Bel. Johann Westphall, Lukas D. Grüdtner, JoãoVitor Cardoso e ao Ricardo do Nascimento Böing.

Aos professores do mestrado que marcaram minha vida acadêmica: Dr. Daniel N. P. Rodriguez, Dr. Elder R. Santos, Dr. Jean E. Martina, Dr. Ricardo F. Custódio e Dra. Vania Bogorny, compartilhando seus conhecimentos com dedicação.

Por fim, o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

A história é escrita pelos vencedores.
(Bertrand Russell)

RESUMO

Fog Computing é uma área da Ciência da Computação que está em construção e constante evolução, e que, em conjunto com a segurança da informação, torna o paradigma mais confiável e seguro para as plataformas da borda da *Internet of Things*. A autenticação de dispositivos restritos de memória apresenta grandes problemas, já que o consumo de memória é elevado, quando aplicada a outros modelos que tenham a finalidade de autenticação mútua. O presente trabalho tem por objetivo propor um modelo de autenticação que tenha o código otimizado, que autentique de forma mútua as partes em um ambiente de *Internet of Things* aplicado no contexto de *Fog Computing*; e por meio da metodologia hipotético-dedutiva apresentam-se as deduções e hipóteses para a avaliação do novo método proposto. Ao final da pesquisa, constatou-se que o protocolo de autenticação mútua pode ser avaliado por meio da viabilidade técnica da autenticação otimizada, por ela consumir em torno de 119,5 KB (compilado) no desenvolvimento em C++ e 48,5 KB (*bytecode*) no desenvolvimento em Java.

Palavras-chave: Autenticação Mútua. *Fog Computing*. Fatores de Autenticação.

ABSTRACT

Fog Computing is an area of Computer Science that is under constant construction and evolution, and in conjunction with information security, the paradigm becomes more reliable and secure for IoT's edge platforms. The authentication of restricted memory devices has major problems because memory consumption is high when applied with other models that have the purpose of mutual authentication. The purpose of this work is to propose an authentication model that has the optimized code, which authentically authenticates the parties in an Internet of Things environment applied in the context of Fog Computing. Through the hypothetical-deductive methodology, the deductions and hypotheses for the evaluation of the new proposed method. At the end of the research it can be verified that the mutual authentication protocol can be evaluated through the technical feasibility of optimized authentication, consuming around 119.5 KB (compiled) in development in C++ and 48.5 KB (bytecode) in Java development.

Keywords: Mutual Authentication. Fog Computing. Authentication Factors.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AES	*Advanced Encryption Standard*
AVISPA	*Automated Validation of Internet Security Protocols and Applications*
CoAP	*Constrained Application Protocol*
DoS	*Denial of Service*
DDoS	*Distributed Denial of Service*
DTLS	*Datagram Transport Layer Security*
GLOMONET	*Global Mobility Networks*
HTTP	*HyperText Transfer Protocol*
ID	Identity
IoT	*Internet of Things*
IP	*Internet Protocol*
IPSec	*Internet Protocol Security*
MANET	*Mobile Ad hoc Networks*
MEC	*Mobile Edge Computing*
MITM	*Man-in-the-middle*
MQTT	*Message Queue Telemetry Transport*
NIST	*National Institute of Standards and Technology*
NoT	*Network of Things*
RFID	*Radio-Frequency IDentification*
RPL	*Routing Protocol for Low-power and Lossy Networks*
RSA	R. Rivest, A. Shamir e L. Adleman
RSSF	Redes de Sensores Sem Fio
SPAN	*Security Protocol ANimator for AVISPA*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
6LoWPAN	*IPv6 over Low-Power Wireless Personal Area Networks*

SUMÁRIO

1 INTRODUÇÃO

A evolução de equipamentos conectados à Internet possibilita interação e transferência de dados entre objetos usados no cotidiano por meio de redes. Cada objeto tem suas peculiaridades, porém, pode-se considerar que cada um deles tem a função de enviar e receber dados entre o mundo real e a Internet, formando, assim, uma rede IoT (*Internet of Things* – Internet das Coisas). A principal característica de uma rede IoT é ter equipamentos que caracterizam uma rede de sensores e atuadores (XIA et al., 2012).

Redes IoT podem ser organizadas em três camadas que são diferenciadas pela localização do processamento: camada de *Cloud Computing*, camada de *Fog Computing* e camada de *Edge Computing* (IORGA et al., 2018).

Esses equipamentos de redes IoT devem possuir meios de segurança adequados para os dados coletados, pois é necessário confiar nesses dados para poder avaliar, decidir e agir em determinados casos relacionados à aplicação final.

O conceito de segurança no contexto da informação refere-se à proteção dos dados e à preservação de seus valores. Um dos mecanismos para obter segurança é realizar a autenticação, isto é, fazer a verificação da identidade de uma entidade para provar e garantir a veracidade dessa identidade (SCHNEIER, 1995). A autenticação é normalmente a primeira etapa nos controles de segurança de um ambiente.

A autenticação pode ser de via única ou mútua. Na autenticação de via única apenas uma das partes apresenta sua identificação e a prova dessa identificação, por exemplo, o nome de usuário e a senha. Já na autenticação mútua, ambas as partes que estão interagindo precisam trocar suas identificações e suas provas de identificação.

Vários são os fatores que podem ser usados para realizar a autenticação: conhecimento de algo (por exemplo a senha), posse de algo (por exemplo um cartão físico), característica biométrica (por exemplo a impressão digital de uma pessoa) e contexto (por exemplo o endereço IP da entidade ou o local onde a entidade se encontra).

Em uma rede IoT, cada dispositivo pode receber um único identificador na rede. Ao realizar acesso aos dispositivos na rede a autenticação pode ser usada para garantir a segurança na interação entre dispositivos. A autenticação mútua permite melhorar a segurança nas interações de uma rede IoT. Também é possível usar vários fatores para melhorar a segurança do processo de autenticação, chamada atualmente

de autenticação multifator. Por exemplo, uma autenticação multifator poderia usar a senha e o IP da entidade para provar sua identidade.

Dispositivos localizados na borda de uma rede IoT têm baixo poder de processamento e são limitados, considerando o consumo de memória. Por esse motivo, esses dispositivos têm certas limitações na forma de implementação de segurança e por isso deve haver uma otimização para se adequar ao ambiente proposto (IVAN, et al. 2008).

No caso desse trabalho a otimização foi aplicada ao código da autenticação mútua resultando em uma autenticação mútua otimizada. O uso de autenticação multifator também foi proposta e desenvolvida no código.

A literatura apresenta trabalhos de pesquisa que usam métodos de autenticação julgados adequados para os fins específicos, como por exemplo, um ID (*identity*) único e exclusivo no método de funcionamento (HU et al., 2018) (GOPE et al. 2017), bem como uma análise simultânea de fatores de autenticação (PIRAMUTHU; DOSS, 2017). Outro método válido é a autenticação mútua com dispositivos físicos, especificamente a RFID (*Radio-Frequency Identification*) (TEWARI; GUPTA, 2017). No entanto, esses trabalhos não abordam de forma combinada a autenticação mútua e o uso de vários fatores.

O presente estudo aplica uma nova estratégia de autenticação mútua com integridade em conjunto com métodos criptográficos existentes para dispositivos com restrição de memória. Desse modo, a estratégia proposta engloba três fatores de autenticação. O primeiro fator é o uso de uma função de desafio-resposta, que é composta de funções matemáticas e visa garantir que a chave pública chegará ao destinatário com segurança. O segundo fator é a análise do tempo de resposta de cada mensagem enviada, o que visa amenizar os ataques do tipo *man-in-the-middle*. O terceiro fator é o uso de um *nonce*, para identificação única e a garantia de *freshness* de cada mensagem enviada.

1.1 PROBLEMAS

Métodos de autenticação de via única e mútua são aplicados à *Internet of Things*. Nesta seção são descritos problemas existentes para viabilizar a autenticação em redes IoT.

Amin et al. (2018) argumenta que a melhor forma de fazer a autenticação é utilizar a camada de *Cloud Computing* que possui mais recursos disponíveis. No entanto, quando o tempo de resposta entre as partes é um fator usado na autenticação, o sistema pode não conseguir estabilizar a conexão.

A estabilização de conexão está definida neste trabalho como o momento em que já ocorreu com sucesso a identificação das partes, a troca de chaves e a concordância das chaves, podendo ser transferido dados cifrados com a chave simétrica.

Grüdtner et al. (2018) propõem um modelo de autenticação com a utilização do fator de autenticação baseado em desafio-resposta. O modelo permite que os dispositivos escolham as funções e um número aleatório, porém, não garante a segurança da chave pública que é enviada.

Já Esiner e Datta (2019) e Piramuthu e Doss (2017) descrevem que a autenticação deve ocorrer de forma única na parte do servidor que recebe as informações. No entanto, isso somente é válido quando utilizado um ambiente com sensor e servidor, desconsiderando o conjunto de atuadores. Usando atuadores deve-se manter válida a autenticação das duas partes, tanto com os dispositivos sensor/servidor como também atuador/servidor.

Como existe restrição de memória em alguns dispositivos não é possível a utilização de certificado digital X.509 já que esse certificado pode ter tamanho considerado proibitivo pela carência de memória dos dispositivos.

Outro problema que deve ser considerado é que a transferência de números pequenos quando usados nos mecanismos criptográficos podem facilitar a quebra durante a transmissão do tráfego nas redes IoT.

1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um modelo de autenticação otimizada, que autentique de forma mútua as partes por meio de fatores de autenticação em um ambiente de *Internet of Things* aplicado no contexto de *Fog Computing*.

Para alcançar o objetivo principal, são definidos os seguintes objetivos específicos:

a) Desenvolver estratégias de fatores de identificação das partes;
b) Propor um método de autenticação mútua otimizado;
c) Implementar o método proposto em um ambiente de *Internet of Things* no contexto de *Fog Computing*; e
d) Avaliar o modelo de autenticação otimizado no ambiente IoT que seja restrito de memória nas partes envolvidas.

Ao final da pesquisa espera-se ter um mecanismo de autenticação mútua otimizado avaliado com diferentes protótipos restritos de memória. Nesses dispositivos, espera-se ter espaço necessário para executar alguma outra aplicação após a autenticação, além dessa aplicação utilizar a capacidade criptográfica das cifras simétricas para a troca de dados. Em suma, espera-se atingir todos os objetivos específicos propostos.

1.3 JUSTIFICATIVA

O uso indevido ou não desejado dos dados pessoais de um usuário pode provocar graves ameaças à privacidade (VILLARREAL, 2017). Por isso, para manter os dados em sigilo utilizam-se técnicas de cifragem de dados, ou seja, os dados são cifrados com chaves que devem ser de conhecimento comum entre as partes.

Outro ponto prejudicial ao ambiente IoT é a utilização de variados e inúmeros dispositivos, de diferentes características, tais como armazenamento, processamento e consumo de energia. Mais especificamente, em relação ao armazenamento, esses dispositivos estão sendo reduzidos ao máximo, para apenas conterem espaço para a aplicação fim, o que leva a segurança desses dispositivos a ficar em segundo plano (KHAN; SALAH, 2018). Nessas condições, o dano pode ser grande quando não for utilizada a segurança adequada.

1.4 MÉTODO

No presente trabalho é utilizado o método Hipotético-Dedutivo. Em Lakatos e Marconi (2011) é apresentado um esquema que mostra as etapas do método hipotético-dedutivo, que foi retirado de Popper (1975): 1º Conhecimento, 2º Problema, 3º Conjectura, 4º Falseamento.

Utiliza-se esta opção pois neste trabalho existe um problema que é fazer a autenticação em redes IoT (fase do conhecimento), existem hipóteses sobre a solução do problema (fase do problema), existem experimentações desenvolvidas durante o trabalho (fase de conjecturas/hipóteses) e também testes para as verificações das hipóteses (fase de falseamento). Nesse sentido a metodologia a seguir, resumidamente explicitada na Figura 1, é apresentada para a obtenção do objetivo descrito na Seção 1.2 do presente trabalho, e baseia-se nos seguintes passos:

1) Realizar um levantamento bibliográfico em livros e artigos científicos sobre *Internet of Things*, *Fog Computing*, autenticação mútua, otimizada/leve e reduzida e TLS/SSL (*Transport Layer Security* / *Secure Sockets Layer*);

2) Analisar estudos existentes relacionados à proposta;

3) Definir os conceitos básicos necessários para a compreensão do assunto e em relação à definição do protocolo;

4) Definir um modelo de protocolo que faça a autenticação mutuamente entre as partes, com base nos protocolos anteriores e existentes;

5) Desenvolver um protótipo na linguagem de programação C++ e na linguagem orientada a objetos (Java), de forma que, em trabalhos futuros, ele possa ser incorporado como base de alguma implementação de ambientes IoT;

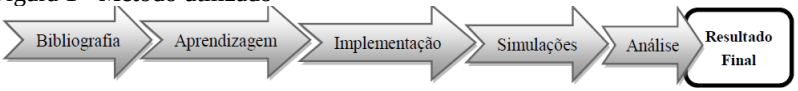
6) Otimizar o desenvolvimento do protocolo para ser utilizado em ambientes que sejam restritos de memória;

7) Definir uma estrutura real que possa utilizar o protocolo otimizado;

8) Simular a aplicabilidade e a utilidade do protocolo mediante a aplicação do modelo em um estudo de caso; e

9) Analisar a viabilidade técnica e a correta operação do protocolo em ambientes restritos de memória.

Figura 1 - Método utilizado



Fonte: elaborada pelo autor (2019).

1.5 ORGANIZAÇÃO DO TRABALHO

Após esta introdução, no Capítulo 2 será apresentada a fundamentação teórica, que é inerente ao tema principal, e no Capítulo 3

serão expostos os trabalhos relacionados a esta proposta. O Capítulo 4 descreverá o protocolo desenvolvido, que inclui a explanação sobre os fatores de autenticação. No Capítulo 5, será apresentada a avaliação do protocolo através da verificação da capacidade de segurança das cifras. Em seguida será analisada a viabilidade técnica da proposta por meio do desenvolvimento de um protótipo usando duas linguagens de programação, inicialmente C++ e posteriormente Java. Por fim, serão expostas as considerações finais, apresentando as contribuições, as limitações e a proposta de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são descritos os conceitos principais para o entendimento do trabalho.

2.1 INTERNET OF THINGS

Atzori, Iera e Morabito (2010) definem a *Internet of Things* (IoT) como uma variedade de objetos e coisas interligadas, como por exemplo, um portão eletrônico de garagem que se interliga a um ar-condicionado. Neste caso, o acionamento do portão acionaria o ar-condicionado, por meio de uma rede local, quando alguém entrasse na garagem, com a finalidade de manter agradável o ambiente, No entanto, Voas (2016) argumenta que a rede IoT contém uma subdefinição, a *Network of Things* (NoT), que difere sutilmente do paradigma IoT. Mais especificamente, a NoT é uma instância da IoT, ou seja, a IoT tem suas 'coisas' ligadas à Internet, enquanto NoT trabalha num contexto de rede local ou específica. Dessa forma, deve-se tratar a IoT como um paradigma com várias possibilidades de contexto de utilização.

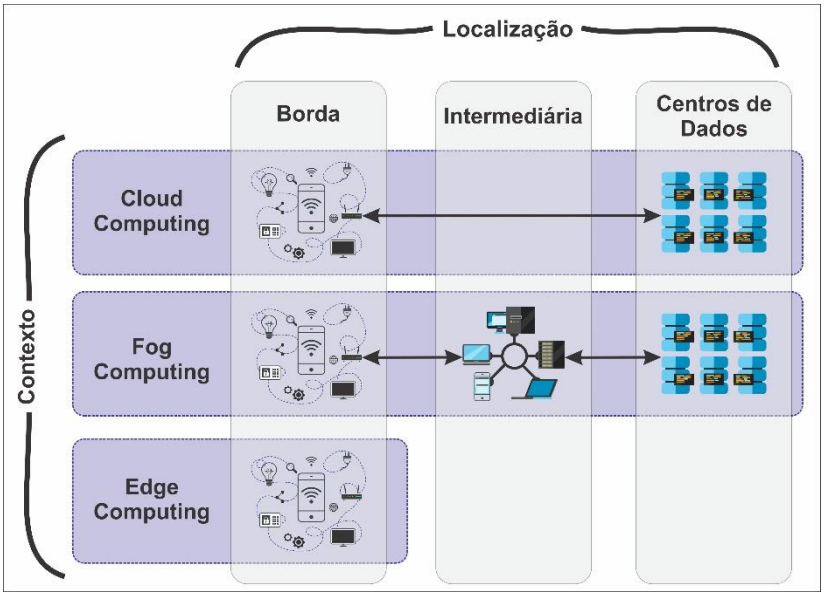
A *Internet of Things* pode ser aplicada a diversos contextos, que são definidos através da infraestrutura e do serviço que serão utilizados. Cada contexto da IoT deve conter ciências subjacentes e fundamentais, e deve envolver a detecção, computação, comunicação e a atuação de determinados equipamentos (VOAS, 2016). De acordo com o trabalho de Gubbi et al. (2013), a infraestrutura básica da IoT foi herdada das Redes de Sensores Sem Fio (RSSF) e define que a estrutura pode conter nós, sensores ou atuadores, nós intermediários e nós com computação em nuvem. Destaca-se que o nó intermediário pode ser o *Gateway* da rede local. O autor pontua ainda que os nós de sensores ou atuadores, em alguns casos, podem ocorrer juntamente com nós intermediários e que, em outros, tal como na computação em nuvem, podem ocorrer em uma infraestrutura de IoT.

Pode-se utilizar para aplicações de IoT três contextos gerais, *Cloud Computing*, *Fog Computing* e *Edge Computing*, sendo que a diferença entre eles está exclusivamente em relação ao contexto da aplicação (MAHMUD; KOTAGIRI; BUYYA, 2018). *Cloud Computing* é um modelo de acesso à rede onipresente e sob demanda de serviço e recursos de computação (MELL; GRANCE, 2011). *Fog Computing* é um modelo com semelhanças à *Cloud Computing*, porém localizado próximo ao usuário e com recursos limitados. Já a *Edge Computing* é um modelo de camada de rede que engloba os dispositivos finais e seus

usuários, para fornecer capacidade de computação local (IORGA et al., 2018).

Na Figura 2 é apresentado cada contexto que está diretamente ligado à sua localização de processamento de dados. Essa localização está subdividida em ‘Borda’, ‘Intermediária’ e “Centros de Dados”. De forma perpendicular à localização, pode-se visualizar as transferências de dados aplicados entre as camadas na IoT, com seus respectivos contextos: *Cloud Computing*, *Fog Computing* e *Edge Computing*.

Figura 2 - Contexto vs. Localização



Fonte: elaborada pelo autor (2019).

2.1.1 Cloud Computing

A *Cloud Computing* ou Computação em Nuvem, foi introduzida por Mell e Grance (2011) como sendo um modelo que possibilitasse o acesso ubíquo, conveniente e sob demanda, através de uma rede, a um conjunto compartilhado de recursos computacionais configuráveis que poderiam ser provisionados e liberados de forma rápida, por meio de uma gestão ou de interação com o provedor de serviço.

O *National Institute of Standards and Technology* (NIST) define que esse modelo de nuvem é composto de cinco características essenciais e quatro modelos, a saber:

a) Autoatendimento sob demanda: um consumidor pode controlar de forma unilateralmente os recursos de computação disponíveis para ele, como o tempo de servidor e armazenamento de rede, e isso automaticamente, sem exigir interação humana com cada provedor de serviços;
b) Amplo acesso à rede: os recursos estão disponíveis na rede e são acessados por meio de mecanismos padrão que promovem o uso por plataformas heterogêneas;
c) Agrupamento de recursos: os recursos de computação do provedor são agrupados para atender a diversos consumidores, com diferentes recursos físicos e virtuais atribuídos. Há um senso de independência de localização em que o cliente geralmente não tem controle ou conhecimento sobre a localização exata dos recursos fornecidos, mas pode ser capaz de especificar a localização em um nível mais alto de abstração (por exemplo, país, estado ou *datacenter*). Exemplos de recursos incluem o armazenamento, processamento, memória e a largura de banda de rede.
d) Rápida elasticidade: os recursos fornecidos através de uma nuvem podem ser provisionados e liberados rapidamente para ajustarem-se à variação da demanda. Como resultado, frequentemente o usuário tem a impressão de que os recursos oferecidos são ilimitados;
e) Serviço medido: sistemas em nuvem controlam e otimizam o uso dos recursos automaticamente, por meio de medições, e podem monitorar e reportar a utilização de recursos de forma transparente aos consumidores e provedores.

O NIST define também três modelos de serviço:

a) Infraestrutura como Serviço (*Infrastructure as a Service* - IaaS): a capacidade oferecida ao consumidor é o fornecimento de processamento, armazenamento, redes e outros recursos fundamentais de computação, nos quais o consumidor pode implantar e executar softwares arbitrários, que podem incluir sistemas operacionais e aplicativos. Portanto, é considerada a camada inferior dos sistemas computacionais em nuvem e

oferece recursos virtualizados de servidores de computação, armazenamento e comunicação sob demanda;

b) Plataforma como serviço (*Platform as a Service* - PaaS): esta camada, com nível de abstração intermediário, oferece um ambiente de desenvolvimento e hospedagem que permite aos usuários criarem aplicações sem preocuparem-se com requisitos de infraestrutura. Portanto, está camada oferece capacidade ao consumidor de implementar uma infraestrutura em nuvem;
c) Software como Serviço (*Software as a Service* - SaaS): é a capacidade oferecida ao consumidor de usar os aplicativos do provedor em execução em uma infraestrutura de nuvem. As aplicações são acessíveis a partir de vários dispositivos clientes e que podem ser acessadas por meio de um navegador da web (por exemplo, um e-mail baseado na web) ou uma *interface* de programa.

Por fim, o NIST define quatro modelos de implantação e acesso:

a) Nuvem privada: é a infraestrutura de nuvem destinada para uso exclusivo por uma única organização, que pode incluir um ou vários consumidores;
b) Nuvem da comunidade: é a infraestrutura de nuvem destinada para uso exclusivo por uma comunidade específica de consumidores de organizações que compartilham preocupações, podendo ser organizações distintas, mas com objetivos ou políticas em comum;
c) Nuvem pública: é modelo de nuvem disponibilizado por organizações empresariais, acadêmicas ou governamentais para o público em geral e que qualquer um pode utilizar;
d) Nuvem híbrida: esta infraestrutura de nuvem é composta por duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem como entidades exclusivas, mas unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados ou/e aplicativos

2.1.2 Fog Computing

Inicialmente, o conceito de *Fog Computing* foi introduzido pela Cisco System, em 2012, como uma extensão do paradigma de *Cloud*

Computing, que, por sua vez, fornece serviços de rede, computação e armazenamento para os dispositivos conectados à *Cloud* (BONOMI et al., 2012). A Cisco introduziu a *Fog Computing* como se fosse uma *Cloud Computing*, porém, localizada em um ambiente mais próximo ao usuário final. Pode-se dizer que a *Fog Computing* complementa todos os serviços da *Cloud Computing*. No entanto, esta é mais heterogênea do que a *Cloud*, por poder coexistir com links de alta velocidade, como também com tecnologias sem fio de diversos tipos com os dispositivos da borda (ROMAN; LOPEZ; MAMBO, 2018).

Em 2018, a NIST (IORGA et al., 2018) definiu que a *Fog Computing* é um modelo de camada de acesso à rede onipresente, que oferece um compartilhamento contínuo de recursos de computação escalonáveis por meio da utilização de redes distribuídas e da baixa latência na comunicação entre o usuário final da borda e a localização intermediária. Trabalhando de forma distribuída, a *Fog Computing* pode ter hierarquia de infraestrutura, na vertical, como também suportar a federação, na horizontal. Como mencionado anteriormente, o modelo de camada minimiza o tempo de resposta de requisições feitas por equipamentos localizados na borda do paradigma IoT. Desse modo, pode-se dizer que a *Fog Computing* não se torna uma mera extensão da computação em nuvem, mas um paradigma próprio (ROMAN; LOPEZ; MAMBO, 2018).

Por fim, pode-se dizer que o paradigma *Fog Computing* tem algumas propriedades proeminentes, como as aplicações em uma arquitetura multicamada, na qual separa e mescla as funções de hardware e software (IORGA et al., 2018), como também predomina o acesso sem fio, uma grande escala de nós, a heterogeneidade e o ambiente geograficamente distribuído (YI et al., 2015).

2.1.3 Edge Computing

A *Edge Computing* é uma camada de rede que vincula os dispositivos finais aos seus usuários, e que tem o objetivo de fornecer capacidade de computação local para um sensor, atuador ou outro dispositivo da rede local (IORGA et al., 2018) (SATYANARAYANAN, 2017). Essa camada executa aplicativos específicos em algum local específico, podendo ter sua conexão restrita entre a aplicação e a Internet.

Basicamente, a *Edge Computing* é formada por uma NoT, uma *Mobile Ad hoc Networks* (MANET) ou qualquer outra rede local, na qual, sua conexão constitui-se de uma conexão local entre a base de

processamento e os sensores/atuadores (IORGA et al., 2018). A *Edge Computing* é aplicada principalmente em dispositivos móveis, tais como *smartphones* e *tablets*, tendo sua denominação como *Mobile Edge Computing* (MEC). Para o armazenamento de dados em *Edge Computing*, utilizam-se as *Cloudlets*, que são nuvens de pequena escala com mobilidade aprimorada que fazem o armazenamento de dados na borda da Internet (SATYANARAYANAN, 2017) (MAHMUD; KOTAGIRI; BUYYA, 2018). Desse modo, as *Cloudlets* foram projetadas para estenderem serviços baseados em nuvem, porém, em dispositivos de baixo poder de processamento e próximos aos dispositivos da borda.

Por fim, a *Edge Computing* tem algumas características em comum com a *Cloud e Fog Computing*, porém caracteriza-se principalmente por ser aplicável a ambientes restritos. Um exemplo que a *Edge Computing* herda são os bancos de dados, que são armazenados uniformemente em apenas um local seguro. Porém, uma característica peculiar é a sua aplicabilidade na borda da Internet ou em uma rede restrita local.

2.2 PROTOCOLOS

A rede Internet é desenvolvida a partir da integração de protocolos, na qual é possível integrar o protocolo de rede IP (*Internet Protocol*) com o protocolo de transporte TCP (*Transmission Control Protocol*) ou com o UDP (*User Datagram Protocol*). A Internet também pode usar os protocolos de aplicação, como o HTTP (*HyperText Transfer Protocol*) e também implementar o protocolo TLS, para o transporte de dados com segurança. Esses protocolos podem ser, respectivamente, da camada de Enlace, Internet ou Aplicação, e também podem ser encontrados em uma conexão comum entre dispositivos computacionais (KUROSE; ROSS, 2006).

Em relação a dispositivos IoT, diferentes protocolos e soluções de segurança são propostos. Alguns desses dispositivos têm *hardware* limitado e que necessitam de protocolos otimizados. Levando em consideração esses aspectos, os dispositivos podem ter conexões com a Internet e/ou com seus vizinhos de borda, utilizando para a comunicação o protocolo de rede 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Networks*) juntamente com o protocolo de aplicação CoAP (*Constrained Application Protocol*) (BRACHMANN, 2012). Em oturo aspecto, o CoAP pode ser integrado ao DTLS (*Datagram Transport Layer Security*) ou ao IPSec (*Internet Protocol Security*),

como também utilizar o protocolo RPL (*Routing Protocol for Low-power and Lossy Networks*) para o roteamento desses dispositivos finais (GRANJAL; MONTEIRO; SILVA, 2015). Outra opção para utilização de protocolos leves para a camada de aplicação é o protocolo AMQP (*Advanced Message Queuing Protocol*) e o MQTT (*Message Queue Telemetry Transport*), que são baseados no protocolo de transporte TCP/IP.

A biblioteca gerada ao final desse trabalho está baseada no protocolo MQTT em conjunto com o protocolo TLS/SSL aplicado na camada do protocolo UDP e o IPv4/IPv6. Nesse sentido, a seguir é apresentado o protocolo MQTT e na sequência o protocolo TLS/SSL.

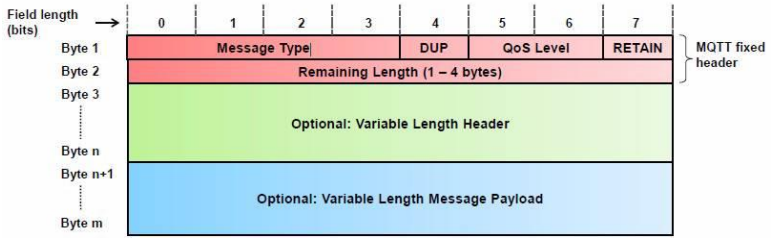
2.2.1 MQTT

O protocolo MQTT (*Message Queue Telemetry Transport*), criado na década de 1990, teve o propósito de ser um simples e otimizado sistema de transferência de dados, podendo assim ser aplicado em rede IoT (OASIS, 2014). Além dessas características, é um protocolo que consome pouca largura de banda na transferência de dados e utiliza o padrão cliente-servidor para a estrutura proposta, porém, com uma peculiaridade de transmissão entre os nós.

A transmissão adotada é por meio de publicação e inscrição, dessa maneira, pode haver um ou inúmeros nós publicando para uma central, chamado de *Broker*, e um ou inúmeros nós inscritos que receberão dados dessa central.

Conforme supracitado, os pacotes de transferência de dados consomem pouco, ou seja, um tamanho fixo de 2 bytes de *header*, e o restante do tamanho do pacote é utilizado para transferência de conteúdo. Os autores Martins e Zem (2015) explanam, conforme apresentado na Figura 3, acerca do cabeçalho das mensagens que transitam com o protocolo MQTT.

Figura 3 - Cabeçalho do MQTT

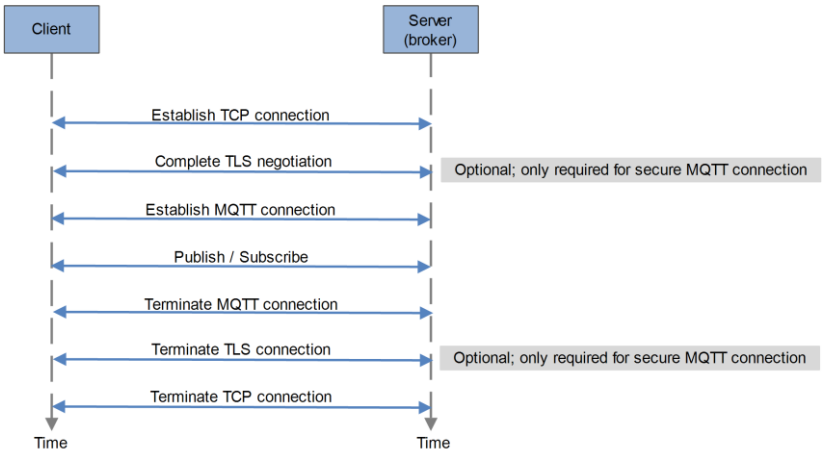


Fonte: Perez (2017).

Portanto, segundo Martins e Zem (2015), o cabeçalho fixo das mensagens MQTT é composto por parâmetros que indicam o tipo da mensagem (*Message Type*), o indicador de mensagem duplicada (DUP), marcador de qualidade do serviço (QoS *level*), o marcador de retenção (RETAIN) e a largura restante representando o espaço reservado ao restante da mensagem (*Remaining Lenght*), contendo o cabeçalho variável e o conteúdo da mensagem, chamado de *payload*. No entanto, em alguns casos é utilizada uma extensão do *header*.

Na Figura 4 é apresentada a estrutura e a sequência de trocas de todas as camadas entre o cliente e o servidor (*Broker*), que se inicia com o estabelecimento de uma conexão TCP, em seguida vem a negociação de TLS, e posteriormente o estabelecimento de conexão por meio do protocolo MQTT. Após todas as trocas de mensagem de estabelecimento e concordância, o cliente ou servidor pode publicar ou se inscrever a determinado *Broker*. Por fim, terminam-se todas as conexões de ordem inversa a estabelecidas anteriormente.

Figura 4 - Sequências de trocas de mensagens



Fonte: Koneski et al. (2018).

2.2.2 TLS/SSL

O protocolo TLS/SSL (*Transport Layer Security / Security Socket Layer*) é amplamente utilizado por diversos aplicativos atualmente, que

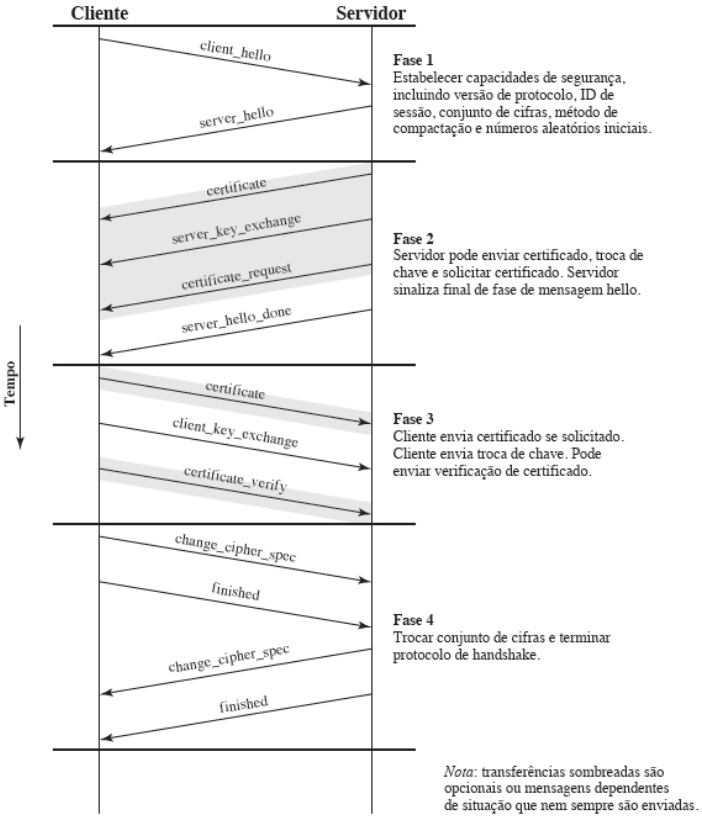
necessitam de uma camada de segurança na aplicação final, pois o principal objetivo do protocolo TLS é fornecer sigilo e integridade de dados entre dois aplicativos de comunicação. O protocolo é composto de duas camadas: o protocolo de registro TLS e o protocolo TLS de *handshake* (DIERKS; RESCORLA, 2008).

O protocolo TLS trabalha em cima de alguma camada de protocolo de transporte confiável (por exemplo, TCP), assim, o protocolo TCP é o protocolo de registro do TLS. Por fim, o protocolo TCP é usado para o encapsulamento de vários protocolos que estão localizados acima do nível atual (IBM, 2018).

O protocolo de *handshake* do SSL/TLS, que pode ser visto na Figura 5, permite que o servidor se autentique ao cliente e o cliente se autentique no servidor, além de negociar um algoritmo de criptografia e chaves criptográficas, antes que o protocolo de aplicação transmita ou receba seu primeiro byte de dados (DIERKS; RESCORLA, 2008).

Como mencionado anteriormente, o *handshake* SSL ou o TLS permitem que o cliente e o servidor SSL ou TLS estabeleçam a chave secreta com a qual eles se comunicam (IBM, 2018). Dessa forma, deve-se levar em consideração que ocorrerão diversas trocas de informações entre as partes, como também, na opção de autenticação mútua, deve de ocorre a troca de certificados de ambas as partes (cliente e servidor). As trocas de informações ocorrem em 13 etapas quando utilizado a autenticação mútua, conforme se pode ver na Figura 5.

Figura 5 - TLS/SSL *handshake*



Fonte: Stallings (2015).

Conforme Stallings (2015); Willeke (2018); Dierks e Rescorla (2008); e IBM (2018), cada etapa do processo de transferência é muito importante para manter a integridade dos dados após o *handshake*. Nesse sentido, a seguir cada etapa é explanada de forma sucinta:

1) O cliente SSL ou TLS envia uma mensagem *cliente_hello* que lista informações criptográficas, como a versão de SSL ou TLS e, na ordem de preferência do cliente, os *CipherSuites* suportados pelo cliente;

2) O servidor SSL ou TLS responde com uma mensagem *server_hello* que contém o *CipherSuite* escolhido pelo servidor a partir da lista fornecida pelo cliente;
3) O servidor SSL ou TLS responde, ainda, com o certificado que pertence a ele, *certificate*;
4) O servidor SSL ou TLS envia o pacote *server_key_exchange* com parâmetros de concordâncias em relação a chave trocada;
5) O servidor SSL ou TLS envia o pacote *certificate_request* requerendo o certificado do cliente;
6) O servidor SSL ou TLS envia um marcador de fim de envio de dados para o cliente com intenção de sinalizar o final da fase de mensagem hello, por meio do *server_hello_done*;
7) O cliente envia para o servidor também seu certificado por meio da mensagem *certificate*;
8) O cliente envia a mensagem *client_key_exchange* que apresenta seus parâmetros de concordâncias em relação a chave trocada;
9) O cliente envia o pacote *certificate_verify* para o servidor, com objetivo de garantia da integridade do certificado;
10) O cliente envia uma mensagem *change_cipher_spec*, que tem o propósito de marcar o ponto no qual o cliente alterna para a cifragem com as chaves de criptografia recém-negociadas;
11) O cliente SSL ou TLS envia um *finished* para verificar se a outra parte está entendendo sua comunicação corretamente por meio da cifragem;
12) O servidor envia também uma mensagem *change_cipher_spec*, marcando assim o início da cifragem dos dados;
13) Do mesmo modo, o servidor SSL ou TLS envia ao cliente uma mensagem *finished*, que é criptografada com a chave secreta, verificando se o *handshake* com as concordâncias de chaves ocorreu corretamente.

2.3 AUTENTICAÇÃO

O termo ‘autenticação’ descreve o processo de verificação da identidade de uma pessoa ou entidade (MOHAMMED; ELSADIG, 2013). Nesse sentido, a seguir é apresentada a diferença básica entre a autenticação simples e a mútua. Será mostrada a autenticação das partes durante o *handshake* de uma comunicação. No entanto, o trabalho está vinculado com a autenticação mútua.

2.3.1 Autenticação simples

A autenticação simples permite que apenas uma das partes conheça a identidade da outra, sendo assim, ela pode ser chamada também de autenticação de uma via. Isto pode ocorrer quando apenas uma das partes tem interesse em identificar a outra parte, solicitando, assim, à outra parte os dados pertinentes à sua identificação.

O principal meio de autenticação simples ou de uma via é o fornecimento de usuário e senha (MOHAMMED; ELSADIG, 2013), em que a camada de controle de segurança encontra-se juntamente com a camada de aplicação. Uma das formas de autenticação do protocolo SSL/TLS também utiliza a de via única, na qual apenas o servidor precisa enviar o certificado digital (SMITH, 2001).

Diversos autores, tais como Needham e Schroeder (1978); e Lamport (1981) mencionam objetivos e métodos que seriam de uma autenticação de uma via, no geral. Em 1994, com a expansão do uso do HTTP, a empresa Netscape desenvolveu a primeira versão do SSL, que logo após foi substituído pelo SSLv2. Outro destaque ocorreu em outubro de 1997, quando Myers (1997) e um grupo de trabalho de redes, desenvolveram um método para adicionar suporte de autenticação simples a protocolos baseados em conexão, sendo que o método insere uma camada de autenticação entre o protocolo e a conexão.

2.3.2 Autenticação mútua

A autenticação mútua pode ser definida como um processo que autentica duas partes, em que comumente uma é o cliente e a outra é o servidor, desse modo, uma autentica a outra. A autenticação mútua também é conhecida como autenticação bidirecional, ou de duas vias (KOTHMAYR et al., 2012) (OTWAY; REES, 1987).

Com a autenticação mútua, uma conexão pode ocorrer somente quando o cliente confiar no certificado digital ou em outra forma de identificação do servidor e o servidor confiar no cliente. Um exemplo claro da troca de certificados e da autenticação mútua acontece por meio do *handshake* do protocolo TLS (ROUSE, 2008).

Quando utilizada a autenticação por meio de certificado, o cliente não está enviando um nome de usuário ou senha para o servidor, podendo ajudar indiretamente a impedir ataques de log de *phishing*, *keystroke* ou *man-in-the-middle* (MITM). Por fim, o certificado X.509 é

considerado o padrão oficial para certificados de chave pública, e o SSL/TLS é baseado nesse padrão.

Para ilustrar o funcionamento da autenticação mútua, a Figura 6, mostra dois dispositivos, sendo que os dois, de forma simultânea, solicitam a identificação do opositor.

Figura 6 - Autenticação mútua

Fonte: elaborada pelo autor (2019).

A autenticação mútua não deve ser confundida com a autenticação de dois fatores, que é um processo de segurança em que o cliente fornece dois meios de identificação para o servidor (ROUSE, 2008). Veremos em seguida alguns exemplos de fatores de autenticação para melhor ilustração.

2.3.3 Fatores de autenticação

Os fatores são os métodos usados para autenticação. Vários são os fatores que podem ser usados para realizar a autenticação: conhecimento de algo (por exemplo a senha), posse de algo (por exemplo um cartão físico), característica biométrica (por exemplo a impressão digital de uma pessoa) e contexto (por exemplo o endereço IP da entidade ou o local onde a entidade se encontra) (STALLINGS, 2015).

Segundo Mohammed e Elsadig (2013), a autenticação pode ser dividida em dois esquemas: simples fator e múltiplos fatores de autenticação.

O simples fator de autenticação depende de apenas um fator, e, geralmente, a autenticação é baseada em nome de usuário e senha. Em alguns casos ocorre a utilização única de um *smartcard*, um *token* ou um PIN.

A autenticação de múltiplos fatores é baseada em dois ou mais fatores de autenticação. Por exemplo, usar algo que se sabe (senha) e algo que se tem (número fornecido por um token físico). Poderia ser também usado um contexto (endereço IP) em conjunto com um algo que se tem (certificado digital). Na autenticação multifator é importante combinar fatores de categorias diferentes, como por exemplo: conhecimento de algo combinado com posse de algo; posse de algo combinado com característica biométrica e assim por diante (ESINER e DATTA, 2019).

Outro método de utilização dos fatores de autenticação é criar funções ou mecanismos que podem ser utilizados durante o *handshake* sem a necessidade de haver alguma entrada do usuário. Um exemplo disso é a geração de um desafio-resposta, que Backes et al. (2007) mencionam como sendo uma abstração para o usuário final. No desafio-resposta também é possível adquirir informações, por exemplo, obter a velocidade de processamento do outro sistema por meio da complexidade do cálculo solicitado como desafio.

3 TRABALHOS RELACIONADOS

A revisão sistemática foi executada com foco de levantamento bibliográfico nas seguintes bases de referências:

- Sítio de periódicos da CAPES;
- IEEE Xplore;
- Science Direct; e
- Google Scholar.

Para a revisão foi utilizado a seguinte string de busca: “Mutual Authentication AND Lightweight AND optimized AND (Fog Computing OR Edge Computing OR Internet of Things)” restringindo artigos dos últimos 9 anos. Resultando em torno de 3.000 artigos para análise inicial. Após primeira análise dos títulos chega-se em 450 artigos. Por fim, chega-se ao conjunto de 10 artigos apresentados a seguir com uma série de restrições, como por exemplo a qualificação do local de publicação.

O Quadro 1 refere-se a uma sumarização de pontos importantes para a presente pesquisa. Inicialmente foram elencados os trabalhos anteriores que continham algum tipo de autenticação, e eles foram subdivido nos trabalhos que continham a autenticação de uma via (simples) e nos que a autenticação era de duas vias (mútua). Posteriormente, desses trabalhos foram elencados os trabalhos que eram aplicados a algum contexto de IoT, como *Edge*, *Fog* ou *Cloud Computing* (IoT), juntamente a esses foram definidos os trabalhos que haviam alguma otimização ou que traziam redução de consumo de espaço (leve). Por fim, foram apontados os trabalhos que tinham algum propósito de autenticação com fatores externos, como por exemplo, o tempo de rede e tempo de processamento (fatores), chegando-se, desse modo, ao Quadro 1 apresentado. Os pontos apresentados foram adquiridos a partir de um estudo do estado da arte.

Através dessa pesquisa sistemática chega-se aos trabalhos dos autores Amin et al. (2018), Esiner e Datta (2019), Gope et al. (2017), Grüdtner et al. (2018), Ibrahim (2016), Jan et al. (2017), Kothmayr et al. (2012), Kumar e Gandhi (2017), Li, Liu e Nepal (2017), Piramuthu e Doss (2017), Tewari e Gupta (2017) e Wu et al. (2018) que de forma geral abordaram métodos de autenticação para a proteção de dados transferidos em rede. No entanto, somente alguns artigos trabalharam com dados sensíveis a um contexto específico. Por exemplo, temas como *healthcare* são tratados pelos autores Kumar e Gandhi (2017) e Wu et al. (2018). Temas envolvendo *smart city* são tratados pelos autores Gope et al. (2017) e Jan et al. (2017). De modo geral, a

tecnologia mais abrangível em termos de autenticação é a utilização do RFID (*Radio-Frequency IDentification*), conforme pode ser observado nos trabalhos dos autores Gope et al. (2017), Piramuthu e Doss (2017) e Tewari e Gupta (2017).

Estratégias de autenticação mútua não são novidades perante os pesquisadores. Alguns autores como Gope et al. (2017), Ibrahim (2016), Jan et al. (2017), Li, Liu e Nepal (2017), Tewari e Gupta (2017) e Wu et al. (2018) trazem comentários de que a autenticação mútua garante a identificação de todas as partes envolvidas. No entanto, os artigos de Gope et al. (2017), Ibrahim (2016) e Wu et al. (2018) não aplicam de forma otimizada a autenticação, simplesmente focam em uma autenticação que tenha mais poder de segurança. O artigo de Gope et al. (2017) apresenta uma arquitetura de autenticação baseada em RFID. Ainda a proposta do Gope et al. (2017) introduz um novo modelo de autenticação que tem base na leitura de *tags*, no registro em um concentrador e que, por meio desse, oferece as funcionalidades de autenticação mútua, anonimato, disponibilidade, escalabilidade e segurança de localização.

Na sequência, o autor Wu et al. (2018) apresenta um novo esquema de autenticação de dois fatores aplicado ao GLOMONETs (*Global Mobility Networks*). Por meio de prova formal, ele demonstra que o atacante não pode quebrar a chave de sessão e a privacidade do esquema com uma probabilidade maior do que adivinhar diretamente a senha. Já no artigo de Ibrahim (2016), tem-se a explanação de um modelo de autenticação mútua em uma arquitetura *Edge-Fog-Cloud*, que requer um usuário (dispositivo) mestre que armazene uma chave secreta para cada novo dispositivo da rede e, além dele ter acesso a todos os dispositivos *Fogs* da rede, o requerente pode se comunicar de forma segura e autenticar-se mutuamente com qualquer dispositivo da rede. O modelo é semelhante a uma terceira parte confiável para autenticar-se mutuamente.

Os autores Jan et al. (2017), Li, Liu e Nepal (2017) e Tewari e Gupta (2017) apresentam em seus artigos a autenticação mútua otimizada (em relação ao tamanho em KB de implementação) com diversas aplicabilidades finais. Entre eles, os autores Tewari e Gupta (2017) apresentam uma autenticação mútua ultraleve, que utiliza apenas operações bit-a-bit, resultando, assim, em uma sobrecarga computacional baixa. Por seguinte, os autores Li, Liu e Nepal (2017) levam em consideração o desempenho e eficiência do protocolo como principal característica, visto que ele será utilizado em aplicações com poucos recursos. Pensando nisso, foi desenvolvido um algoritmo de

criptografia de chave pública, para posteriormente ser utilizado na autenticação mútua leve/otimizada. Os autores Jan et al. (2017) desenvolveram um método de autenticação de quatro vias. Esse esquema é baseado em *payload*, que usa um mecanismo simples no *handshake* para verificar a identidade das partes. O modelo utiliza os recursos do CoAP para a comunicação dos pacotes e usa o algoritmo de cifragem AES com chave 128 bits para a sessão.

Num contraponto aos trabalhos citados anteriormente, os artigos de Amin et al. (2018), Esiner e Datta (2019), e Piramuthu e Doss (2017) indicam que se pode aplicar de forma simples a autenticação entre as partes e mesmo assim isso ser de grande validade para a pesquisa sucessora. Exemplo disso é o artigo de Amin et al. (2018), que implementa a autenticação simples e otimizada. Esse artigo propôs uma arquitetura aplicável ao ambiente de nuvem, que aplica um novo modelo proposto de autenticação por meio de *smartcards*, além de utilizar a ferramenta AVISPA e o modelo lógico BAN. De forma semelhante, só que sem ser um protocolo otimizado/leve, há o artigo de Piramuthu e Doss (2017), que propôs métodos de autenticação de alto desempenho que devem ser considerados. O artigo introduz fatores para a autenticação, como ocorre no trabalho de Esiner e Datta (2019), que menciona que pode ser utilizada a localização, a distância e a temperatura da *tag* como um fator, ou simplesmente o tempo para se autenticar, algo que varia para cada dispositivo.

O artigo de Grüdtner et al. (2018) é um estudo inicial sobre a autenticação mútua aplicado a pontos gerais de *Fog Computing*, no qual se pode constatar um fator de autenticação, a função desafio-resposta, que é utilizado com adaptações para a presente pesquisa.

Dando continuidade, os artigos de Kumar e Gandhi (2017) e de Kothmayr et al. (2012) utilizam o DTLS para a autenticação e incluem ainda outros pilares da segurança no mesmo mecanismo. Esse modelo proposto por Kumar e Gandhi (2017) utiliza datagramas, que podem ser obtidos através do tráfego dos dados em uma rede. Posteriormente, é analisada e obtida a autenticidade na camada de transporte. Seu protocolo consiste em uma melhoria do DTLS com base no protocolo CoAP, só que trabalha somente com autenticação simples, pois seu objetivo é de amenizar os ataques DDoS (*Distributed Denial of Service*). No entanto, em trabalho anterior, Kothmayr et al. (2012), os autores aplicaram diversas técnicas para obterem um protocolo de troca de chaves e autenticação mútua no DTLS.

De modo geral, os trabalhos citados deixam em aberto a aplicação das estratégias de desafio-resposta, tempo de resposta e

identificação com freshness (nonce) em conjunto com os criptossistemas. Pensando nisso, em resumo, a estratégia de troca de chaves e de cifras abre a possibilidade de implementação de um método de autenticação mútua, que utiliza as estratégias de função desafio-resposta, de tempo de resposta e de *nonce* para autenticação. Assim, os autores supracitados deixam em aberto a possibilidade de implementar conjuntos de fatores específicos para o contexto de *Fog Computing*. Por fim, no Quadro 1 pode-se visualizar, de modo geral, a comparação dos trabalhos anteriores com o presente trabalho.

Quadro 1 - Trabalhos anteriores vs. Presente trabalho

Trabalho	Simples	Mútua	Leve	IoT	Fatores
Amin et al. (2018)	■		■	■	
Gope et al. (2017)		■		■	
Esiner e Datta (2019)	■			■	Tempo de resposta
Grüdtner et al. (2018)		■		■	Desafio-Resposta
Ibrahim (2016)		■		■	
Jan et al. (2017)		■	■	■	
Kothmayr et al. (2012)		■		■	
Kumar e Gandhi (2017)	■			■	
Li, Liu e Nepal (2017)		■	■	■	
Piramuthu e Doss (2017)	■				Nonce
Tewari e Gupta (2017)		■	■	■	
Wu et al. (2018)		■		■	
Este trabalho		■	■	■	Desafio-Resposta, Tempo de resposta e Nonce

Fonte: elaborado pelo autor (2019).

4 PROTOCOLO DE AUTENTICAÇÃO MÚTUA COM FATORES

A seguir são apresentados aspectos referentes à proposta de pesquisa, que engloba a delimitação do escopo de pesquisa e a definição da proposta, finalizando o capítulo com a explanação com mais detalhes acerca da proposta.

4.1 ESCOPO DO PROTOCOLO

A presente pesquisa tem como finalidade a modelagem de uma autenticação que seja aplicada em dispositivos restritos de memória. Dessa forma, o escopo de aplicação da autenticação estará voltado para um ambiente com dispositivos restritos de memória e o modelo irá usar os seguintes métodos de criptografia: AES (*Advanced Encryption Standard*), RSA (R. Rivest, A. Shamir e L. Adleman) e Diffie-Hellman. Como o presente trabalho foi baseado no protocolo do TLS, foram escolhidos o AES, o RSA e o Diffie-Hellman como criptossistemas iniciais do protocolo, porém, em trabalhos futuros podem ser usados outros protocolos. Uma tarefa importante neste trabalho é adequar os mecanismos criptográficos escolhidos para que possam compor um método de autenticação leve/otimizado.

4.2 DEFINIÇÃO DO PROTOCOLO

De modo geral, a solução proposta tem características semelhantes ao *handshake* do protocolo SSL/TLS, porém com algumas peculiaridades referentes à sua autenticação mútua e seus fatores de autenticação. Inicialmente, o protocolo pode ser dividido em duas etapas, como pode-se ver na Figura 7, utilizando a cifragem dos dados de forma simétrica e segura após toda a verificação e o *handshake*. Na primeira característica, ocorre o *handshake* em três etapas e na segunda ocorre a autenticação mútua com três fatores.

Na vertical esquerda da Figura 7 pode-se visualizar primeira característica, o *handshake* em que ocorrem três etapas. A primeira etapa é o reconhecimento das partes por meio de mensagem de identificação, posteriormente, na segunda etapa, ocorre um conjunto de trocas, que tem o objetivo de autenticar mutuamente e trocar as chaves assimétricas das partes. Na terceira etapa ocorre um conjunto de concordâncias, que tem o objetivo de estabelecer uma chave de sessão ou chave secreta. Pode-se também visualizar a divisão das etapas na

Figura 8, de tal forma que a primeira etapa é simbolizada pela cor laranja, a segunda pela cor verde e a terceira pela cor amarela.

Na segunda característica da proposta ocorrem três fatores de autenticação. O primeiro fator é a função desafio-resposta, que tem o objetivo de garantir que permaneça segura a chave pública do emissor. O segundo fator é a análise do tempo de resposta por mensagem enviada, e tem por objetivo amenizar os ataques *man-in-the-middle* na comunicação. E por último, o terceiro fator é a inserção de um novo *nonce* no pacote a cada envio no passo de *handshake*, que tem por objetivo manter as mensagens sincronizadas juntamente com o *freshness*, e também identificadas pelas partes comunicantes, o que, de forma complementar, ajuda na análise dos fatores anteriores.

Figura 7 - Autenticação mútua com fatores

Fonte: elaborada pelo autor (2019).

4.2.1 Handshake

O *handshake* das partes acontece a partir de trocas de mensagens entre elas, tendo o objetivo de trocar chaves, cifras e informações para uma troca de dados seguros da aplicação final. A seguir é explanado o funcionamento de cada etapa do *handshake*, podendo-se visualizar os passos na Figura 8, juntamente com a notação apresentada no Quadro 2.

Quadro 2 - Notação

Syn	Pedido inicial de sincronização para a autenticação
Ack	Mensagem de entendido
n^x	*Nonce* da parte X
tp^x	Tempo gasto para cálculo e construção do pacote - tempo de processamento
tr^x	Tempo de rede - tráfego
K^x_{pub}	Chave pública da parte X
K^x_{priv}	Chave privada da parte X
$\}K^x_{Sign}$	Assinatura dos parâmetros utilizando a chave privada da parte X
$\}K^x_{pub}$	Cifragem dos parâmetros utilizando a chave pública da parte X
Fdr^x	Função desafio-resposta da parte X
$F(K^x_{pub})$	Resposta da Função desafio-resposta de X aplicado ao valor da chave pública enviada da parte X
DH^x	Valor resultante do cálculo do *Diffie-Hellmann* da parte X
g	Valor da base do cálculo do *Diffie-Hellman*n
p	Valor do módulo do *Diffie-Hellmann*
$\}DH_K$	Cifragem dos parâmetros utilizando a chave secreta compartilhada entre as partes
ctr	Contador de tempo de rede
ctp	Contador de tempo de processamento
time	Valor do tempo local obtido no momento da captura
ID_{orig}	Valor de identificação da parte de origem do pacote
ID_{dest}	Valor de identificação da parte de destino do pacote
rad	Valor randômico para identificar o pacote.
x	É considerado de forma resumida a PARTE A ou B

Fonte: elaborado pelo autor (2019).

Antes da explanação de cada passo do *handshake* apresentado na Figura 8, são apresentados processos que são padrões e utilizadas em diversos passos.

O **valor *nonce*** (n^A ou n^B) para todas as transações que são necessários é obtido a partir da Função 1:

$$n^X = HASH(time \mid ID_{dest} \mid ID_{orig} \mid rad) \quad (1)$$

Processo de finalização: termina a conexão e, caso a PARTE A finalize, inicia-se outro pedido de sincronização no primeiro passo.

Processo de análise do *nonce*: se o valor *nonce* da PARTE (n^X) for igual ao *nonce* X gerado no passo anterior, continua a autenticação, caso não for, termina o presente passo e inicia-se o processo de finalização.

Processo de análise da assinatura dos parâmetros: após a decifragem do *hash* cifrado com a chave pública do emissor, verifica-se se o valor de *hash* é igual ao cálculo do *hash* dos parâmetros iniciais, caso seja, continua a autenticação, caso não, termina o presente passo e inicia-se o processo de finalização.

Processo de análise da função desafio-resposta: se o valor resultante da função (Fdr^x) aplicado à chave pública (K^x_{pub}) for igual à resposta da função ($F(K^x_{pub})$), continua a autenticação, caso não seja, termina o presente passo e inicia-se o processo de finalização.

A seguir, para melhor entendimento de cada etapa, durante a montagem do pacote e de recebimento, é explanada de forma individual a PARTE A (cliente) e a PARTE B (servidor). Deve-se levar em consideração que no servidor deve de haver um processamento igual ou superior do que no cliente.

4.2.1.1 PARTE A - cliente

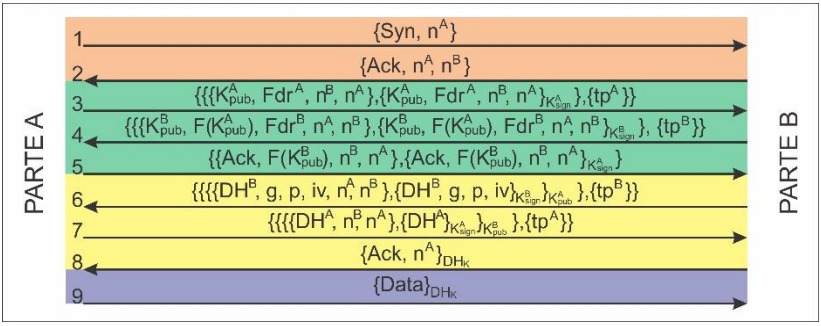
Primeiro passo: inicia-se com a PARTE A calculando o valor de *nonce* (n^A) a partir da Função 1 - o valor resultante é a partir da entrada do valor de tempo, identificador de destino, identificador de origem e um valor randômico que é criado aleatoriamente. Posteriormente, a PARTE A concatena em um pacote a *Flag Syn* ativada - utilizando um valor booleano verdadeiro para representar a *Flag Syn* - e o *nonce* anteriormente gerado. Antes de enviar ao destinatário a PARTE A, inicia-se um contador de tempo de rede (ctr) - que auxilia no monitoramento do tempo de rede e de processamento do destinatário - na sequência é encaminhado o pacote montado para a PARTE B e passa-se para o segundo passo.

Segundo passo: inicia-se monitorando se o tempo de recebimento não extrapola 5 segundos, caso ultrapasse, termina este passo e inicia-se o processo de finalização, caso não, continua a autenticação neste passo. Ao receber o pacote da PARTE B, termina o contador de tempo de rede (ctr), calcula-se o valor do tempo de rede

(tr^A) e inicia-se o contador de tempo de processamento (ctp). Posteriormente, a PARTE A abre o pacote recebido e entra no processo de análise de *nonce* A (n^A) com o recebido e o armazenado. Sequencialmente, a PARTE A armazena o tempo do contador de rede e do *nonce* de B (n^B) recebidos e passa-se para o terceiro passo.

Terceiro passo: nesta etapa inicia-se a troca de conjuntos de parâmetros da autenticação e de chaves assimétricas das partes. Inicialmente, a PARTE A gera um par de chaves assimétricas com a ajuda do algoritmo RSA - para efeito de otimização de memória e desempenho não é utilizado certificado digital, somente números nas chaves - sendo elas uma pública (K^A_{pub}) e uma privada (K^A_{priv}). Em complemento, é gerado um novo valor de *nonce* (n^A) baseado na descrição da Função 1. Além da criação de uma função desafio-resposta (Fdr^A), que é definida a partir de uma combinação de função básica de matemática (soma, subtração ou multiplicação) e um valor aleatório gerado. Ocorre também a assinatura com o algoritmo RSA dos parâmetros gerados anteriormente e o *nonce* B (n^B) recebido no passo anterior. Então é calculado o valor de tempo de processamento (tp^A) a partir do termino do contador de tempo de processamento (ctp). Por fim, um pacote é montado, contendo: a chave pública da PART A (K^A_{pub}); uma função de desafio-resposta (Fdr^A); o valor de *nonce* (n^B) que no passo anterior foi recebido da outra parte; o valor de *nonce* (n^A) gerado; a assinatura de todo conjunto de parâmetros anteriores; e o tempo de processamento (tp^A). Na sequência, inicia-se um contador de tempo total (ctt) e é enviado o pacote com dados para a PARTE B. Por fim, passa-se para o quarto passo.

Figura 8 - *Handshake*



Fonte: elaborada pelo autor (2019).

Quarto passo: inicia-se monitorando se o contador de tempo total (ctt) não extrapola a soma do tempo de processamento (tp^A) com o tempo de rede (tr^B), e com tolerância de 10% de acréscimo no tempo total ($tp^A + tr^B + ((tp^A + tr^B) * 0,1)$) (esse valor foi pensado e testado em laboratório para a estabilização exclusivamente com a *Fog Computing*), caso passe, termina este passo e inicia-se o processo de finalização, caso não, monitora-se o recebimento do pacote da PARTE B. Ao receber o pacote, termina o contador de tempo total (ctt). A PARTE A abre o pacote recebido e entra no processo de análise da assinatura dos parâmetros, depois entra no processo de análise de *nonce* A com o n^A recebido e armazenado, e, após isso, ocorre o processo de análise da função desafio-resposta. Sequencialmente, caso passe por todas as análises, a PARTE A armazena o *nonce* de B (n^B) recebido, a função desafio-resposta de B (Fdr^B), a chave pública de B (K^B_{pub}) e, por fim, passa para o quinto passo.

Quinto passo: com o objetivo de responder à parte em que todos os parâmetros foram armazenados e entendidos, este passo envia um pacote contendo algumas informações de resposta. Antes do envio, a PARTE A calcula o valor resultante da função desafio-resposta ($F(K^B_{pub})$) e gera um novo valor para o *nonce* (n^A), que tem o intuito de vincular esta mensagem com a mensagem sucessora. Sequencialmente, monta-se um pacote contendo: a *flag* com valor falso que representa o entendido (Ack); a resposta do desafio-resposta ($F(K^B_{pub})$); o valor de *nonce* recebido no passo anterior (n^B); e o novo valor gerado de *nonce* (n^A). Após isso, ocorre a assinatura do pacote montado com a chave privada do remetente. Por fim, é enviado de forma concatenada o pacote montado com a assinatura do pacote para a PARTE B e inicia-se o contador de tempo total (ctt). Termina então esta fase de troca de chaves e passa-se para o sexto passo.

Sexto passo: a partir deste passo ocorre a fase de concordância de chave simétrica. Porém, antes de ocorrer isso, deve-se monitorar se o contador de tempo total (ctt) não extrapola 5 segundos, caso ultrapasse este tempo, termina este passo e inicia-se o processo de finalização, caso não, continua a concordância de chave neste passo. Ao receber o pacote da PARTE B, termina o contador de tempo total (ctt) e inicia-se o contador de tempo de processamento (ctp). A PARTE A faz a decifragem do pacote recebido com a sua chave privada (K^A_{priv}) e entra no processo de análise da assinatura dos parâmetros, depois entra no processo de análise de *nonce* A, com o *nonce* (n^A) recebido e armazenado. Somente após verificar a integridade e autenticidade do pacote, a parte armazena o *nonce* de B (n^B) e o valor resultante de B

(DH^B), juntamente com o módulo (p) e a base (g) do *Diffie-Hellman* gerado na PARTE B. Por fim, passa-se para o sétimo passo.

Sétimo passo: a PARTE A realiza o cálculo dos valores de *Diffie-Hellman* a partir das informações provenientes da PARTE B no passo anterior, gerando o valor resultante (DH^A) e o valor para a chave de sessão (DH_K). A PARTE A ainda calcula um novo valor de *nonce* (n^A) por meio da Função 1. Sequencialmente, é aplicada uma assinatura para o valor de *Diffie-Hellman* (DH^A) com a chave privada da PARTE A. Ainda, na sequência, o resultado é concatenado no início aos seguintes valores: valor resultante do *Diffie-Hellman* (DH^A); *nonce* de destino (n^B); e *nonce* gerado (n^A). Após a concatenação ocorre a cifragem dos dados com a chave pública (K^B_{pub}) da PARTE B. Após a cifragem dos dados é parado o contador de tempo de processamento (ctp) que, após o cálculo, resulta no valor do tempo de processamento (tp^A) que é concatenado no fim do pacote cifrado. Posteriormente, esse pacote resultante é enviado para a PARTE B e, em conjunto, é iniciado o contador de tempo total (ctt). Por fim, finaliza-se este passo e encaminha-se para o oitavo passo.

Oitavo passo: inicia-se monitorando se o contador de tempo total (ctt) não extrapolar a soma do tempo de processamento (tp^A) com o tempo de rede (tr), e com tolerância de 10% de acréscimo no tempo total (tpA + trB + ((tpA + trB) * 0,1)), caso ultrapasse o tempo, termina este passo e inicia-se o processo de finalização, caso não, monitora-se o recebimento do pacote da PARTE B. Ao receber o pacote da PARTE B, finaliza-se o contador de tempo total (ctt) e a PARTE A faz a decifragem do pacote recebido com a chave de sessão (DH_K), inicia-se o processo de análise de *nonce* A com o *nonce* (n^A) recebido e armazenado. Assim, ao final, verifica-se se a descifragem do pacote ocorreu corretamente com a leitura da *Flag Ack*, encerrando-se, assim, o processo de concordância de chave simétrica (secreta) entre as partes e passando para o nono passo.

Nono e sucessores passos: após a troca de chaves, a PARTE A e a PARTE B serão capazes de trocar dados cifrados com uma chave simétrica (DH_K), que pode durar tanto por apenas uma troca de mensagem como também ser permanente. A partir desse momento, a parte A pode usar os métodos *Publish* e *Disconnect* presentes em sua biblioteca de *interface* (ligação entre aplicação final e protocolo proposto).

4.2.1.2 PARTE B - servidor

Primeiro passo: recebe o pacote da PARTE A e armazena o *nonce* da parte (n^A).

Segundo passo: inicia-se com a PARTE B calculando e armazenando o valor de *nonce* a partir da função 1 (n^B). Sequencialmente, a PARTE B concatena-se em um pacote a *Flag Ack* ativada - utilizado valor booleano falso para representar a *Flag Ack* - e o *nonce* (n^B) gerado e o *nonce* (n^A) recebido anteriormente. Posteriormente, inicia-se o contador de tempo de rede (ctr) e envia-se o pacote criado para a PARTE A, após isso, passa-se para o terceiro passo.

Terceiro passo: inicia-se monitorando se o tempo de recebimento não extrapola 5 segundos, caso ultrapasse, termina este passo e inicia-se o processo de finalização, caso não, continua a autenticação neste passo. Ao receber o pacote da PARTE A, finaliza-se o contador de tempo de rede (ctr) e inicia-se o contador de tempo de processamento (ctp). Em seguida, a autenticação da PARTE B abre o pacote recebido e entra no processo de análise da assinatura dos parâmetros, depois entra no processo de análise de *nonce* B (n^B) com o recebido e armazenado, após isso, entra-se no processo de análise da função desafio-resposta. Somente após toda as análises serão armazenados os valores de *nonce* da parte opositora (n^A), da função desafio-resposta (Fdr^A), da chave pública (K^A_{pub}) e o tempo de processamento (tp^A). Por fim, passa-se para o quarto passo.

Quarto passo: inicia-se o contador de tempo auxiliar (ctx) e a PARTE B gera um par de chaves assimétricas, sendo uma pública (K^B_{pub}) e a outra privada (K^B_{priv}). Também é gerado um novo valor de *nonce* (n^B) a partir de parâmetros adotados na Função 1. Na sequência, é calculado o valor resultante da função desafio-resposta ($F(K^A_{pub})$). Além da criação de uma função desafio-resposta (Fdr^B). Após isso, ocorre a assinatura dos parâmetros gerados anteriormente (K^B_{pub}, $F(Fdr^A)$, Fdr^B e n^A), em conjunto com o *nonce* recebido (n^B) no passo anterior. Então, é encerrado o contador de tempo de processamento (ctp) e calculado o valor de tempo de processamento (tp^B). Além disso, é encerrado o contador de tempo auxiliar (ctx), que será subtraído do valor do contador de tempo de rede (ctr), resultando, assim, no valor do tempo de rede (tr^A). Sequencialmente, é montado um pacote contendo: sua chave pública (K^B_{pub}); a resposta para a função desafio-resposta ($F(K^A_{pub})$); a função de desafio-resposta (Fdr^B) gerada; o valor de *nonce* que no passo anterior recebeu da parte opositora (n^A); o valor de *nonce* gerado (n^B); a assinatura dos parâmetros; e o valor de tempo de processamento (tp^B). Em seguida é iniciado o contador de tempo total (ctt) e é enviado o pacote para a PARTE A.

Quinto passo: inicia-se monitorando se o contador de tempo total (ctt) não extrapolar a soma do tempo de processamento (tp^B) com o tempo de rede (tr^B), e com tolerância de 10% de acréscimo no tempo total (tpA + trB + ((tpA + trB) * 0,1)), caso ultrapasse, termina este passo e inicia-se o processo de finalização, caso não, monitora-se o recebimento do pacote da PARTE A. Ao receber o pacote, finaliza-se o contador de tempo total (ctt). Posteriormente a PARTE B abre o pacote recebido e entra no processo de análise da assinatura dos parâmetros, depois entra no processo de análise de *nonce* B (n^B) com o recebido e o armazenado, após isso, entra-se no processo de análise da função desafio-resposta. Sequencialmente, caso passar por todas as análises, a PARTE B armazena o *nonce* de A (n^A) recebido, a chave pública de A (K^A_{pub}) e, por fim, passa-se para o sexto passo.

Sexto passo: inicia-se o contador de tempo de processamento (ctp) e realiza-se o cálculo dos valores de *Diffie-Hellman*. Calcula-se um valor resultante do *Diffie-Hellman* (DH^B) juntamente com os valores da base (g) e o valor do módulo (p). Além desses valores, é gerado um novo *nonce* (n^B) e um valor aleatório (iv) para a utilização posterior no algoritmo simétrico. Sequencialmente, é montado um pacote com os valores anteriores concatenados (DH^B, g, p, iv, n^B), juntamente com o *nonce* recebido da parte opositora (n^A). Posteriormente, ocorre a assinatura dos valores de *Diffie-Hellman* (DH^B), com seus parâmetros (g e p) e o valor aleatório (iv). O resultando do valor assinado é concatenado com o pacote anterior montado, e esse é cifrado com a chave pública do destinatário (K^A_{pub}). Em continuação, é parado o contador de tempo de processamento (ctp) e calculado o tempo de processamento (tp^B), esse valor é concatenado no final do pacote anterior. Por fim, é iniciado o contador de tempo total e ele é enviado para a PARTE A do pacote resultante. Ao final do processo, passa-se para o sétimo passo.

Sétimo passo: inicia-se monitorando se o contador de tempo total (ctt) não extrapola a soma do dobro do tempo de processamento (tp^B) com o tempo de rede (tr^B), e com tolerância de 10% de acréscimo no tempo total (tpA + trB + ((tpA + trB) * 0,1)), caso ultrapasse, finaliza-se este passo e inicia-se o processo de finalização, caso não, monitora-se o recebimento do pacote da PARTE A. Ao receber o pacote, termina o contador de tempo total (ctt). Posteriormente, a PARTE B decifra o pacote recebido com a chave privada (K^B_{priv}) e entra no processo de análise da assinatura dos parâmetros, depois entra no processo de análise de *nonce* B (n^B) recebido e do armazenado. Em continuação, armazena-se o *nonce* (n^A) recebido e calcula-se a chave de sessão com *Diffie-*

Hellman e seus parâmetros (g e p). Por fim, passa-se para o oitavo passo.

Oitavo passo: inicia-se montando um pacote que contém a *flag* com valor falso que representa o entendido (Ack) e o valor de *nonce* recebido no passo anterior (n^B). Sequencialmente, cifra-se o pacote com a chave de sessão (DH_K) e envia-se para a PARTE A o pacote.

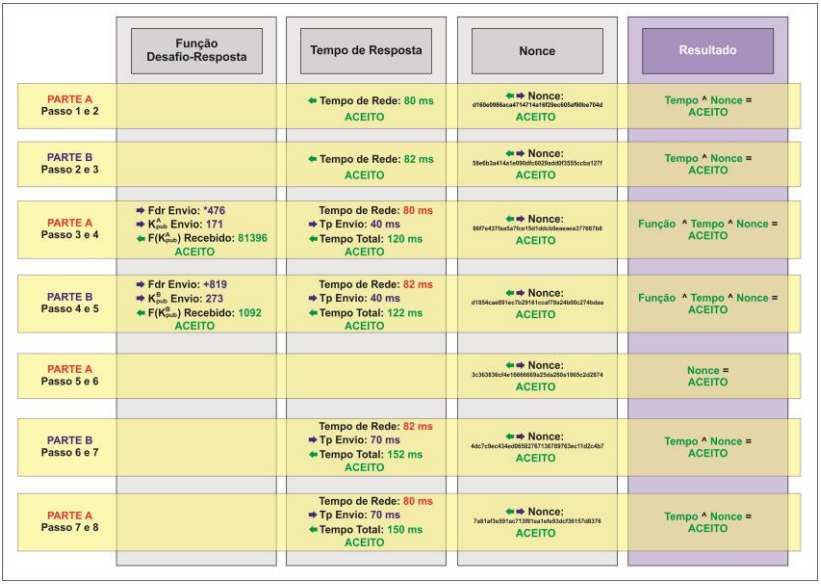
Nono e sucessores passos: após a troca de chaves, a PARTE B é capaz de receber dados cifrados com uma chave simétrica (DH_K), que pode durar por apenas uma troca de mensagens ou pode ser permanente.

4.2.2 Fatores de autenticação da proposta

Os fatores de autenticação têm o objetivo de detectar uma autenticação com seu destinatário ou uma descontinuação da conversa entre as partes. Os autores Piramuthu e Doss (2017) mencionam que, a partir das condições coletadas durante as comunicações se pode chegar a algumas conclusões com as informações. Por meio desse ponto de vista, são montados três fatores de autenticação: a função desafio-resposta (BACKES et al., 2007), o tempo de resposta e o *nonce*. A partir de informações coletadas em cada passo da troca de mensagens no *handshake*, pode-se definir cada fator. Cada passo, em conjunto com os fatores de autenticação, pode ser visualizado na Figura 9.

Função desafio-resposta: é um mecanismo de verificação de identidade por meio de uma função matemática. Essa verificação pode resultar em uma descontinuação da comunicação, caso o valor de resposta da função seja incorreto. A função desafio-resposta é composta por três componentes: o valor desafiado (K^x_{pub}); a função matemática (Fdr^x) que deve ser armazenada na parte receptora, para posterior aplicação do valor desafiado; e o valor resultante ($F(K^x_{pub})$), que é o resultado da aplicação do valor desafiado na função matemática do desafio-resposta. No caso desse trabalho, foi utilizado a seguinte definição para a função: a função é gerada a partir de três possibilidades de funções matemáticas (+, - ou *) e um número aleatório de 100 a 1000.

Figura 9 - Fatores de autenticação



Fonte: elaborada pelo autor (2019).

Tempo de resposta: é um mecanismo de coleta de dados e análise de semelhanças de identidades, que mensura o *delay* resultante do envio até a recepção do pacote de resposta. Tendo como conjunto de cálculo, por parte do cliente (PARTE A) e servidor (PARTE B), o valor do tempo de processamento (tp) mais o valor de tempo de rede (tr) entre as partes mais uma taxa de tolerância de 10% ($tp^x + tr^x + ((tp^X + tr^X) * 0,1)$).

Nonce: é um mecanismo de identificação dos pacotes, que se utiliza o *nonce* do *handshake* para a verificação. A cada passo é verificado se o *nonce* enviado é o mesmo que é recebido no próximo passo, o que é feito por meio da Função 1, que pode estar corrompida, caso alguma parte receba um valor diferente ao enviado. O *nonce* representa o momento (tempo) de envio concatenado ao identificador de destino, identificador de origem e um número aleatório, como apresentado na Função 1.

Inicialmente, os três fatores trabalham independentemente, porém, a cada passo de recepção, eles podem declarar descontinuidade da comunicação entre as partes, caso haja divergência em um dos fatores. Pode-se visualizar na Figura 9 um exemplo de procedimento

com a função desafio-resposta, tempo de resposta e *nonce*, durante cada passo do *handshake* e em cada parte comunicante. Por fim, são esses fatores de autenticação que trazem garantias que o protocolo traz em relação aos ataques.

5 AVALIAÇÃO DO PROTOCOLO

Neste capítulo, o protocolo proposto é avaliado mediante sua aplicação em um estudo de caso a partir do desenvolvimento de um protótipo que implementa o protocolo proposto.

5.1 PROTÓTIPO

A fim de verificar a viabilidade técnica e o funcionamento correto do protocolo proposto, além de servir de base para uma futura adição em alguma ferramenta de nível de aplicação, foi desenvolvido inicialmente um protótipo em C++ para dispositivos como Arduíno e Raspberry Pi, que posteriormente foi aprimorado em Java, e que serve de base para uma aplicação em dispositivos embarcados e *smartphones*.

5.1.1 Escopo e ambiente de avaliação

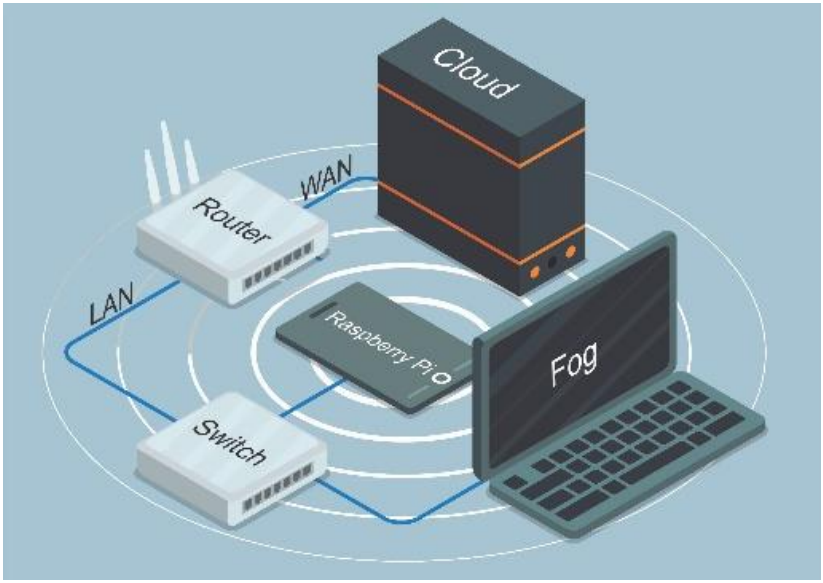
A presente pesquisa visa a modelagem de uma autenticação que é aplicada em dispositivos de memória restrita, desse modo, o escopo de autenticação do aplicativo é um ambiente restrito de memória, em que o modelo abrangerá os métodos de criptografia: AES (*Advanced Encryption Standard*), RSA e *Diffie-Hellman.* Assim, o escopo maior do trabalho é adaptar os sistemas criptográficos apropriados para atender aos requisitos de um método de autenticação otimizado, utilizando a linguagem C++ e Java, e para compilação o GCC 7.4 para o C++ e JDK 1.8.0_191 para o Java.

A arquitetura e plataforma de avaliação foi implementada em um ambiente controlado no Laboratório de Redes e Gerência da Universidade Federal de Santa Catarina, no Brasil. O ambiente consiste em um equipamento, no caso foi utilizado um Raspberry Pi 3 do modelo B com 1 GB de RAM e cartão de 16 GB de memória regrável, além do *clock* ser de 1.2 GHz. Foi utilizado esse dispositivo pois ele tem a possibilidade de utilizar um cartão removível, utilizando diferentes tamanhos de memória. Por fim, foi utilizado um ambiente semelhante ao apresentado na Figura 10 para verificar a viabilidade técnica do método de autenticação para a *Fog Computing* em vez da *Cloud Computing*.

Vale ressaltar que o foco do trabalho é desenvolver a autenticação do sensor/atuador até o primeiro servidor. Nesse sentido, quando a solução é aplicada no contexto de *Fog Computing*, as transações correm entre o sensor/atuador e o *Gateway* intermediário conforme explanado no capitulo de fundamentação teórica. Por outro lado, quando aplicado

no contexto de *Cloud Computing*, as transações correm entre o sensor/atuador e a *Cloud/Data Center*.

Figura 10 - Ambiente de avaliação



Fonte: elaborada pelo autor (2019).

***Nó sensor*:** consiste em uma camada da borda da rede IoT. Inicialmente, quando foi iniciado o estudo de autenticação mútua no trabalho Grüdtner et al. (2018) foi utilizada uma placa de Arduíno UNO que contém apenas 32 KiB de Flash, e utilizou-se 31 KB para o programa desenvolvido. No entanto, verificou-se que o protocolo desenvolvido não estava com nível de segurança adequado. Então, foi aprimorado o desenvolvimento do protocolo anterior e desenvolveu-se o código baseado no protocolo. Com isso e após otimizações o código atingiu 119,5 KB de memória, portanto, não sendo possível a implantação na placa Arduíno. Optou-se por utilizar a placa Raspberry Pi 3 com memória removível (microsd).

***Nó Fog*:** consiste em uma camada intermediária com um ambiente computacional reduzido em comparação com a *Cloud*, já que também está localizado perto da borda, ou simplesmente dentro da mesma rede (IBRAHIM, 2016). Para isso, foi utilizado um notebook com processador i7 de 8 núcleos e com 8 GB de RAM, com o Ubuntu

18.04 LTS. Foi utilizado esse notebook, pois ele estava disponível para testes e se apresentava mais potente que o sensor. A distância entre o servidor e o cliente é de aproximadamente 100 metros, localizada no mesmo domínio da Universidade.

***Nó Cloud*:** consiste em um ambiente de alta capacidade computacional e distribuição esparsa (AMIN et al., 2018). Para esse contexto, uma instância redundante do *Google Cloud Compute Engine* foi utilizada. Especificamente, foi utilizado como servidor uma máquina virtual de 1 núcleo com 8 GB de RAM e sistema operacional Ubuntu 18.04 LTS. Essa máquina é compatível com a plataforma de CPU *Skylake*. Foi utilizada a Cloud da Google pois foi a mais conveniente para o caso, como também, havia a disponibilidade de realizar os testes gratuitamente. A distância entre o servidor e o cliente é de aproximadamente 4.758 quilômetros, localizado o cliente na Universidade (Brasil) e o servidor nos servidores da Google (Carolina do Sul, EUA).

5.1.2 Protótipo em C++

Para o desenvolvimento do protótipo em C++ deve-se levar em conta as peculiaridades que a linguagem tem, como a utilização de ponteiros e a não utilização de *strings*. A seguir será abordada de forma dividida a primeira versão de implementação e depois a segunda versão de implementação.

5.1.2.1 Implementação v1

Conforme abordado anteriormente, inicialmente houve a implementação de um protocolo desenvolvido no artigo Grüdtner et al. (2018). Posteriormente, houve um estudo sobre as trocas de mensagens e desenvolveu-se o novo protocolo de autenticação mútua apresentado na Figura 8. A primeira versão do código com a autenticação mútua baseado na Figura 8 não teve o cuidado de otimizar o código, apenas em desenvolver e aplicar. Como resultado o código ficou com 185 KB (compilado), não era possível realizar chamadas para a biblioteca e tinha vários problemas com ponteiros.

5.1.2.2 Implementação v2

Iniciou-se a implementação do código otimizado usando o código disponível na primeira versão. Com o objetivo de reduzir a alocação de

memória e reduzir o tamanho do código, as otimizações realizadas em relação ao código anterior são: utilização de mesmas variáveis, utilização de ponteiro em passagem de métodos, remoção de métodos não utilizados e adequação do tamanho das variáveis.

Para poder testar toda a funcionalidade do protocolo em desenvolvimento, foi feita uma *interface* de ligação entre o protocolo e a camada de aplicação, sendo chamada esta *interface* desenvolvida em C++ de *IoT_Auth_2*. Dessa forma, a aplicação implementa classes juntamente com bibliotecas que podem ser utilizadas nas chamadas para conectar, enviar e receber dados, desconectar e verificar conexão. Essa *interface* é baseada no protocolo MQTT, que tem chamadas específicas em cada parte (cliente ou no servidor).

Quando um cliente ou dispositivo sensor/atuador deseja adquirir uma conexão estável com o servidor, deve-se chamar uma conexão de *handshake* do protocolo proposto (Quadro 3 - 1ª opção). Tem-se como pré-requisito o IP e a porta de destino, após isso, a conexão será estabelecida por meio de uma conexão UDP com o servidor.

Para o cliente ou dispositivo sensor/atuador enviar dados/informação, deve-se chamar o método de publicação (Quadro 3 - 2ª opção), após isso, o protocolo envia esses dados de forma cifrada. A cifragem desses dados acontece no momento da chamada para a publicação a determinado destino, ou seja, nenhum dado que é enviado para publicação sai do emissor em claro. A cifragem para publicação ocorre por meio de criptografia simétrica, utilizando uma chave de sessão, que anteriormente foi concordada por meio do *handshake* da chamada de conexão.

Por outro lado, para o cliente ou dispositivo sensor/atuador receber informações, ele deve fazer uma requisição de dados ao servidor por meio de um método de escuta (Quadro 3 - 3ª opção). Após a requisição, o servidor enviará, via publicação ao sensor, os dados requeridos. Todas essas transferências são cifradas antes do encaminhamento ao destinatário. A cifragem da requisição de dados ocorre de forma similar ao que ocorre na publicação.

Outras duas chamadas estão disponíveis no Quadro 3, sendo elas a “desconexão com o servidor” (4ª opção) e a “verificação de conexão” (5ª opção). A chamada da desconexão ocorre por meio de uma chamada de publicação simples, repassando uma mensagem de *done*. Por outro lado, a chamada de verificação de conexão, tem a capacidade de atualizar chave de sessão e também de verificar se a conexão está estabelecida.

Quadro 3 - Métodos de chamada do cliente/dispositivo sensor/atuador em C++

```
1st. int connect( char *address, int port );
2º.  int publish( char *data );
3º.  string listen( );
4º.  int disconnect( );
5º.  bool isConnected( );
```

Fonte: elaborado pelo autor (2019).

No outro lado da comunicação está localizado o servidor, que tem chamadas semelhantes às dos sensores/atuadores, porém a principal função do servidor é receber informações. Dessa forma, o processo inicia-se com a espera de requisições de conexões (Quadro 4 - 1ª opção), posteriormente, ocorre a chamada para ouvir os dados que são enviados (Quadro 4 - 2ª opção), além da requisição de publicação de dados ao sensor que requisitou (Quadro 4 - 3ª opção) e, por último, semelhante aos sensores, ocorre a chamada de desconexão com o sensor requisitante (Quadro 4 - 4ª opção) e a verificação de conexão com atualização de chave de sessão (Quadro 4 - 5ª opção).

Quadro 4 - Métodos de chamada do servidor em C++

```
1º.  bool wait connection( );
2º.  string listen( );
3º.  int publish( char *data );
4º.  int disconnect( );
5º.  bool isConnected( );
```

Fonte: elaborado pelo autor (2019).

Para uso do protocolo proposto para o cliente em um ambiente, utiliza-se a classe de ligação nomeada de *AuthClient*. Por meio dessa, foi desenvolvida uma aplicação final de envio de dados que simboliza o sensoriamento de temperatura do ambiente.

Inicia-se o exemplo do Quadro 5 com a requisição de conexão com o servidor, por meio do argumento de execução, passando o endereço IP e a porta. Posteriormente, verifica-se a conexão, caso for afirmativa a conexão, é atualizada a chave de sessão, caso não, a aplicação é finalizada. De forma sucessiva, a atualização da chave de sessão pública dados no servidor. De forma, reversa, o cliente requere dados do servidor e permanece escutando o envio dos dados. Por fim, o

cliente desconecta do servidor que anteriormente estava conectado, estando ela agora livre para uma nova conexão (*handshake*).

A biblioteca em C++ está disponível no *github*, no seguinte link: https://github.com/LukasGrudtner/iotAuth.2.

Quadro 5 - Exemplo de aplicação do cliente em C++

```
#include <iostream>
#include "Auth/AuthClient.h"

using namespace std;
AuthClient auth;
int main(int argc, char *argv[]){
    auth.connect(argv[1], argv[2]);
    char data[] = “temperature: 15ºC”;
    if (auth.isConnected()){
        cout << “Sent: ” << data << endl;
        auth.publish(data);
        cout << “Received: ” << auth.listen() << endl;
        auth.disconnect();
    }
}
```

Fonte: elaborado pelo autor (2019).

Por outro lado, para uso do protocolo proposto pelo servidor em um ambiente, utiliza-se a classe de ligação nomeada de *AuthServer*. Por meio desse, foi dado continuidade à aplicação final de recebimento de dados de sensoriamento simbólico de temperatura do ambiente.

Inicialmente, no exemplo de aplicação apresentado no do Quadro 6, declara-se esperando conexão por meio da porta inserida nos argumentos da execução. Posteriormente, verifica-se a conexão e atualiza-se a chave de sessão, estando pronta a aplicação para a publicação e requisição de dados. Desse modo, ela permanece ouvindo as requisições de publicação e requisição de dados e, posteriormente, caso seja uma requisição de dados, o servidor publica um dado ao sensor conectado. Por fim, caso seja um pedido de desconexão, o servidor desconecta do cliente.

Quadro 6 - Exemplo de aplicação do servidor em C++

```
#include <iostream>
#include "Auth/AuthServer.h"
using namespace std;
AuthServer auth;
int main(int argc, char *argv[]){
    char data[] = “resposta do request”;
    auth.wait_connection(argv[1]);
    while (auth.isConnected()){
        Char[] received = auth.listen();
        cout << “Received: ” << received << endl;
        if(received == “request”){
            cout << “Publish: ” << auth.publish(data) << endl;
        }
        If(received == “done”){
            Auth.disconnect();
        }
    }
    Cout << “SERVIDOR DESCONECTADO” << endl;
}
```

Fonte: elaborada pelo autor (2019).

5.1.2.3 Resultados e contribuições do protótipo

A seguir são apresentados valores relacionados ao tempo de processamento de cada passo. Mais especificamente, a Tabela 1 apresenta o tempo que ocorreram avaliando o protocolo em um ambiente *Fog Computing* e na Tabela 2 em um ambiente *Cloud Computing*. Deve-se considerar que pode haver algumas variações, já que as informações utilizam rotas diferentes no tempo de rede apresentado.

Tabela 1 - Resultado de performance em C++ no contexto *Fog Computing* (milissegundos)

Parte	Passo	Rede	Envio Total	Verificação	Assinatura	*Nonce*	Cifragem
Cliente	1º e 2º	0,879	1,012	0,334	-	0,279	-
Servidor	2º e 3º	0,878	1,542	0,101	-	0,177	-
Cliente	3º e 4º	0,921	1,249	0,188	1,556	0,168	-
Servidor	4º e 5º	0,910	2,872	0,246	0,705	0,130	-
Cliente	5º e 6º	0,867	1,998	0,774	7,417	0,215	-
Servidor	6º e 7º	0,946	9,403	0,143	0,102	0,055	0,089
Cliente	7º e 8º	0,930	1,319	0,214	0,172	0,149	3,617

Fonte: elaborada pelo autor (2019).

A guia ‘Rede’ está relacionada ao tempo de rede entre os pares. “Envio Total” é o tempo total de envio para o receptor. ‘Verificação’ é o

tempo total para verificar os fatores de autenticação. ‘Assinatura’ é tempo de assinar os parâmetros. A guia *Nonce* apresenta o tempo de criar um novo *nonce*, usando a Função 1. ‘Cifragem’ é o tempo para a cifragem dos dados que serão trafegados.

Tabela 2 - Resultado de performance em C++ no contexto *Cloud Computing* (milissegundos)

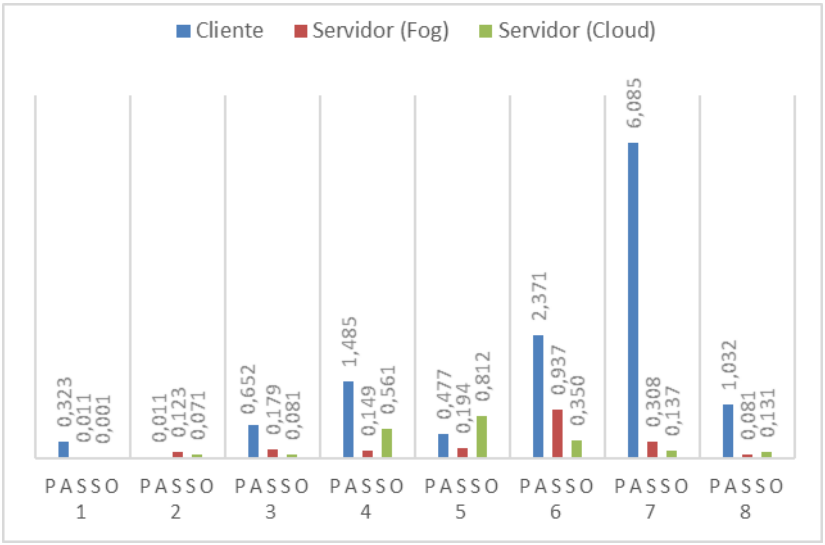
Parte	Passo	Rede	Envio Total	Verificação	Assinatura	*Nonce*	Cifragem
Cliente	1º e 2º	129,952	130,024	0,330	-	0,285	-
Servidor	2º e 3º	129,982	130,647	0,100	-	0,155	-
Cliente	3º e 4º	130,204	130,341	0,435	1,273	0,216	-
Servidor	4º e 5º	130,158	132,132	0,214	0,115	0,052	-
Cliente	5º e 6º	130,356	130,787	0,560	8,722	0,162	-
Servidor	6º e 7º	205,827	213,321	0,141	0,099	0,052	0,091
Cliente	7º e 8º	205,641	205,909	0,214	0,169	0,172	4,292

Fonte: elaborada pelo autor (2019).

Pode ser visto claramente, nas Tabelas 1 e 2 que o tempo de rede depende muito da localização. No contexto da *Cloud Computing*, ocorreu uma variação de mais de 58% entre o primeiro passo e o penúltimo e último passo. Justificando isso pode-se citar que diferentes rotas podem ocorrer entre o cliente e o servidor. Dessa maneira, não há a estabilização de uma conexão segura usando o contexto de *Cloud Computing* neste modelo de protocolo de autenticação mútua.

Em seguida, na Figura 11, o tempo em milissegundos para cada etapa do processamento de autenticação é exibido. Os números mais abaixo são de cada etapa mostrada na Figura 8. Pode-se ver que o tempo mais longo pertence ao cliente porque tem menor poder de processamento. Outro ponto que pode ser analisado é que a nuvem tem alto processamento em relação ao ambiente de *Fog Computing*, pois é um local com alto poder de processamento.

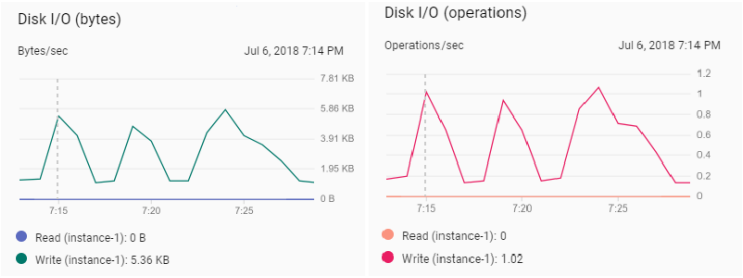
Figura 11 - Tempo de processamento de cada etapa em C++



Fonte: elaborada pelo autor (2019).

Tendo como monitor de memória o servidor, a Figura 12 representa uma memória utilizada durante a autenticação. Pode-se verificar que em torno de 5 KB de memória RAM são usados no processamento (foi executado 3 vezes o mesmo código). A aplicação ao total consumiu 119,5 KB no servidor e 114,6 KB no disco/memória do cliente. Ressalta-se que esse valor supracitado é quando o código está compilado.

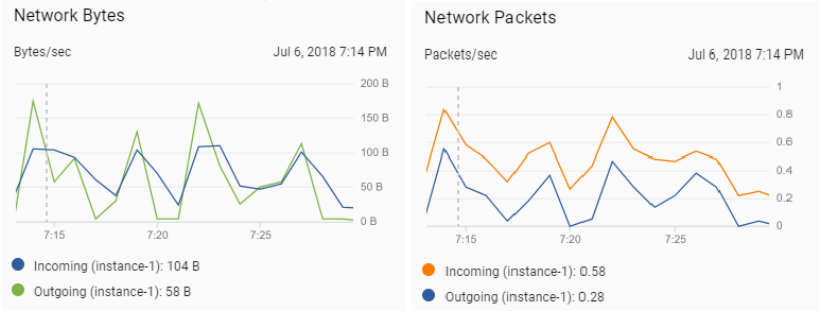
Figura 12 - Consumo de disco local durante o processamento em C++



Fonte: elaborada pelo autor (2019).

Semelhante à análise da memória consumida, mas agora referente à rede consumida, na Figura 13 podemos ver que em torno de 100 Bytes como entrada e 175 Bytes estão saindo de cada solicitação de autenticação.

Figura 13 - Consumo de rede durante o processamento em C++



Fonte: elaborada pelo autor (2019).

5.1.3 Protótipo em Java

O desenvolvimento do protocolo em Java, inicialmente, focou-se no desenvolvimento da *interface* de ligação entre o protocolo e a camada de aplicação, para, posteriormente, ser aplicado em um protótipo funcional por meio de uma aplicação de exemplo.

5.1.3.1 Implementação v1

Conforme abordado anteriormente, inicialmente houve a implementação de um protocolo desenvolvido no artigo Grüdtner et al. (2018) para Arduíno, depois ocorreu o desenvolvimento em C++ sem otimização e posteriormente, ocorreu a otimização do código em C++. Após todos esses desenvolvimentos ocorreu o desenvolvimento em Java. Da mesma forma que ocorreu com a primeira versão com o C++, a primeira versão do código em Java com a autenticação mútua baseado na Figura 8 não teve o cuidado de otimizar, apenas em desenvolver e aplicar. Foi obtido um código de 62,5 KB (*bytecode*) que tinha diversos problemas com os métodos e variáveis que não eram utilizadas. Além disso, as estratégias utilizadas aumentavam o grau de complexidade, chegando a ser $O(n^n)$.

5.1.3.2 Implementação v2

Iniciou-se a implementação do código otimizado baseando-se no código disponível na primeira versão. Da mesma forma que ocorreu no C++, o objetivo era reduzir a alocação de memória e reduzir o tamanho do código. Portanto, as otimizações realizadas em relação ao código anterior são: adequação no uso das mesmas variáveis, remoção de métodos não utilizados e adequação do tamanho das variáveis.

Da mesma forma que o anteriormente explanado, para poder testar toda a funcionalidade do protocolo em desenvolvimento, propôs-se uma *interface* de ligação entre ele e a camada de aplicação, e chamamos essa *interface* desenvolvida em Java de *IoT_Auth_Java*.

De forma semelhante à *interface* desenvolvida em C++, na linguagem de programação Java ela foi desenvolvida baseando-se no protocolo MQTT, e no Quadro 7 pode-se visualizar os métodos de publicação e requisição disponíveis para o cliente. Portanto, para o sensor/atuador fazer a conexão com o servidor deve-se ter o endereço de destino e a porta utilizável (1ª opção), e posteriormente pode-se fazer a publicação de algum dado no servidor, sendo necessário o dado publicável (2ª opção). De forma reversa, o sensor/atuador pode requisitar dados disponíveis no servidor por meio de um tópico específico (3ª opção). Por fim, é possível desconectar do servidor (4ª opção), como também atualizar a chave de sessão (5ª opção).

Quadro 7 - Métodos de chamada do cliente/dispositivo sensor/atuador em Java

```
1st. boolean connect(String address, int port){}
2º.  boolean publish(String data){}
3º.  String request(String topic){}
4º.  boolean disconnect( ){}
5º.  boolean isConnected( ){}
```

Fonte: elaborado pelo autor (2019).

A seguir no Quadro 8, são apresentados os métodos possíveis para o servidor, quando implementado com a linguagem de programação Java, em que se pode esperar conexão em determinada porta (1ª opção), escutar requisição ou publicação em alguma porta por determinado tempo (2ª opção), publicar algum dado quando requisitado (3ª opção), desconectar do sensor/aturador (4ª opção) e atualizar chave de sessão (5ª opção).

Quadro 8 - Métodos de chamada do servidor em Java

```
1st. boolean wait_connect(int port){}
2º.  String listen(int timeout){}
3rd. boolean publish_server(String data){}
4º.  void disconnect( ){}
5º.  boolean isConnected( ){}
```

Fonte: elaborado pelo autor (2019).

Na sequência, no Quadro 8, é apresentado um código de aplicação final para o cliente sensor/atuador, utilizando a biblioteca de *interface* (ligação entre o protocolo de autenticação mútua e a aplicação final), chamada de *IoTAuth_client*, e a representação da utilização dos métodos é apresentada no Quadro 7. Do mesmo modo que ocorreu com a linguagem C++, nesse a aplicação final representa o funcionamento de envio e requisição de dados em um sensoriamento de temperatura.

A biblioteca em Java está disponível no *github* também, disponível no seguinte link: https://github.com/leandroloffi/iotAuth_Java.

O processo inicia-se com a requisição de conexão com determinado servidor, no caso com IP “35.243.230.153” e porta ‘8654’, posteriormente ocorre a verificação se ele está conectado, caso esteja, chave de sessão é atualizada, o cliente publica o dado de temperatura no servidor e, logo após, ele requisita os dados que estão no servidor, sem especificar tópico. Por fim, o cliente requer a desconexão com o servidor conectado anteriormente.

Quadro 9 - Exemplo de aplicação do cliente em Java

```
import java.net.SocketException;
import java.net.UnknownHostException;

public class main_client {
    public static void main(String[] args) throws
SocketException, UnknownHostException, Exception {
        IoTAuth_client client1 = new IoTAuth_client(false);
        do {
             client1.connect("35.243.230.153", 8654);
        } while (!client1.isConnected());

        String ent = "temperature: 25ºC";
        if (client1.isConnected()){
            client1.publish(ent);

            String req = client1.request();
            System.out.println("REQUEST RESP: " + req);

            client1.disconnect();
        }
        System.out.println("CLIENTE DESCONECTADO");
    }
}
```

Fonte: elaborado pelo autor (2019).

A execução do cliente se completa com a execução do servidor, desse modo, no Quadro 10 é apresentada uma aplicação final para o servidor com a utilização dos métodos de *interface* com o protocolo proposto. Tal qual foi desenvolvido em C++ e em Java o processo inicia-se esperando conexão em uma porta (8654), e, em seguida, espera-se a publicação ou a requisição de dados. Do mesmo modo, verifica-se há uma requisição de desconexão, se não ter, a chave de sessão é atualizada.

Quadro 10 - Exemplo de aplicação do servidor em Java

```
public class main_servidor {
    public static void main(String[] args) throws Exception {
        IoTAuth_server server = new IoTAuth_server(false);

        Server.wait_connect(8654);
        String p = “”;
        do {
            p = server.listen(10000);
            if(!p.equals(“timeOut”)){
                System.out.println(server.IP + ":" +
server.Port + " ::: VALOR: " + p);
            }
            if (p.equals("request")) {
                server.publish_server("Olá sou a resposta do
request");
            }
        } while (!p.equals("done") && server.isConnected());
    }
}
```

Fonte: elaborado pelo autor (2019).

5.1.3.3 Resultados e contribuições do protótipo

Perante os exemplos de códigos apresentados anteriormente e o escopo abordado na subseção 5.1.1, foram efetuados alguns testes de velocidade de performance do protocolo proposto. Conforme apresentado anteriormente na Tabela 1, em C++, na Tabela 3 são apresentados os valores de performance com o código em linguagem Java.

Na aba 'Rede' da Tabela 1 tem-se os valores do tempo de rede consumido entre as partes, que, no caso, teve uma média de 2,8416 milissegundos. Na sequência, é exibido, na aba "Envio Total", o valor de tempo total a partir do momento de envio até o recebimento do pacote, e somando esses valores tem-se um total de tempo de 9.025,4956 milissegundos, ou seja, em torno de 9 segundos para ocorrer as trocas de informação e efetuar o *handshake* entre as partes. Na aba 'Verificação', são apresentados os valores de tempo para verificação dos fatores de verificação. Na aba 'Assinatura' é apresentado o tempo que é discorrido ao ser feita a assinatura dos parâmetros. Em sequência, na aba *Nonce* é apresentado o tempo que foi necessário para a criação do *nonce*. Por fim, é exibido na aba 'Cifragem' o tempo total para efetuar a cifragem dos dados, quando necessário.

Tabela 3 - Resultado de performance em Java no contexto *Fog Computing* (milissegundos)

Parte	Passo	Rede	Envio Total	Verificação	Assinatura	*Nonce*	Cifragem
Cliente	1º e 2º	3,324	18,622	0,263	-	51,570	-
Servidor	2º e 3º	3,274	6304,486	481,279	-	14,986	-
Cliente	3º e 4º	2,481	1039,764	24,951	1051,187	0,824	-
Servidor	4º e 5º	3,000	357,234	1,171	5,052	0,162	-
Cliente	5º e 6º	2,413	31,221	908,147	249,811	0,695	-
Servidor	6º e 7º	3,143	1215,482	34,520	2,072	0,190	1,141
Cliente	7º e 8º	2,254	58,685	0,048	228,777	0,347	20,976

Fonte: elaborada pelo autor (2019).

Com as mesmas abas exibidas na Tabela 2, na Tabela 3 são apresentados os valores referentes ao processamento e performance utilizando Java para representação do protocolo aplicado no contexto de *Cloud Computing*, conforme o escopo explicado na subseção 5.1.1. Tendo como destaque, o valor total da soma dos tempos apresentados na aba "Envio Total", de 18.581,5936 milissegundos, ou seja, em torno de

18 segundos e meio para o processo de *handshake*, quando utilizado o contexto de *Cloud Computing*.

Para poder executar e coletar os dados apresentados na Tabela 4, foram necessários o desligamento da verificação do tempo de resposta, já que o tempo de resposta impactaria no não estabelecimento de conexão. Pode-se ver que da linha 4 para a 5 há uma disparidade no tempo de rede, na aba 'Rede'. Por esse motivo, haveria uma não conformidade ao final da verificação do tempo de resposta.

Tabela 4 - Resultado de performance em Java no contexto *Cloud Computing* (milissegundos)

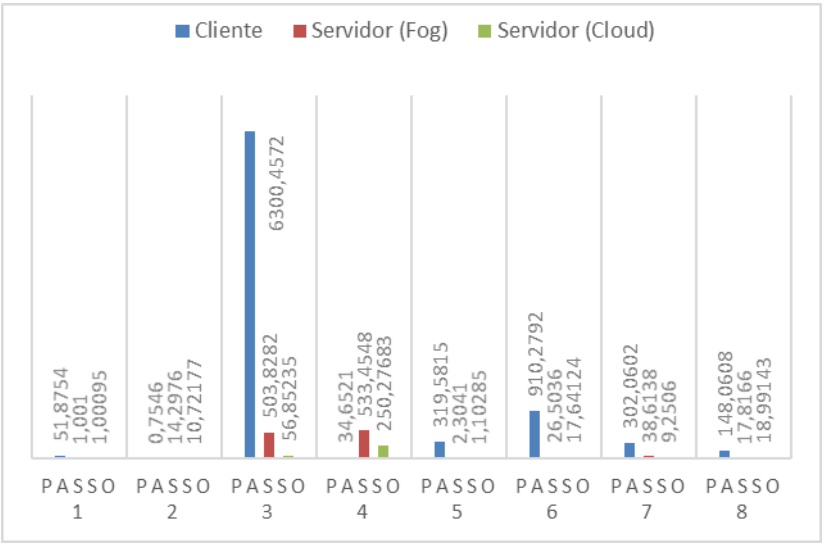
Parte	Passo	Rede	Envio Total	Verificação	Assinatura	*Nonce*	Cifragem
Cliente	1º e 2º	173,287	185,010	0,294	-	47,031	-
Servidor	2º e 3º	169,212	13720,072	46,917	-	10,758	-
Cliente	3º e 4º	168,573	475,702	9,735	1041,486	0,790	-
Servidor	4º e 5º	171,302	526,561	0,657	6,172	0,183	-
Cliente	5º e 6º	766,689	785,433	950,718	287,082	0,431	-
Servidor	6º e 7º	797,352	2067,629	7,336	3,601	0,125	0,988
Cliente	7º e 8º	792,942	821,184	0,051	233,807	0,375	20,842

Fonte: elaborada pelo autor (2019).

Podemos ver, na Figura 14, os tempos de cada passo, divididos em passos iguais aos anteriormente apresentados na Figura 8. Portanto, pode-se verificar que o tempo maior pertence ao cliente, por ter um menor poder de processamento. No caso, o maior tempo de processamento é atribuído ao cliente no passo 3, por se tratar do momento da geração das chaves públicas e privadas, ou seja, devido à complexidade de gerar um valor válido, o tempo de processamento é maior do que nos outros passos.

Outro ponto que podemos analisar, é que o passo 6 é o único momento em que o cliente recebe dados e processa, o que resultou um tempo acima do que seu oponente servidor contém. Isso acontece, devido ao tempo de verificação dos fatores de autenticação, conforme pode ser visto nas Tabela 3 e 4 da aba 'Verificação'.

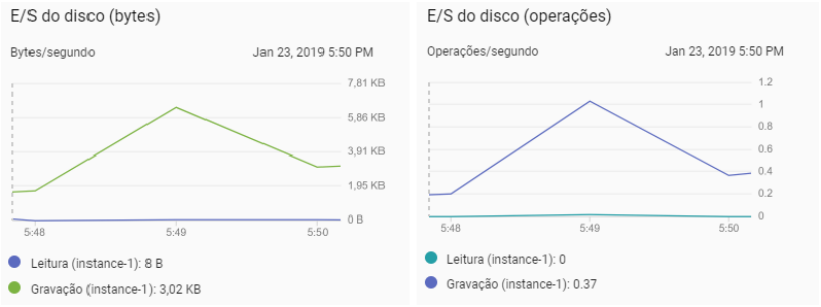
Figura 14 - Tempo em cada passo em Java



Fonte: elaborada pelo autor (2019).

Da mesma forma como ocorreu na avaliação de performance na linguagem C++, em Java utilizou-se também o monitor de memória do servidor, resultando nos dados apresentados na Figura 15. Desse modo, podemos analisar, após a execução, que ouve um pico de processamento de gravação, resultando em torno de 6 KB de uso de disco para escrita de dados.

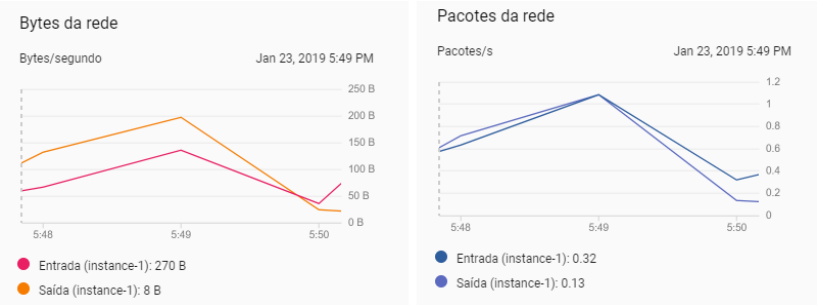
Figura 15 - Consumo de disco local durante o processamento em Java



Fonte: elaborada pelo autor (2019).

Diante do monitoramento dos processos na máquina servidora, pode-se constatar o consumo de rede para a execução do protocolo, conforme apresentado na Figura 16. Desse modo, pode-se analisar que, em média, 1 pacote por segundo de dados teve de entrada e 1 pacote por segundo de dados saiu, sendo que os pacotes de entrada tiveram uma média em torno de 200 Bytes e os pacotes de saída tiveram uma média em torno de 140 Bytes.

Figura 16 - Consumo de rede durante o processamento em Java



Fonte: elaborada pelo autor (2019).

Por fim, a biblioteca desenvolvida tanto para o cliente como para o servidor consumiu 48,5 KB no disco/memória. Ressalta-se que esse valor supracitado é valor quando o código estiver em *bytecode*.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, um novo modelo de autenticação mútua foi introduzido em ambientes de IoT para o contexto de *Fog Computing*. O protocolo proposto foi adequado e otimizado juntamente com alguns protocolos existentes (RSA, AES-192 e SHA-512). A autenticação é realizada durante o *handshake*, mais especificamente durante a verificação dos fatores de autenticação: função de desafio-resposta, tempo de resposta e *nonce*.

Por meio de uma aplicação final, foi possível testar a viabilidade técnica utilizando as linguagens de programação C++ e Java. Essa avaliação ocorreu em dois ambientes de IoT, no contexto de *Fog Computing* e *Cloud Computing*. Claramente conclui-se que, em um ambiente *Cloud*, pode haver interrupções que não deveriam existir, justamente por causa do fator de autenticação "Tempo de Resposta". Para ser utilizado em ambientes *Cloud*, outros fatores devem ser pesquisados, como o balanceamento de carga e o roteamento além da previsão do aumento da tolerância no tempo de resposta (acima de 58%). Por outro lado, no contexto do *Fog Computing* a velocidade de autenticação do modelo se destacou, já que a infraestrutura está próxima dos sensores/atuadores.

Como a infraestrutura de processamento no contexto de *Fog Computing* está localizada próxima dos sensores, tem menor possibilidade de ocorrer o ataque *Man-in-the-Middle*. Outro ponto que ajuda a combater os ataques é a taxa de tolerância de 10% testada em laboratório, pois com a capacidade atual de processamento no mundo, tornaria inviável um ataque.

A combinação de autenticação mútua e diferentes fatores em um ambiente de IoT é uma contribuição deste trabalho. No entanto, trabalhos relacionados resumidos no Quadro 1 não combinam autenticação mútua e diferentes fatores em suas propostas. Além disso, os trabalhos relacionados não descrevem nenhum resultado de desempenho ou performance em ambientes semelhantes. Consequentemente, não é possível comparar os resultados de desempenho obtidos neste trabalho com qualquer outro trabalho relacionado.

6.1 CONTRIBUIÇÕES

O presente trabalho atingiu algumas contribuições significativas para a área de pesquisa de segurança de redes e da área de dispositivos

embarcados. As principais contribuições do trabalho são: (i) desenvolvimento de um modelo de autenticação mútua, que utiliza (ii) estratégias de fatores de autenticação nos nós e que, (iii) foi desenvolvida uma biblioteca otimizada, e ao final, pode (iv) agregar um novo método de autenticação em um ambiente real IoT no contexto de *Fog Computing*.

Além da contribuição estar ligada aos objetivos deste estudo, em relação aos trabalhos anteriores este trabalho apresenta uma contribuição significativa com a autenticação mútua usando funções "desafio-resposta" juntamente com o tempo de resposta e a verificação do *nonce.* O trabalho contribui com o desenvolvimento de uma biblioteca otimizada do modelo para ambientes de IoT, resultando em um código em C++ em torno de 119,5 KB (compilada) e um código em Java de 48,5 KB (*bytecode*). Por fim, o modelo proposto da biblioteca desenvolvida em C++ e Java aplicado ao contexto de *Fog Computing* tem, respectivamente, uma média de 24,03 milissegundos para C++ e de 9,02 segundos em Java.

Como o trabalho envolveu um estudante de graduação do Curso de Bacharelado em Ciências da Computação, foram inicialmente feitas algumas pesquisas em relação aos fatores de autenticação e análise de alguns resultados. Dessa forma, foi desenvolvido o artigo Grüdtner et al. (2018), preliminarmente com estudo da autenticação e com o fator de autenticação desafio-resposta que foi publicado no Workshop de Trabalhos de Iniciação Científica e Graduação do SBRC. Posteriormente, o artigo Loffi et al. (2019) apresenta resultados em relação a presente pesquisa, juntamente com o desenvolvimento do caso de uso em C++ e seus resultados, também incluída a avaliação do protocolo com a ferramenta AVISPA que foi publicado no *11th International Conference on COMmunication Systems & NETworkS* (COMSNETS 2019).

6.2 LIMITAÇÕES

O protocolo utilizou algumas técnicas para os fatores de autenticação, sendo eles o desafio-resposta, tempo de resposta e o de *freshness* (*nonce*). O tempo de resposta deve ser analisado a cada passo de recebimento e, caso não haja sincronia entre as partes, podem ocorrer falhas na tentativa da autenticação mútua. Desse modo, os dispositivos não podem trabalhar com múltiplas *threads* no momento da autenticação, devendo o dispositivo servidor/sensor ser exclusivo para cada dispositivo sensor/servidor que desejar se autenticar.

O trabalho se limitou a considerar o consumo de memória. Não foi possível adequar o protocolo para um menor consumo de energia já que não foi realizada a verificação do consumo de energia.

Outra limitação do trabalho é que inicialmente ao implementar os códigos houve limitação de memória do Arduíno. Pela impossibilidade de desenvolver o trabalho com o Arduíno foi utilizado dispositivo que tem memória removível (microsd). Assim, o dispositivo usado não se caracteriza diretamente como um ambiente restrito de memória, porém, como resultado das otimizações nos códigos a utilização da memória foi a mínima possível em cada linguagem de programação, C++ e Java.

Portanto, a principal limitação deste trabalho em relação ao desenvolvimento está vinculada aos dispositivos que possuem restrição de memória, o que gerou alguns problemas no desenvolvimento inicial do protocolo na linguagem de programação C++, como a inclusão de bibliotecas de cifragem. E ao final do desenvolvimento em Java houve ainda alguns erros de tamanhos de blocos na cifragem, porém, isso foi contornado com a aplicação de uma técnica de quebra de pacotes e uma posterior cifragem individual. Por fim, em termos de algoritmos adequados, o protótipo não apresenta alto desempenho na criptografia de dados com a biblioteca de algoritmos AES na linguagem C++ e não apresentam alto desempenho os sensores/atuadores para gerarem um novo par de chaves para a cifragem assimétrica na linguagem Java.

6.3 TRABALHOS FUTUROS

Após todo o desenvolvimento do presente trabalho, alguns trabalhos futuros podem ser descritos, tais como:

a) melhoria do modelo com o uso de vários tipos de cifras, utilizando suítes de *handshake*, semelhantes às utilizadas em TLS/SSL;
b) realização de testes de usabilidade para verificar os efeitos do modelo de protocolo nos usuários;
c) aplicar o protocolo em ambientes reais, como também integrá-lo em algum serviço ou outro protocolo da camada de aplicação;
d) de forma oposta a característica do trabalho de redução de memória, também pode-se verificar o consumo de energia do protocolo com a aplicação em ambientes com uso de baterias; e,
e) adequar as bibliotecas utilizadas reduzindo ainda mais o tamanho do arquivo final de execução, além da possibilidade de ser implementado em um ambiente mais restrito de memória.

REFERÊNCIAS

AMIN, Ruhul et al. A light weight authentication protocol for IoT-enabled devices in distributed Cloud Computing environment. **Future Generation Computer Systems**, v. 78, p. 1005-1019, 2018.

ARMANDO, Alessandro et al. AVISPA: automated validation of internet security protocols and applications. **ERCIM News**, v. 64, 2006.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The internet of things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787-2805, 2010.

BACKES, Michael et al. A calculus of challenges and responses. *In:* FORMAL METHODS IN SECURITY ENGINEERING, 7., 2007. **Proceedings [...].** ACM: 2007.

BLANCHET, Bruno et al. Modeling and verifying security protocols with the applied pi calculus and ProVerif. **Foundations and Trends® in Privacy and Security**, v. 1, n. 1-2, p. 1-135, 2016.

BONOMI, Flavio et al. Fog computing and its role in the internet of things. *In:* MOBILE CLOUD COMPUTING, 1., 2012. **Proceedings [...].** ACM: 2012.

DIERKS, Tim; RESCORLA, Eric. **The transport layer security (TLS) protocol Version 1.2**. 2008. Disponível em: https://pdfs.semanticscholar.org/6a74/a8573cb1bd15c5f4fa4e047613d2340e61b9.pdf?_ga=2.39308257.350808236.1548524097-1031158746.1548524097. Acesso em: 26 jan. 2019.

ESINER, Ertem; DATTA, Anwitaman. Two-factor authentication for trusted third party free dispersed storage. **Future Generation Computer Systems**, v. 90, p. 291-306, 2019.

GOPE, Prosanta et al. Lightweight and privacy-preserving RFID authentication scheme for distributed IoT infrastructure with secure localization services for smart city environment. **Future Generation Computer Systems**, v. 83, p. 829-637, 2017.

GRANJAL, Jorge; MONTEIRO, Edmundo; SILVA, Jorge Sá. Security for the internet of things: a survey of existing protocols and open research issues. **IEEE Communications Surveys & Tutorials,** v. 17, n. 3, p. 1294-1312, 2015.

GUBBI, Jayavardhana et al. Internet of Things (IoT): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, v. 29, n. 7, p. 1645-1660, 2013.

GRÜDTNER, Lukas Derner et al. Autenticação mútua de nós sensores com nós intermediários para IoT no contexto de Fog Computing. **Workshop de Trabalhos de Iniciação Científica e Graduação do SBRC (WTG - SBRC)**, [S.l.], v. 1, may 2018. Disponível em: <http://ojs.sbc.org.br/index.php/sbrcwtg/article/view/2508>. Acesso em: 26 jan. 2019.

HU, Pengfei et al. A unified face identification and resolution scheme using cloud computing in Internet of Things. **Future Generation Computer Systems**, v. 81, p. 582-592, 2018.

IBM. **Uma visão geral do handshake SSL ou TLS.** 2018. Disponível em: https://www.ibm.com/support/knowledgecenter/pt-br/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q009930_.htm. Acesso em: 18 jan. 2019.

IBRAHIM, Maged Hamada. Octopus: An Edge-fog Mutual Authentication Scheme. **IJ Network Security**, v. 18, n. 6, p. 1089-1101, 2016.

IORGA, Michaela et al. **Fog Computing Conceptual Model** (N. Special Publication) (NIST SP-500-325). 2018.

IVAN, Ion et al. Aspects Concerning the Optimization of Authentication Process for Distributed Applications. **Theoretical and Applied Economics**, v. 6, n. 6, p. 39, 2008.

JAN, Mian Ahmad et al. A payload-based mutual authentication scheme for Internet of Things. **Future Generation Computer Systems**, 2017.

KHAN, Minhaj Ahmad; SALAH, Khaled. IoT security: Review, blockchain solutions, and open challenges. **Future Generation Computer Systems**, v. 82, p. 395-411, 2018.

KONESKI, Eduardo de Meireles et al. **Ambiente de comunicação segura de Internet das Coisas com a utilização do MQTT e TLS.** 2018. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Universidade Federal de Santa Catarina, Florianópolis, 2018.

KOTHMAYR, Thomas et al. A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication. *In:* LOCAL COMPUTER NETWORKS WORKSHOPS (LCN), 37., 2012. **Proceedings [...].** IEEE: 2012. p. 956-963.

KUMAR, Priyan Malarvizhi; GANDHI, Usha Devi. Enhanced DTLS with CoAP-based authentication scheme for the internet of things in healthcare application. **The Journal of Supercomputing**, p. 1-21, 2017.

KUROSE, James F.; ROSS, Keith W. **Redes de Computadores e a Internet.** São Paulo: Pearson, 2006.

LAKATOS, Eva Maria. MARCONI, Mariana de Andrade. **Metodologia científica.** 6. ed. São Paulo. Editora Atlas, 2011.

LAMPORT, Leslie. Password authentication with insecure communication. **Communications of the ACM**, v. 24, n. 11, p. 770-772, 1981.

LI, Nan; LIU, Dongxi; NEPAL, Surya. Lightweight mutual authentication for IoT and its applications. **IEEE Transactions on Sustainable Computing**, v. 2, n. 4, p. 359-370, 2017.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing. Gaithersburg, MD: NIST, 2011.

MAHMUD, Redowan; KOTAGIRI, Ramamohanarao; BUYYA, Rajkumar. Fog computing: A taxonomy, survey and future directions. **Big Data and Cognitive Computing**, v. 2, n. 2, 2018.

MOHAMMED, Mahmoud Musa; ELSADIG, Muna. A multi-layer of multi factors authentication model for online banking services. *In:* INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING, 2013. **Proceedings [...].** IEEE: 2013. p. 220-224.

NEEDHAM, Roger M.; SCHROEDER, Michael D. Using encryption for authentication in large networks of computers. **Communications of the ACM**, v. 21, n. 12, p. 993-999, 1978.

OASIS. **MQTT Version 3.1.1.** 2014. Disponível em: http://docs.oasisopen.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. Acesso em: 17 jan. 2019.

ODELU, Vanga; DAS, Ashok Kumar; GOSWAMI, Adrijit. A secure biometrics-based multi-server authentication protocol using smart cards. **IEEE Transactions on Information Forensics and Security**, v. 10, n. 9, p. 1953-1966, 2015.

OTWAY, Dave; REES, Owen. Efficient and timely mutual authentication. **ACM SIGOPS Operating Systems Review**, v. 21, n. 1, p. 8-10, 1987.

ROMAN, Rodrigo; LOPEZ, Javier; MAMBO, Masahiro. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. **Future Generation Computer Systems**, v. 78, p. 680-698, 2018.

ROUSE, Margaret. **Mutual Authentication.** 2008. Disponível em: https://searchsecurity.techtarget.com/definition/mutual-authentication. Acesso em: 18 jan. 2018.

PEREZ, Cristiano. **Arquiteturas para aplicações realtime utilizando MQTT.** 2017. Disponível em: https://engenharia.elo7.com.br/arquiteturas-para-aplica%C3%A7%C3%B5es-realtime-utilizando-mqtt/. Acesso em: 18 jan. 2019.

PIRAMUTHU, Selwyn; DOSS, Robin. On sensor-based solutions for simultaneous presence of multiple RFID tags. **Decision Support Systems**, v. 95, p. 102-109, 2017.

POPPER, Karl Raimund. **Conhecimento objetivo: uma abordagem evolucionária.** Editora da Universidade de São Paulo, 1975.

SATYANARAYANAN, Mahadev. The emergence of edge computing. **Computer**, v. 50, n. 1, p. 30-39, 2017.

STALLINGS, William. **Criptografia e segurança de redes: princípios e práticas** / William Stallings; tradução Daniel Vieira; revisão técnica Paulo Sérgio Licciardi Messeder Barreto, Rafael Misoczki. – 6. ed. – São Paulo: Pearson Education do Brasil, 2015.

SCHNEIER, Bruce. **Applied Cryptography Protocols, Algorithms, and Source Code in C**. New York, NY, USA: John Wiley and Sons, 1995.

SMITH, Richard E. **Authentication: from passwords to public keys.** Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

TEWARI, Aakanksha; GUPTA, B. B. Cryptanalysis of a novel ultra-lightweight mutual authentication protocol for IoT devices using RFID tags. **The Journal of Supercomputing**, v. 73, n. 3, p. 1085-1102, 2017.

VILLARREAL, María Elena. **Token de privacidade:** um mecanismo de especificação de preferências de privacidade para sistemas de gerenciamento de identidade em nuvem. 2017. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Santa Catarina, Florianópolis, 2017.

VOAS, Jeffrey. Networks of ‘things’. **NIST Special Publication**, v. 800, n. 183, p. 800-183, 2016.

WAZID, Mohammad et al. Secure authentication scheme for medicine anti-counterfeiting system in iot environment. **IEEE Internet of Things Journal**, v. 4, n. 5, p. 1634-1646, 2017.

WILLEKE, Jim. **How SSL-TLS Works.** 2018. Disponível em: https://ldapwiki.com/wiki/How%20SSL-TLS%20Works. Acesso em: 18 jan. 2019.

WU, Fan et al. A novel mutual authentication scheme with formal proof for smart healthcare systems under global mobility networks notion. **Computers & Electrical Engineering**, v. 68, p. 107-118, 2018.

XIA, Feng et al. Internet of things. **International Journal of Communication Systems**, v. 25, n. 9, p. 1101, 2012.

YI, Shanhe et al. Fog computing: Platform and applications. *In:* HOT TOPICS IN WEB SYSTEMS AND TECHNOLOGIES (HotWeb), 3., 2015. **Proceedings [...].** IEEE: 2015. p. 73-78.