# Mecanismo de Autenticação de Dispositivos para Internet das Coisas

**Trabalho de Conclusão do Curso de**
**Tecnologia em Sistemas para Internet**

**Jonathan Monteiro Araujo**
**Orientador(a): André Peres**

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Porto Alegre
Av Cel Vicente, 281, Porto Alegre – RS – Brasil

jonathanmaraujo@gmail.com, andreperes@poa.ifrs.edu.br

***Resumo.*** *Considerando a grande oferta de dispositivos conectados entre si através da internet das coisas, como por exemplo: smarthomes, carros inteligentes, etc; e o crescente número de falhas de segurança envolvendo tais dispositivos, fica evidente a necessidade de incremento dos mecanismos para a implementação de segurança nesse tipo de solução. Um exemplo de incidente recente pôde ser notado com o ataque à empresa Dyn através do uso da botnet Mirai, cujos dispositivos comprometidos são em sua maioria câmeras de vigilância, entre outros objetos conectados à internet. Falhas na implementação de um dos principais atributos de segurança, a autenticação, é um dos pontos explorados para que estes dispositivos aceitem realizar ações maliciosas. Este trabalho objetiva a descrição do desenvolvimento de um mecanismo cuja proposta é a de ser utilizado no processo de autenticação de dispositivos, oferecendo assim um pouco mais de segurança aos mesmos. No trabalho é apresentado um cenário utilizando a plataforma arduino para implementação e validação do mecanismo proposto.*

## 1. Introdução

Atualmente percebe-se que a tecnologia está cada vez mais integrada na vida da população. Um dos aspectos mais notáveis dessa integração pode ser percebido no uso frequente de soluções baseadas em IoT (*Internet of Things*). Segundo a Gartner, empresa referência no mercado de tecnologia desde 1979, no ano de 2016 o número de dispositivos baseados em IoT chegará a 6,4 bilhões, o que corresponde a um aumento de 30% em relação a 2015, com a expectativa de atingir a marca de 20,8 bilhões até 2020 [Gartner 2015]. Desde smarthomes (casas inteligentes) até os wearables (tecnologias vestíveis), a IoT vem crescendo e se integrando ao cotidiano.

No contexto de internet das coisas, uma coisa é definida como um objeto capaz de receber dados do ambiente através de sensores, e possivelmente alterando esse ambiente através de atuadores além de comunicar-se com outros objetos [Alecrim 2016]. Os dados recebidos pelo dispositivo precisam ser processados, esse processamento é realizado por um controlador conectado à rede.

Com o rápido avanço da oferta de dispositivos e o desejo de difusão dessa tecnologia por parte dos fabricantes, tem-se um cenário onde, por falta de conhecimento técnico,

existe pouca preocupação por parte dos desenvolvedores em relação a configurações de soluções de segurança disponíveis para autenticação de dispositivos. Isto resulta em uma brecha considerável quando é levado em conta que esses dispositivos podem ser responsáveis por gerenciar aspectos sensíveis como energia elétrica, água e até outros dispositivos de segurança, por exemplo: câmeras e alarmes, e que a qualquer momento podem ser alvos de ataques que poderiam ocasionar danos físicos, materiais e informacionais.

Recentemente observou-se as consequências do maior ciberataque do tipo DDoS, do inglês *Distributed Denial-of-Service attack* (ataque distribuído por negação de serviço), que sobrecarregou os servidores de produção da empresa Dyn uma das maiores provedoras de servidores DNS (*Domain Name System*) do mundo, o que ocasionou a queda de diversos serviços como: Airbnb, Amazon, Netflix, Reddit, Twitter e Spotify. Tal ataque foi possível graças a um software que se utiliza de uma extensa rede de dispositivos comprometidos do tipo IoT.

O ataque à Dyn foi feito utilizando a metodologia de ataque conhecida como DDoS o qual tem como objetivo tornar indisponível um determinado serviço na rede, através da sobrecarga dos recursos do servidor. Neste tipo de ataque é enviada uma grande quantidade de requisições para o servidor, tal ataque demanda um considerável número de dispositivos fazendo o envio simultâneo. Para tal, os dispositivos precisam ter sido comprometidos, o que pode ocorrer de diversas formas, por exemplo: através da instalação de um *backdoor* (meio secundário de acesso facilitado ao recurso computacional), de escalonamento de acesso no sistema ou falsificação de identidade na rede. Nesse caso, essa massa de dispositivos comprometidos é chamada de *botnet*. Para utilizar a *botnet* é necessário o uso de um software executado de forma distribuída, ou seja, que seja executado em todos os dispositivos da botnet. No caso do ataque à Dyn o software utilizado é conhecido pelo nome de "Mirai". Esse software utilizou-se de uma rede de cerca de 500.000 dispositivos IoT comprometidos. O ataque ocorrido em outubro de 2016 foi capaz de gerar um tráfego de dados registrado de cerca de 1,2 Tbps (Terabits por segundo) [Sun 2016, Loshin 2016]. Um dos motivos que tornaram este ataque possível foi a ausência de verificações de autenticidade das origens das requisições enviadas aos dispositivos.

A Kaspersky Lab [Kaspersky 2017], empresa reconhecida mundialmente na área de cybersegurança, descreveu recentemente no artigo *Honeypots and the Internet of Things* [Kuskov 2017], uma pesquisa realizada nesse ano onde aplicações IoT foram desenvolvidas para servirem como *Honeypots* (softwares desenvolvidos para serem alvos de ataques), a pesquisa tinha por objetivo identificar ameaças e vulnerabilidades no âmbito de aplicações IoT. Os resultados obtidos na pesquisa foram bastante alarmantes, no ano de 2016 a empresa tinha conhecimento de cerca de 3219 tipos de *malwares* (softwares maliciosos) com ênfase em soluções IoT, porém a última pesquisa expandiu esse número para mais que o dobro, atingindo o valor de 7242 ameaças conhecidas. Ainda no artigo o pesquisador Vladimir Kuskov ressalta que a maioria dos dispositivos são atacados visando torná-los integrantes de alguma botnet com fins de uso em atividades ilícitas, o que já resultaria em problemas tanto para o desenvolvedor quanto para o dono do dispositivo. Além disso existe também o risco de o dispositivo vir a ser utilizado para prejudicar duretamente o usuário final, por exemplo para espionar o usuário e chantageá-lo.

A OWASP (*Open Web Application Security Project*, do português Projeto Aberto de Segurança em Aplicações Web [OWASP 2017]) órgão referência na comunidade de cybersegurança desde 2001, conhecido por disponibilizar ferramentas e artigos para desenvolvedores e especialistas de segurança, inaugurou no ano de 2015 um projeto para tratar especificamente de questões de segurança em dispositivos IoT [Miessler 2017]. O projeto está disponibilizando no momento vários artigos, guias de melhores práticas de desenvolvimento e inclusive uma lista com as 10 principais vulnerabilidades.

Este trabalho possui como objetivo a implementação de um mecanismo de autenticação entre dispositivos IoT. Para o desenvolvimento e realização dos testes deste mecanismo será construído um cenário composto por dispositivos comumente utilizados na prototipação de equipamentos de IoT.

Um exemplo de controlador bastante utilizado em ambientes acadêmicos e de prototipagem é a solução de hardware livre Arduino. Arduino corresponde a uma plataforma de prototipagem eletrônica, criada por Massimo Banzi e David Cuartielles em 2005, com o intuito de possibilitar o desenvolvimento de microcontroladores para sistemas interativos de baixo custo e acessível a todos [Arduino 2016b, Lemos 2015]. O arduino suporta a adição de diversos módulos como: shields de rede, módulos de cervos motores, módulos de ultrassom, etc.

O arduino possui uma vasta coleção de bibliotecas e *frameworks* para programação [Arduino 2016a], porém poucas implementam certos aspectos da segurança da informação, como por exemplo as que contemplam o âmbito da autenticidade, um dos principais atributos da segurança computacional[IT 2015]. A falta de autenticidade nas transmissões pode permitir, a qualquer pessoa conectada à mesma rede dos microcontroladores, o envio de informações incorretas ou imprecisas que podem resultar no comportamento impróprio da solução, como por exemplo: uma solução responsável pela detecção de fumaça em um ambiente, haja vista que um atacante se passando por um dos agentes detectores de fumaça envie para o servidor a informação de que fumaça doi detectada no recinto, pode ocasionar a ativação de *sprinklers*, responsável por descarregar água com o intuito de combater incêndios, o que considerando um ambiente com equipamentos sensíveis pode ocasionar em perda material.

Como a maioria dos módulos existentes para aplicação de segurança em sistemas *web* ou *desktop* não são suportados em arduinos, a hipótese deste trabalho parte da seguinte questão: seria possível utilizar alguma metodologia de implementação de autenticação de algum destes módulos para a implementação em um mecanismo, para que este venha a ser empregado no desenvolvimento de soluções em *IoT* que utilizam arduino?

## 2. Autenticação

Segundo a ISO 27002 [IT 2015] pode-se definir a segurança da informação em 5 tópicos principais conhecidos como os 5 pilares da segurança computacional, são eles: Confidencialidade, Autenticidade, Integridade, Disponibilidade e Irretratabilidade. A solução descrita neste artigo tem por objetivo implementar o âmbito da autenticidade em soluções *IoT*.

Define-se como autenticação o ato de validar a procedência ou a identidade de um ativo, que pode representar um objeto ou uma pessoa, que até o momento era consid-

erado desconhecido. Tal validação pode ser feita através de um segredo compartilhado entre ambos os indivíduos, esse segredo é conhecido como senha e serve para identificar indivíduos que possuem autorização, ou seja, que estão devidamente autenticados para acessar um determinado recurso. No meio computacional a senha tornou-se uma das formas mais difundidas e utilizadas de autenticação, por exemplo: através do fornecimento de um *login* (nome utilizado para identificar um ativo perante um sistema) e de sua respectiva senha um usuário estará devidamente autenticado.

Atualmente a autenticação têm-se difundido de forma a não mais abranger apenas a identificação de usuários como também a de ativos como: software e hardware com acesso a bases de dados ou informações sensíveis. O objetivo da autenticação destes ativos é oferecer maior segurança para os recursos disponíveis em uma rede ou na internet.

## 3. Estado da Arte

Mesmo com a demanda crescente na área de segurança em internet das coisas, durante as pesquisas realizadas foram encontradas poucas soluções prontas para uso em soluções, que contemplem o âmbito de autenticação entre os dispositivos da rede.

Entre as soluções encontradas está uma proposta de projeto (sem implementação) disponível no site *Evo things* [Ardiri 2014], que descreve uma possível solução através da utilização do método de autenticação utilizado no protocolo SSL (*Secure Sockets Layer*), ao adaptá-lo de forma que possa ser utilizado em dispositivos IoT. A proposta descreve o uso de um servidor responsável por autenticar os dispositivos e identificá-los entre si, o trabalho também propõe o uso de uma biblioteca de criptografia AES (*Advanced Encryption Standard*) para cifrar os dados transmitidos.

Outra solução encontrada está disponível no próprio site do arduino é a WiFi101 library [Arduino 2016c], que possui diversos métodos de conexão para o uso em shields de interface wireless. Entre os métodos disponíveis pode ser encontrado um método de autenticação que utiliza criptografia, porém essa bilioteca é exclusiva para ser utilizada com o WiFi shield 101, assim não contemplando outros shields mais acessíveis e disponíveis no mercado.

Dentre as soluções encontradas destacam-se, as soluções para implementação de interfaces de login de usuários, no entanto tal solução não contempla o âmbito de autenticidade entre os dispositivos da rede IoT.

## 4. Proposta

Visando o aumento da segurança em redes de dispositivos IoT e principalmente com o intuito de aplicação de regras de autenticação entre os dispositivos, é intencionado o desenvolvimento de um mecanismo capaz de identificar e autenticar tais dispositivos. Ao ser enviada uma nova transmissão o mecanismo é capaz de verificar no servidor se o dispositivo solicitante está autenticado na rede e, desta forma, apto a receber informações. O módulo também deve ser utilizado pelo dispositivo que receberá as informações, pois o mesmo somente deverá ser capaz de receber transmissões de dispositivos autenticados na rede.

O público-alvo deste mecanismo são desenvolvedores que têm como objetivo a construção de soluções de IoT, e garantindo com o uso do mecanismo que seus disposi-

tivos somente receberão dados de outros dispositivos autenticados, evitando dessa forma que suas soluções sejam comprometidas e utilizadas com outros fins.

A estrutura de comunicação entre os dispositivos da rede é composta por um servidor e seus dispositivos, o servidor é responsável pelo cadastro, autenticação e comunicação dos dispositivos.

Para o cadastro dos dispositivos foi implementada uma página web, onde o responsável pela rede irá cadastrar um nome e o endereço de rede do dispositivo. Após o envio do formulário com as informações do dispositivo, são gerados: um número identificador do dispositivo, uma senha e uma chave de criptografia. Essas informações são armazenadas em dois locais: primeiramente no banco de dados do servidor e em seguida em um arquivo de texto que é, após a confirmação do cadastro, disponibilizado para download para o usuário. O arquivo, denominado "auth.txt", precisa ser adicionado a um cartão micro SD que por sua vez deve ser inserido no leitor de cartões do shield ethernet, conforme ilustrado na figura 1.

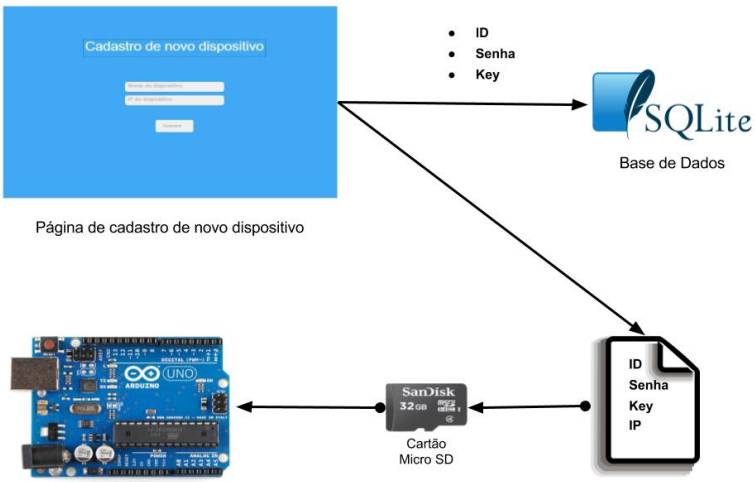



**Figure 1. Cadastro de novo dispositivo**

Após cadastrado no servidor toda a comunicação entre os dispositivos de uma rede irá passar pelo servidor de autenticação, e para que este possa transmitir informações de um membro da rede para outro de forma segura é indispensável o estabelecimento de um canal de comunicação criptografado. Para o estabelecimento desse canal é preciso que o dispositivo passe pelo seguinte processo de autenticação:

1. O arduino irá ler seu número de identificação no arquivo armazenado no cartão micro SD e enviá-lo para o servidor. Este irá gerar um texto aleatório (texto desafio) e enviá-lo para o arduino, conforme descrito na figura 2.
2. O arduino ao receber o texto irá concatená-lo com a senha, que está no arquivo "auth.txt", e calculará o *Hash MD5* do resultado dessa concatenação. Com o *Hash* calculado o dispositivo irá gerar um valor aleatório de 2 bytes denominado "proc". Este é adicionado ao final do *Hash*. O resultado desse processo é então cifrado,

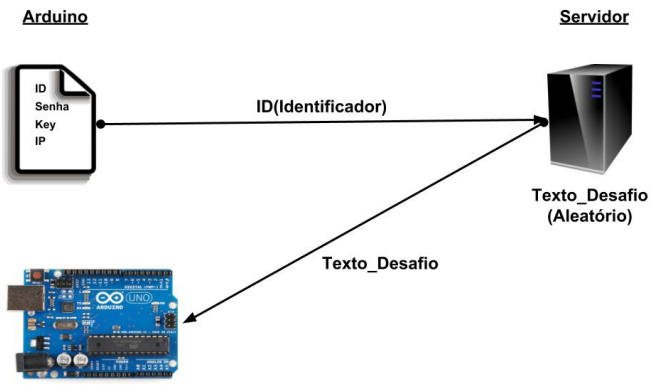



**Figure 2. Autenticação passo 1 - envio do identificador para o servidor e geração do texto desafio.**

utilizando o algoritmo de chave simétrica RC4, com a chave de criptografia gerada no momento do cadastro do dispositivo e armazenada no cartão SD. Em seguida o texto cifrado é enviado para o servidor. Esse processo está demonstrado na figura 3.

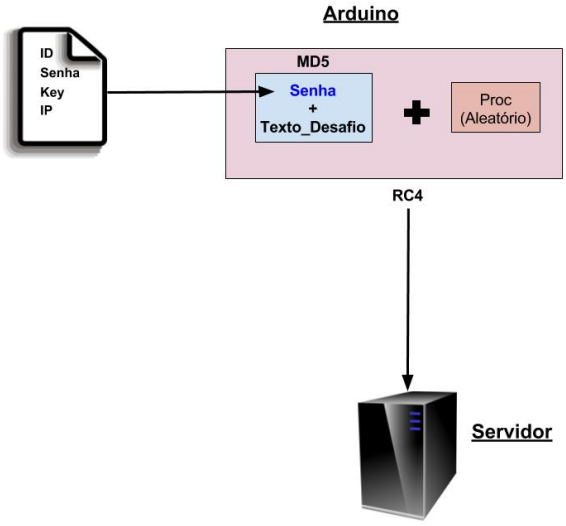



**Figure 3. Autenticação passo 2 - cálculo do *Hash*, geração do ”proc” e envio do texto cifrado para o servidor.**

3. O texto criptografado é decifrado no servidor e uma validação das informações é realizada com base nos registros da base de dados. Caso a validação ocorra com sucesso o dispositivo esta autenticado. O servidor então irá enviar uma mensagem de confirmação cifrada contendo o ”proc” gerado pelo arduino, seguido de um novo valor aleatório gerado pelo servidor denominado “proc_c”, conforme mostrado na figura 4

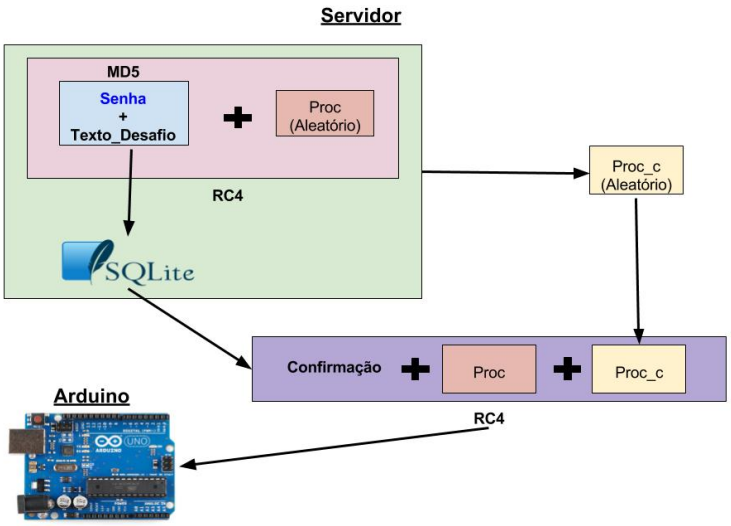



**Figure 4. Autenticação passo 3 - validação dos dados pelo servidor, geração do ”proc_c” e envio da mensagem de confirmação para o arduino.**

4. O dispositivo então decifra a mensagem de confirmação e valida o ”proc”. A partir dessa etapa ambos servidor e dispositivo estão autenticados entre si. A mensagem enviada em seguida contém a informação que deverá ser recebida pelo próximo arduino da rede e deverá conter o ”proc_c” do servidor e um novo ”proc” gerado pelo arduino. Dessa forma é estabelecido um canal criptografado de comunicação, e a cada nova troca de mensagens, entre o servidor e o arduino, os valores de ”proc” e ”proc_c” são alterados, conforme demonstrado na figura 5.

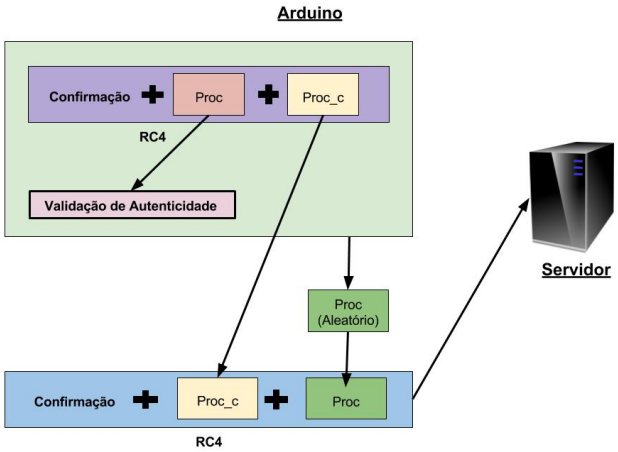



**Figure 5. Autenticação passo 4 - recebimento da mensagem de confirmação, validação do ”proc”, geração de novo ”proc” e envio de mensagem para outro dispositivo através do servidor utilizando o ”proc_c”.**

A implementação dos valores aleatórios ”proc” e ”proc_c” possuem como principal finalidade a prevenção de ataques do tipo *replay* ou *playback*. Esse tipo de ataque

consiste na interceptação e no reenvio de um ou mais pacotes válidos que não pertencem ao atacante, ataques dessa natureza são muito comuns, haja vista que o atacante nem mesmo precisa ter acesso às chaves de criptografia utilizadas no tráfego.

## 5. Tecnologias Empregadas e Metodologia

Inicialmente para o desenvolvimento do mecanismo foi utilizado como base o arduino de modelo *UNO*, ou genuino UNO. Esse modelo de arduino havia sido escolhido devido ao fato de ser popular e acessível, considerando a sua disponibilidade e custo. Entretanto devido a limitações de hardware do modelo *UNO* viu-se a necessidade de migração para o arduino do modelo MEGA 2560, ou Genuino MEGA 2560. Essas limitações ocorreram devido a dois atributos: *Flash Memory* e *SRAM*. A *Flash Memory* ou memória Flash é um tipo de memória não-volátil, ou seja, persiste suas informações mesmo após deixar de ser alimentada por uma fonte de energia. No arduino essa memória é utilizada para armazenar informações não variáveis. A memória *SRAM* (*Static Random Access Memory*) é volátil e perde as informações armazenadas quando sofre descarga de energia. No arduino a SRAM é empregada no armazenamento de informações variáveis utilizadas em tempo de execução. A tabela 1 apresenta uma comparação entre a disponibilidade de ambos os tipos de memória para os modelos de arduino citados.

| | *MEGA* 2560 | *UNO* |
|---|---|---|
| *Flash Memory* | 256KB | 32KB (*ATmega328P*) |
| *SRAM* | 8KB | 2KB |

**Table 1. Tabela de comparação entre os modelos de arduino *UNO* e *MEGA 2560*.**

Cada modelo de placa arduino possui características específicas, entre elas existe a linguagem na qual será possível programar. No desenvolvimento da solução proposta será utilizada a linguagem C++, esta sendo ideal na implementação de soluções onde deseja-se aliar padrões de desenvolvimento, presentes em linguagens de alto nível, com o acesso facilitado aos recursos de hardware do dispositivo, haja vista que C++ é uma linguagem considerada de médio nível [Rongala 2015], ou seja, combina atributos de linguagens de alto e baixo nível.

Como os modelos básicos de arduino atuais não dispnibilizam por padrão uma interface de conexão *ethernet*, foi necessária a utilização de um acessório que acoplado ao arduino permitesse conectá-lo. Para o desenvolvimento deste mecanismo foi definida a utilização do Arduino Ethernet Shield V2, que dispõem de uma interface do tipo RJ45 para conexão com a rede através de um cabo padrão. Outra vantagem na utilização deste shield consiste na disponibilização por padrão de uma interface para cartão do tipo micro-SD, o qual é útil no mecanismo para armazenar informações empregadas no processo de autenticação.

Ao adicionar novos dispositivos à rede é utilizado um servidor, responsável por autenticar todos os dispositivos, além de também hospedar a aplicação de cadastro de dispositivos e por fim é responsável pela comunicação entre os dispositivos.

O servidor de autenticação utiliza o sistema operacional *Ubuntu Server*, que consiste em uma versão da distribuição *Linux Ubuntu* com *kernel* baseado em *UNIX*, essa distribuição possui uso voltado para servidores. Esta versão se diferencia da *desktop* por

possuir menos recursos de usabilidade como interface por exemplo, e mais recursos computacionais por exemplo: formas de acesso, sistema de armazenamento, backup, etc. O *ubuntu server* foi escolhido para este trabalho primeiramente por seu *kernel UNIX* não exigir consideráveis recursos de *hardware*, permitindo assim sua execução mesmo em máquinas de baixo poder computacional. Outra razão para esta escolha reside no fato de ser uma distribuição constantemente atualizada, nesse caso foi escolhida a versão 16.04 *LTS* (*Long Term Support* [Mahnke 2017]), que além de ser a mais atualizada também significa que esta terá suporte garantido pela *Canonical*, empresa responsável pelo suporte ao *Ubuntu*, por pelo menos 5 anos após o seu lançamento [Ubuntupédia 2015].

No processo de autenticação o arduino enviará requisições com informações que confirmarão sua autenticidade, para o recebimento dessas requisições será utilizado um *webservice*, este consiste em uma aplicação que quando executada em um servidor *web* é capaz de receber requisições de outras aplicações, sendo utilizados principalmente para comunicação B2B (*Business-to-Business*).

Para que haja um padrão no envio e recebimento das informações, as mesmas serão trocadas utilizando um protocolo de comunicação definido. Neste projeto a *API* (*Application Programming Interface*, do português "Interface de Programação de Aplicações"), que representa a forma de acesso ao *webservice* foi construída na linguagem de programação Ruby e utilizou-se a arquitetura de comunicação *REST* (*REpresentational State Transfer* [Rouse 2014]). A figura 6 apresenta um exemplo de uma requisição utilizando o protocolo *REST*:


**Figure 6. Exemplo de requisição via método *GET***

*REST* representa um padrão de arquitetura de aplicação que utiliza, para comunicação, os métodos do protocolo *HTTP* (*Hypertext Transfer Protocol* [Lafon 2014]), como por exemplo: *GET*, *POST*, *PUT*, *DELETE*, etc. [Filho 2011] O uso da arquitetura *REST* no projeto deve-se principalmente a facilidade de implementação e pelo seu padrão de comunicação descomplicado e bastante aberto, permitindo que sejam transmitidos dados em diversos formatos conforme a necessidade da aplicação.

O Ruby foi lançada em 1995 por Yukihiro "Matz" Matsumoto, trata-se da união de partes de várias linguagens diferentes como: Perl, Smalltalk e Lisp [on Rails 2017]. É uma linguagem interpretada, de tipagem dinâmica forte e multipradigma, permitindo tanto o paradigma procedural quanto POO (Programação Orientada a Objetos). A linguagem está atualmente em 11º lugar entre as mais utilizadas no mundo, esse crescimento deve-se principalmente ao *framework Ruby on Rails*, este sendo atualmente empregado em grandes sistemas como por exemplo: Github, Airbnb, Hulu e Kickstarter. O Ruby on Rails tem por objetivo, assim como outros frameworks, facilitar e acelerar o desenvolvimento de aplicações, no entanto seu diferencial consiste na geração automática de diversas partes de código que geralmente precisariam ser desenvolvidas pelo programador e que necessitariam de um conhecimento profundo do framework assim como da linguagem em si. No desenvolvimento do mecanismo o Ruby on Rails foi empregado em dois pontos: no desenvolvimento da aplicação de cadastro dos dispositivos assim como na *API* e no *webservice* de autenticação.

Para cifrar e decifrar os textos requisitados na autenticação foi definido o uso do algoritmo RC4. Criado em 1967 por Ronald Rivest o algoritmo RC4 ou ARC4 consiste em um algoritmo de chave simétrica de cifra de fluxo [Stallings 2014], ou seja, o processo de cifragem e decifragem independem do tamanho da chave e as operações são orientadas a byte. O RC4 é bastante difundido principalmente na área de comunicação e transmissão de dados, sendo utilizado em diversos padrões conhecidos de transmissão como: *SSL/TLS(Secure Sockets Layer/Transport Layer Security)*, comunicação entre navegadores *WEB* e servidores e também no protocolo *WEP(Wired Equivalent Privacy)*. O RC4 desempenha a tarefa fundamental de garantir a confidencialidade na transmissão das informações no mecanismo, e demonstrou um bom desempenho ao ser implementado na plataforma arduino, sendo executado rapidamente considerando as limitações da plataforma. Um problema enfrentado com a cifragem utilizando RC4 foi o uso pelo algoritmo de carateres especiais, esse problema foi tratado através do uso de codificação *Base64* no texto cifrado.

Na implementação do mecanismo de autenticação foi inicialmente necessário estudar o funcionamento da comunicação entre o arduino e o servidor de autenticação através do shield ethernet, e então desenvolver uma solução de comunicação entre ambos. Também foi preciso estudar o funcionamento dos métodos da biblioteca de criptografia e validar se os mesmos estão de acordo para serem utilizados no mecanismo. Outra questão que foi analizada é se o arduino UNO será capaz de executar os métodos da biblioteca de criptografia, pois o processamento desse modelo de arduino é considerado mediano em relação a modelos mais robustos.

Após os estudos e validações de compatibilidade, utilizando-se dos conhecimentos adquiridos foi desenvolvido o mecanismo de autenticação dos dispositivos.

## 6. Cenário de Testes e Resultados Obtidos

O cenário de testes foi construído e planejado visando a realização de um teste do tipo ”caixa preta”, onde os ativos simularam o processo de autenticação e envio de uma mensagem entre dois dispositivos arduino, para tal foram empregados os seguintes ativos no cenário: um atacante (executor do teste), dois dispositivos arduino e um servidor de autenticação.

Para validar o envio da mensagem e garatir proteção contra os ataques destacados anteriormente, os testes foram realizados através da execução de um software na rede capaz de analisar o tráfego dos dados, neste teste foi usado o *Wireshark*. O *Wireshark* é um *Sniffer*, que consiste em uma aplicação capaz de ”capturar” o tráfego de uma rede, fornecendo neste caso a opção de filtragem de dados, organizando assim os pacotes obtidos por: porta, protocolo, endereço de rede, aplicação, etc. A escolha do *Wireshark* como software de análise de tráfego para os testes deve-se principalmente a sua popularidade, haja vista que é muito empregado em análises de segurança de redes. A figura 7 apresenta uma captura de pacote do mecanismo, é possível ver que na requisição obtida é apresentado o texto cifrado assim como o identificador do dispositivo.

Considerando que o *webservice* representa uma grande parte do mecanismo, este também precisou passar por testes de aceitação, para tal foi necessário pesquisar por um software capaz de realizar os testes da *API*, ou seja, capaz de simular um dispositivo na rede que tenta se autenticar através do envio das requisições para a *API* de autenticação.

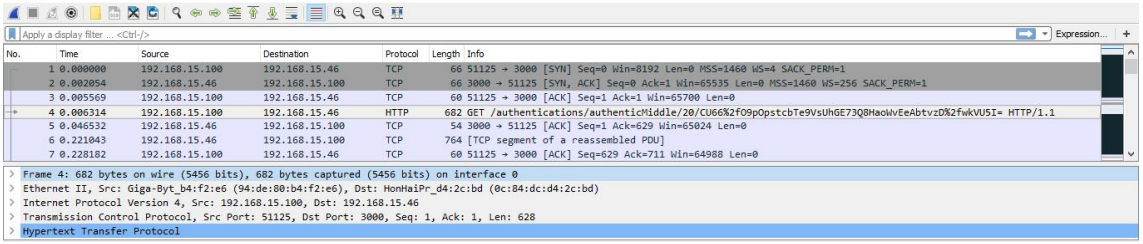



**Figure 7. Exemplo de captura de pacote no software *Wireshark*.**

O software escolhido para este teste foi o *Postman*, este sendo capaz de simular um cliente com o intuito de consumir os serviços disponibilizados através da *API*. A figura 8 demonstra uma tentativa de autenticação que resultou em falha pois foi re-enviada para o *web-service* uma requisição, o erro ocorreu pois o ”proc” fornecido já havia sido utilizado anteriormente.

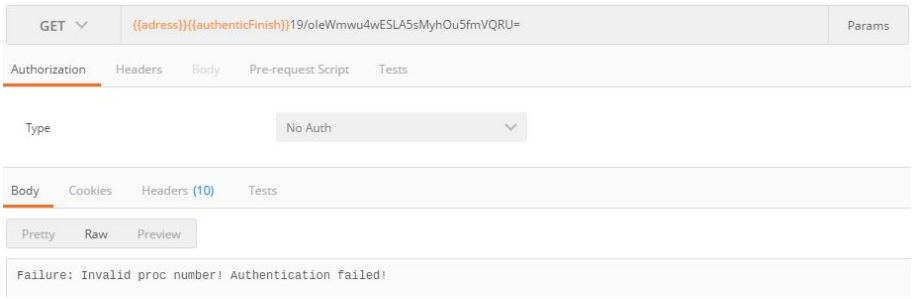



**Figure 8. Exemplo de requisição no software Postman.**

Este trabalho não contempla a segurança e integridade física dos dispositivos componentes da rede, como exemplo no caso da subtração do cartão microSD e seu uso em outro dispositivo poderia ocasionar em uma vulnerabilidade. Considerando que a aplicação de cadastro assim como o servidor em si constituem partes do mecanismo, ambos normalmente passariam por testes e validações de segurança, no entanto este trabalho desconsiderada a gestão de segurança e de acessos do servidor e da aplicação de cadastro, como: em caso de erros de configuração, a deprecação de bibliotecas e módulos utilizados no mecanismo, etc.

A figura 9 apresenta os logs de monitoramento de um arduino, nele é possível ver os passos de um processo de autenticação bem sucedido.

A figura 10 apresenta o terminal de execução do *webservice*, onde é possível ver uma requisição de início do processo de autenticação. Também é mostrado o texto desafio (*”Challenge”*) gerado para aquele dispositivo cadastrado.

A figura 11 apresenta o código utilizado para validar o *Hash* contendo a senha e o texto desafio enviados pelo arduino. Após a validação é gerado o ”proc_c” do servidor e a mensagem de confirmação da autenticação é retornada para o dispositivo.

O projeto resultou em uma solução capaz de autenticar o envio e o recebimento de informações entre os dispositivos de uma rede, servindo como mediador das transmissões. O mecanismo foi concebido de forma a ser utilizado por desenvolvedores de soluções IoT, fornecendo assim maior segurança em suas aplicações.


**Figure 9. Processo de autenticação de dispositivo.**

**Figure 10. Servidor recebendo requisição de início de autenticação.**

## 7. Conclusão

Através dos estudos realizados e apresentados, conclui-se que a IoT é uma área em expansão e que logo a população passará a depender do uso de tais soluções. Todavia a dependência dessas tecnologias para a realização de tarefas fundamentais torna a questão de garantia de segurança nestas redes um assunto relevante e de suma importância, portanto novos estudos são necessários e novas soluções precisam ser desenvolvidas e difundidas.

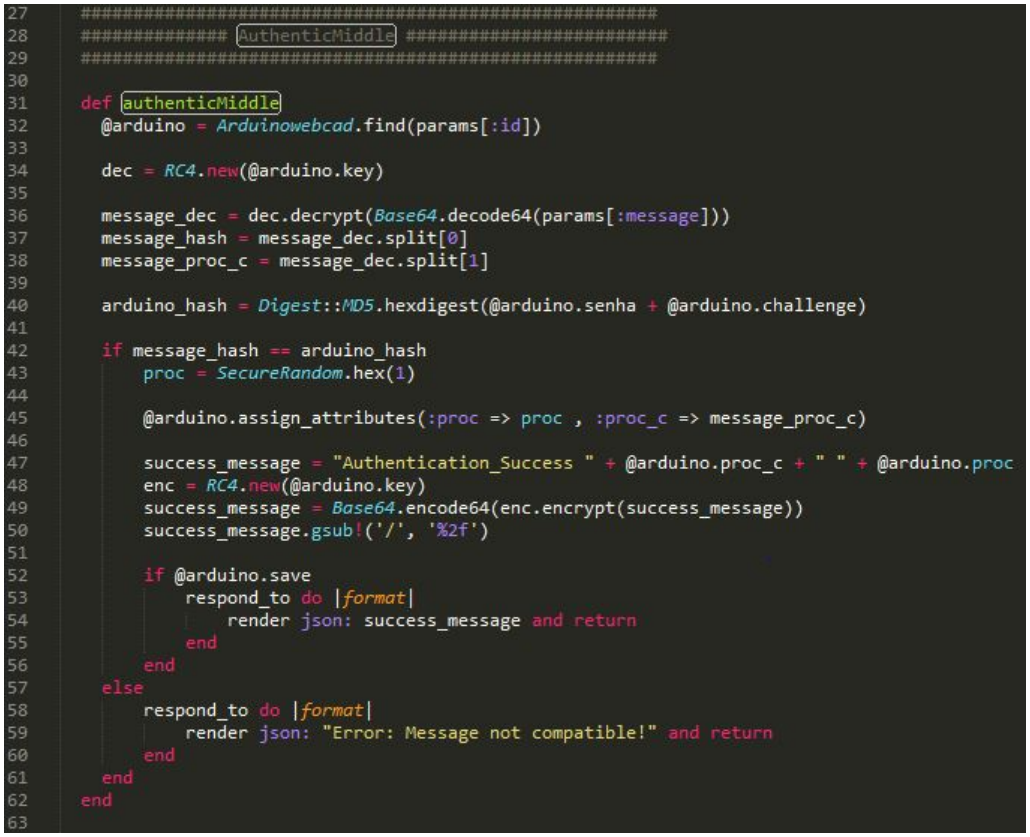
A segurança da informação possui diversos atributos, entre eles é possível identificar a autenticação como um dos mais importantes e impactantes. Haja vista o caso de um ativo que não esteja devidamente identificado em um meio, podendo este vir a provocar danos ao meio ou até mesmo o vazamento de informações sensíveis.

O ataque ocorrido à empresa Dyn comprova a necessidade de estudos mais aprofundados na área de segurança em soluções IoT. O ataque ocasionou em danos á disponibilidade dos serviços, outro importante pilar da segurança computacional, o que acarreta em prejuízos para as empresas que hospedam serviços com a Dyn.

Durante o desenvolvimento e a execução dos testes do mecanismo alguns pontos de melhoramento do software foram levantados, levando-se em consideração o desempenho, a qualidade do código desenvolvido e a visão de um produto final.

O mecanismo de autenticação apesar de funcional pode de certa forma ser considerado inviável, do ponto de vista de consumo de memória tanto *Flash* quanto *SRAM*. No entanto existem técnicas de melhoramento de código, como a compressão de código por exemplo, que são capazes de melhorar significativamente o consumo de recursos de hardware de aplicações como essa, tornando-a assim passível de uso em dispositivos de baixo poder de processamento.

Outra oportunidade de melhoramento encontra-se na substituição do algoritmo de criptografia, de RC4 para AES. Apesar de considerado bastante seguro o RC4 já foi



```
27  ##################################################
28  ############# AuthenticMiddle ####################
29  ##################################################
30
31  def authenticMiddle
32    @arduino = Arduinowebcad.find(params[:id])
33
34    dec = RC4.new(@arduino.key)
35
36    message_dec = dec.decrypt(Base64.decode64(params[:message]))
37    message_hash = message_dec.split[0]
38    message_proc_c = message_dec.split[1]
39
40    arduino_hash = Digest::MD5.hexdigest(@arduino.senha + @arduino.challenge)
41
42    if message_hash == arduino_hash
43        proc = SecureRandom.hex(1)
44
45        @arduino.assign_attributes(:proc => proc , :proc_c => message_proc_c)
46
47        success_message = "Authentication_Success " + @arduino.proc_c + " " + @arduino.proc
48        enc = RC4.new(@arduino.key)
49        success_message = Base64.encode64(enc.encrypt(success_message))
50        success_message.gsub!('/', '%2f')
51
52        if @arduino.save
53            respond_to do |format|
54                render json: success_message and return
55            end
56        end
57    else
58        respond_to do |format|
59            render json: "Error: Message not compatible!" and return
60        end
61    end
62  end
63
```

**Figure 11. Método ruby para realização do 2º passo da autenticação.**

substituído em diversos softwares e protocolos críticos pelo algoritmo AES, devido a este ser mais difícil de ser decifrado por um atacante.

Considerando que este trabalho objetiva primariamente a garatia de autenticidade de dispositivos, e todos os atributos componentes da segurança da informação fica clara a necessidade de novas soluções que contemplem esses atributos. Para tal uma boa solução seria o desenvolvimento de um *famework* visando a disponibilização de métodos e mecanismos implementados com as boas práticas de segurança, dessa forma os desenvolvedores de aplicações IoT poderão, de certa forma, abstrair a segurança e concentrar esforços no desenvolvimento da solução.

O desafio deste trabalho foi o de desenvolver um mecanismo de autenticação capaz de ampliar a segurança de soluções de IoT que utilizam a arquitetura Arduino. Os limites de poder de processamento e memória do Arduino acabam delimitando o conjunto de possíveis soluções para este mecanismo. Considera-se que o mecanismo desenvolvido atinge os objetivos propostos e apresenta como contribuição uma possibilidade de uso e estudo para desenvolvedores de aplicações IoT.

## References


Alecrim, E. (2016). O que é Internet das Coisas (Internet of Things)? [Online: http://www.infowester.com/iot.php Acessado em 18/11/2016].

Ardiri, A. (2014). Is it possible to secure micro-controllers used within IoT? [Online: https://evothings.com/is-it-possible-to-secure-micro-controllers-used-within-iot/

Acessado em 5/10/2016].

Arduino (2016a). Libraries. [Online: https://www.arduino.cc/en/Reference/Libraries Acessado em 17/11/2016].

Arduino (2016b). What is Arduino? [Online: https://www.arduino.cc/en/Guide/Introduction Acessado em 08/09/2016].

Arduino (2016c). WiFi101 library. [Online: https://www.arduino.cc/en/Reference/WiFi101 Acessado em 28/10/2016].

Filho, C. A. (2011). Qual a diferença entre REST e SOAP? [Online: https://thoughtsasaservice.wordpress.com/2011/03/17/qual-a-diferenca-entre-rest-e-soap/ Acessado em 21/05/2017].

Gartner (2015). Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. [Online: http://www.gartner.com/newsroom/id/3165317 Acessado em 10/11/2016].

IT, V. (2015). Segurança da Informação: Integridade, Confidencialidade e Disponibilidade! [Online: http://blog.valit.com.br/seguranca-da-informacao-integridade-confidencialidade-e-disponibilidade/ Acessado em 27/11/2016].

Kaspersky (2017). Kaspersky Labs. [Online:https://www.kaspersky.com/about Acessado em 22/06/2017].

Kuskov, V. (2017). Honeypots and the Internet of Things. [Online:https://securelist.com/honeypots-and-the-internet-of-things/78751/ Acessado em 22/06/2017].

Lafon, Y. (2014). HTTP - Hypertext Transfer Protocol. [Online: https://www.w3.org/Protocols/ Acessado em 21/05/2017].

Lemos, M. (2015). Massimo Banzi – Makers que você precisa conhecer. [Online: http://blog.fazedores.com/massimo-banzi-makers-que-voce-precisa-conhecer/ Acessado em 08/09/2016].

Loshin, P. (2016). Details emerging on Dyn DNS DDoS attack, Mirai IoT botnet. [Online: http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet Acessado em 8/11/2016].

Mahnke, P. (2017). LTS. [Online:https://wiki.ubuntu.com/LTS Acessado em 21/05/2017].

Miessler, D. (2017). OWASP Internet of Things Project. [Online:https://www.owasp.org/index.php/OWASPAcessado em 22/06/2017].

on Rails, R. (2017). Ruby on Rails. [Online:http://rubyonrails.org/].

OWASP (2017). Honeypots and the Internet of Things. [https://www.owasp.org/ Acessado em 22/06/2017].

Rongala, A. (2015). Benefits of C / C++ over Other Programming Languages. [Online: https://www.invensis.net/blog/it/benefits-of-c-c-plus-plus-over-other-programming-languages Acessado em 17/11/2016].

Rouse, M. (2014). REST (representational state transfer). [Online: http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer Acessado em 21/05/2017].

Stallings, W. (2014). *Criptografia e segurança de redes Princípios e práticas*, volume 4. PEARSON Prentice Hall.

Sun, L. (2016). These 2 tech companies are working to protect the Internet of Things. [Online: http://uk.businessinsider.com/these-2-tech-companies-are-working-to-protect-the-internet-of-things-2016-11 Acessado em 08/11/2016].

Ubuntupédia (2015). Ubuntu Server. [Online:http://ubuntupedia.info/index.php/UbuntuServer Acessado em 21/05/2017].