PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIAS

ROGÉRIO NUNES DE FREITAS

IMPACTO NO DESEMPENHO DE RSSF COM DIFERENTES TÉCNICAS DE CIFRAGEM

CAMPINAS

2018

ROGÉRIO NUNES DE FREITAS

IMPACTO NO DESEMPENHO DE RSSF COM DIFERENTES TÉCNICAS DE CIFRAGEM

Dissertação apresentada ao Programa de Pós Graduação Stricto Sensu em Engenharia Elétrica do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Omar Carvalho Branquinho.

PUC-CAMPINAS

2018

Ficha Catalográfica

Elaborada pelo Sistema de Bibliotecas e

Informação - SBI - PUC-Campinas

ROGÉRIO NUNES DE FREITAS

IMPACTO NO DESEMPENHO DE RSSF COM DIFERENTES TÉCNICAS DE CIFRAGEM

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências Exatas, Ambientais e de Tecnologias da Pontifícia Universidade Católica de Campinas como requisito parcial para obtenção do título de Mestre em Gestão de Redes de Telecomunicações.
Área de Concentração: Engenharia Elétrica.
Orientador: Prof. Dr. Omar Carvalho Branquinho

Dissertação defendida e aprovada em 28 de novembro de 2018 pela Comissão Examinadora constituída dos seguintes professores:

Prof. Dr. Omar Carvalho Branquinho
Orientador da Dissertação e Presidente da Comissão Examinadora
Pontifícia Universidade Católica de Campinas

Profa. Dra. Indayara Bertoldi Martins
Pontifícia Universidade Católica de Campinas

Prof. Dr. Paulo Cardieri
Universidade Estadual de Campinas

Dedico este trabalho a minha família, que sempre esteve presente em todos os momentos da minha vida.

AGRADECIMENTOS

Primeiramente a Deus,

Por ter me dado saúde e forças até este presente momento.

A minha família,

Em especial minha mãe Cleusa e minha esposa Ligia, pelo apoio incondicional e pelo incentivo nos momentos mais difíceis.

Ao Prof. Dr. Omar Carvalho Branquinho,

Incentivador, guia e amigo nesta parte da Jornada.

Ao colega e amigo André Luis Boni Déo,

Por todo apoio, dicas e companheirismo nesta trajetória.

À Pontifícia Universidade Católica de Campinas,

Pela conceção da bolsa no Programa de Mestrado, sem o qual este trabalho não poderia ter sido concretizado.

À FATEC – Faculdade de Tecnologia de Americana,

Em especial aos colegas de trabalho, que de uma forma ou de outra contribuiram para a elaboração deste trabalho.

Aos colegas e companheiros da turma,

Pelo apoio e toda solidariedade

“A experiencia é o nome que damos aos nossos erros”

(Oscar Wilde)

RESUMO

FREITAS, Rogério Nunes. Cifragem para Redes de Sensores Sem Fio. 2018. 124 pgs. Dissertação (Mestrado profissional em Gestão de Redes de Telecomunicações) – Pontifícia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Campinas, 2018.

A utilização de Redes de Sensores Sem Fio vem crescendo nos últimos anos. Isto se deve à sua ampla gama de aplicações, principalmente na Internet das Coisas. Nestas redes, existe a necessidade de implantar soluções de segurança que consumam poucos recursos de seus elementos. Este trabalho apresenta os principais tipos de ataques e vulnerabilidades existentes nas Redes de Sensores Sem Fio, além de alguns mecanismos de segurança e proteção já existentes neste tipo de rede. A proposta deste trabalho, é apresentar dois métodos de cifragem para a segurança dos dados nas Redes de Sensores Sem Fio. Ambas as estratégias visam a proteção de todo o pacote transmitido entre os elementos da rede, para que ela se torne confiável. Nos dois métodos desenvolvidos, uma senha é utilizada para auxiliar na cifragem das informações antes da transmissão. Para avaliar o impacto na rede, foram realizados testes com algumas métricas existentes nas Redes de Sensores Sem Fio, como por exemplo: taxa de erro do pacote, tempo de processamento do Nó Sensor e tempo de resposta da Rede de Sensores Sem Fio. Os experimentos avaliam os impactos dos métodos de cifragem desenvolvidos, em relação ao desempenho da rede. Para testar a eficácia dos métodos, em ambos os casos, foi desenvolvido um elemento capaz de capturar pacotes de comunicação entre as entidades da rede, com o intuito de emular um possível invasor. A implementação atendeu ao que foi proposto, do ponto de vista da Segurança da Informação, uma vez que foi possível realizar as cifragens das informações por meio da Rede de Sensores Sem Fio. Os resultados mostram uma estabilidade dos métodos de cifragem implementados. Os testes realizados demonstram que houve um aumento no processamento nos elementos da rede, porém, este acréscimo de tempo é aceitável no âmbito das Redes de Sensores Sem Fio. Os testes demonstram ainda, que não houve perda de pacotes com a implementação dos métodos de cifragem.

Termos de indexação: Internet das Coisas; Redes de Sensores Sem Fio; Cifragem; Segurança.

ABSTRACT

FREITAS, Rogério Nunes. Encryption for Wireless Sensor Networks 2018. 124 pgs. Dissertation (Master in Telecommunication Network Management) - Pontificia Universidade Católica de Campinas, Centro de Ciências Exatas, Ambientais e de Tecnologias, Campinas, 2018.

The use of Wireless Sensor Networks has been growing over the last years. This is due to its wide range of applications in the Internet of Things. In these networks, there is a need to deploy security solutions that consume few resources from their elements. This work presents the main types of attacks and vulnerabilities in Wireless Sensor Networks, as well as some security and protection mechanisms that already exist in this type of network. The proposal of this work is to present two encryption methods in order to perform the data security in Wireless Sensor Networks. Both strategies aim to protect the whole package, transmitted among the elements of the network, so that it becomes reliable. In both methods, a password is used in order to help in the encryption of information before the transmission. To evaluate the impact on the network, tests were performed with some important metrics from the Wireless Sensor Networks, such as: packet error rate, Sensor Node processing time and packet response time of the Wireless Sensor Networks. The experiments evaluate the impacts of the developed encryption methods related to the network performance. To test the effectiveness of the methods, in both cases, an element was developed, capable of capturing communication packets among entities of the network, in order to emulate a possible invader. The implementation complied with what was proposed from the point of view of Information Security, since it was possible to carry out the encryptions of the information through the Network of Wireless Sensors. The results show a stability of the implemented encryption methods. The tests showed that there was an increase in the processing of the elements of the network, but this increase of time is acceptable in the Wireless Sensor Networks. The tests also show that there was no loss of packets with the implementation of the encryption methods

Index terms: Internet of Things; Wireless Sensor Networks; Encryption; Safe.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RSSF	=	Redes de Sensores Sem Fio
NS	=	Nó Sensor
IoT	=	*Internet of Things*
ERB	=	Estação Rádio Base
RAM	=	*Random Access Memory*
ROM	=	*Read Only Memory*
EEPRON	=	*Electrically Erasable Programmable Read Only Memory*
IoT-PM	=	*Internet of Things Proxy Manager*
ISO	=	*International Standard Organization*
ITU	=	*International Telecommunication Union*
OSI	=	*Open System Interconnection*
DES	=	*Data Encryption Standard*
IBM	=	*International Business Machines*
MD5	=	*Message-Digest algorithm* 5
SHA-1	=	*Secure Hash Algorithm* 1
XOR	=	*Or exclusive*
OTP	=	*One Time Pad*
MAC	=	*Media Access Control*
I/O	=	*Input / Output*
AD	=	*Analogic Digital*
USB	=	*Universal Serial Bus*
HDMI	=	*High Definition Multimedia Interface*
SD	=	*Secure Digital*
ANATEL	=	Agência Nacional de Telecomunicações

MHz	=	*mega hertz*
FSK	=	*Frequency shift keying*
Kbps	=	Quilo bit por segundo
RS232	=	*Recommended Standard* 232
V	=	*Volts*
MB	=	*Megabyte*
GB	=	*Gigabyte*
dBm	=	*Decibel Miliwa*

SUMÁRIO

1. INTRODUÇÃO

A Internet das Coisas (IoT) é uma revolução tecnológica que proporciona interligação e comunicação de objetos de forma inteligente à Internet. Estes objetos, são geralmente dispositivos eletrônicos baseados em circuito integrado que podem enviar dados através de uma rede, podendo ser esta uma Rede de Sensores Sem Fio (RSSF) (GANESHAN, 2017).

As RSSF constituem em vários componentes chamados de Nós Sensores (NS), conectados entre si, sem a utilização de cabos. Esta tecnologia tem se desenvolvido muito nos últimos anos, isto se deve não só a enorme gama de ambientes em que uma RSSF pode ser útil, mas também ao custo reduzido deste tipo de solução, em comparação a outros tipos de tecnologias, como as redes Ethernet, ou as redes de celular, por exemplo.

Os equipamentos existentes nas RSSF são capazes de monitorar grandezas, como temperatura, ou umidade por exemplo, e encaminhar as informações coletadas para um computador ou até mesmo para Internet. As RSSF diferem de redes de computadores tradicionais em vários aspectos (LOUREIRO et al., 2003). Normalmente estas redes possuem um grande número de NS distribuídos e devem possuir mecanismos de autoconfiguração e adaptação devido a problemas como falhas de comunicação ou perda do NS (PHOHA; LAPORTA; GRIFFIN, [s.d.]).

As RSSF têm sido utilizadas nos mais diversos setores, alguns exemplos práticos de aplicações são exibidos a seguir (YICK; MUKHERJEE; GHOSAL, 2008).

- **Meio ambiente:** rastreamento de pequenos animais, detecção de incêndios em florestas, estudo de poluentes, etc;
- **Saúde:** monitoramento de pacientes, administração de drogas, rastreamento de pacientes e doutores em hospitais, etc;
- **Residenciais:** monitoramento do consumo de água, interligação de eletrodomésticos, casas inteligentes, etc;
- **Industriais:** diagnósticos das máquinas, rastreamento de veículos, etc.

As RSSF utilizam comunicação sem fio entre seus componentes. Neste tipo de comunicação, o modo de transmissão utilizado é *broadcast*, onde cada NS se comunica

com os NS em seu entorno simultaneamente. Desta forma, a rede fica mais vulnerável, se tornando assim, mais susceptível à ação de intrusos, pois estes podem facilmente escutar, interceptar e alterar os dados que trafegam por ela (PINHEIRO; MAGALHÃES, 2014). A segurança dos dados que são transmitidos entre os NS é de grande importância para o funcionamento eficiente e seguro da rede, uma vez que os dados inconsistentes podem gerar resultados imprecisos pelos NS. Para que o envio das informações feitos pelos elementos de uma RSSF tenha segurança, de forma a garantir a integridade, confidencialidade e disponibilidade dos mesmos, são necessários novos mecanismos e segurança para proteger estes dados do início ao fim da transmissão.

Assim como ocorre em outras redes, como nas redes de computadores por exemplo, existe nas RSSF, a necessidade de proteger as informações que são transmitidas entre seus componentes. Com relação à segurança da RSSF em si, um invasor pode prejudicar o bom funcionamento da rede ou até mesmo deixa-la inoperante. Além disso, como já citado, as RSSF utilizam comunicação *sem fio,* por este motivo, em algumas situações, não é necessário a ação de um atacante para que haja prejuízo ao funcionamento da rede. Se algum dispositivo que atue na mesma faixa de frequência, estiver próximo aos componentes da RSSF, o mesmo pode causar interferência no sinal, o que pode acarretar perda de desempenho da rede (TABRIZI; IBRAHIM, 2016).

Entretanto, em algumas ocasiões, um invasor pode capturar as informações interceptadas de forma transparente para a RSSF, ou seja, sem necessariamente afetar seu desempenho. Nestas situações, o objetivo do atacante é simplesmente se beneficiar, obtendo alguma informação transmitida pelos NS (PRANATA; ATHAUDA; SKINNER, 2012).

1.1. Motivação

As RSSF geralmente não são providas de segurança (ASSIS et al., 2015). Existe uma escassez de soluções de segurança para RSSF disponíveis, pois, fatores como as baixas capacidades de memória, processamento e energia limitam muito esta questão. Sendo assim, implementações complexas de segurança são inviáveis pelo fato da sobrecarga que elas poderiam trazer na RSSF (PINHEIRO; MAGALHÃES, 2014), causando uma possível lentidão no tráfego de informações.

Devido aos problemas citados, surge a necessidade de novas soluções e tecnologias para prover segurança para os dados transmitidos por meio da RSSF, de maneira que as informações sejam protegidas contra possíveis invasores ou ainda

contra possíveis fenômenos que eventualmente, possam causar algum mau funcionamento da rede.

Tendo este cenário em vista, foram propostos neste trabalho métodos de cifragem para as RSSF. Esta técnica, consiste em transformar a informação original para outra ilegível, de forma que esta possa ser conhecida apenas por seu destinatário, o que a torna difícil de ser lida por alguém não autorizado. Assim sendo, apenas o receptor da mensagem pode ler a informação com facilidade, ou seja, decifrada (STALLINGS, 2008).

1.2. Objetivos

O objetivo principal deste trabalho é propor soluções de cifragem para proteger as informações nas RSSF, com o objetivo de proteger não só os dados, mas a rede em si. Os métodos de cifragem serão capazes de proteger as informações transmitidas por meio de uma RSSF. As estratégias desenvolvidas irão atuar não apenas na segurança das informações geradas pelos elementos da rede, como também das informações inerentes às camadas da pilha de protocolos, ou seja, a proteção será realizada em todo o pacote transmitido.

Outro objetivo é abordar os atributos básicos necessários para prover segurança em uma RSSF, que são (MORAES, 2010):

- **Integridade dos dados**: Visa garantir que a informação permaneça íntegra, o que significa que ela não sofra nenhuma espécie de modificação durante a sua transmissão ou armazenamento;
- **Confidencialidade dos dados:** visa garantir que apenas os elementos que possuem interesse na informação a acessem;
- **Autenticidade**: visa garantir que a informação que chega ao destino, é a informação original que foi enviada pelo remetente;
- **Disponibilidade:** visa garantir que a informação esteja disponível e acessível na rede o maior tempo possível. A informação deve estar sempre disponível e ser obtiva sempre que for necessária, para quem precisar dela.

1.3. Objetivos específicos

O projeto de pesquisa terá como objetivos específicos:

- Estudar os mecanismos de segurança das informações em RSSF;
- Propor mecanismos eficientes de cifragem que funcionem nas RSSF sem prejudicar de forma considerável seu funcionamento;
- Descrever os mecanismos propostos, relatando os métodos utilizados e testes realizados;
- Demonstrar a viabilidade dos mecanismos propostos e possíveis impactos no desempenho das RSSF;

1.4. Organização do Trabalho

O restante deste trabalho é organizado da seguinte forma:

- **Capitulo 2:** Apresenta os conceitos relacionados a uma RSSF, abordando seus principais elementos, funcionalidade e pontos fortes e fracos;
- **Capitulo 3:** Apresenta conceitos relacionados à segurança da informação, no âmbito geral e nas RSSF, abordando os principais tipos de ataques em que elas estão sujeitas;
- **Capitulo 4:** Apresenta trabalhos relacionados, encontrados na literatura;
- **Capitulo 5**: Apresenta as propostas de cifragem, desenvolvidas para RSSF;
- **Capitulo 6:** Apresenta a implementação das propostas deste trabalho com materiais e métodos utilizados;
- **Capitulo 7:** Apresenta os resultados obtidos, além um comparativo das métricas com a utilização dos dois métodos desenvolvidos no trabalho, bem como uma visão dos testes futuros a serem realizados;
- **Capitulo 8**: Apresenta as considerações finais, através de uma conclusão e perspectivas de trabalhos futuros.

2. INTERNET DAS COISAS E AS REDES DE SENSORES SEM FIO

Uma rede nada mais é do que dois ou mais dispositivos conectados (TANENBAUM, 2011). Há pouco tempo, esses dispositivos eram basicamente computadores de mesa. No entanto, cada vez mais outros dispositivos finais como televisão, laptops, telefones celulares, sistemas internos de segurança, estão sendo conectados em rede. O termo redes de computadores começa a soar um tanto desatualizado, dados os muitos equipamentos não tradicionais que estão sendo ligados na Internet (JAMES F. KUROSE; KEITH W. ROSS, 2010), que é uma rede de computadores ou telecomunicações que interconecta milhares de dispositivos computacionais ao redor do mundo.

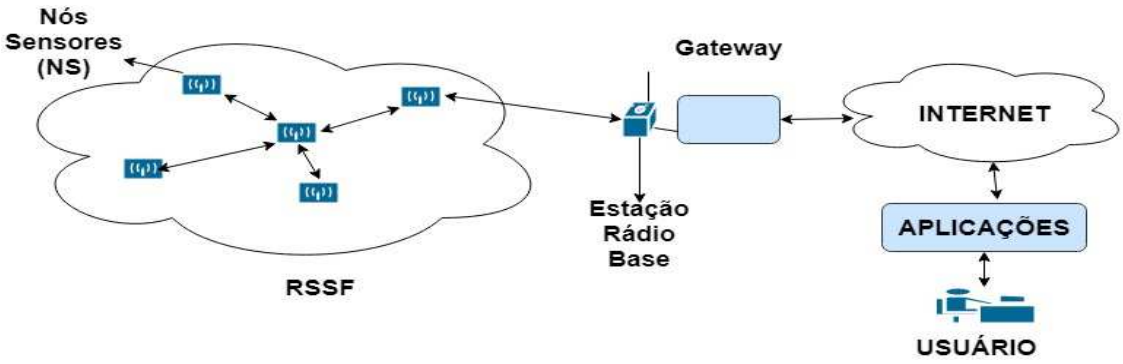
A proposta da IoT é ligar todas as coisas à Internet (NUNO et al., 2016), de sofisticados equipamentos até os mais simples, para que estes objetos possam se comunicar entre si e entre os usuários e consumidores, a fim de gerar informações a serem usadas nas mais distintas funções (TABRIZI; IBRAHIM, 2016). Estima-se que até 2020 existam mais de 50 bilhões de dispositivos conectados na internet (FEKI et al., 2013), tais como comidas, roupas, móveis, papéis, monumentos, veículos automotivos, obras de arte, etc; estarão conectados à Internet através de tecnologias pervasivas, se comunicando e colaborando entre si.

No contexto da IoT, estão inseridas as RSSF. Estas redes estão em amplo crescimento atualmente (GANESHAN, 2017). Isto se deve dentre outros fatores, aos avanços da microeletrônica, que estimularam o desenvolvimento de pequenos NS, que foram acoplados em dispositivos com a comunicação sem fio, com pouca capacidade de processamento e limitados recursos de computação (LOUREIRO et al., 2003). O conjunto desses NS trabalhando cooperativamente formam uma RSSF.

2.1. Principais componentes de uma RSSF

As RSSF tradicionalmente, são compostas de alguns componentes principais: NS, Estação Rádio Base (ERB) e Gateway. Na Figura 1, são mostrados os componentes de um RSSF.

Figura 1- Modelo de RSSF com seus componentes principais



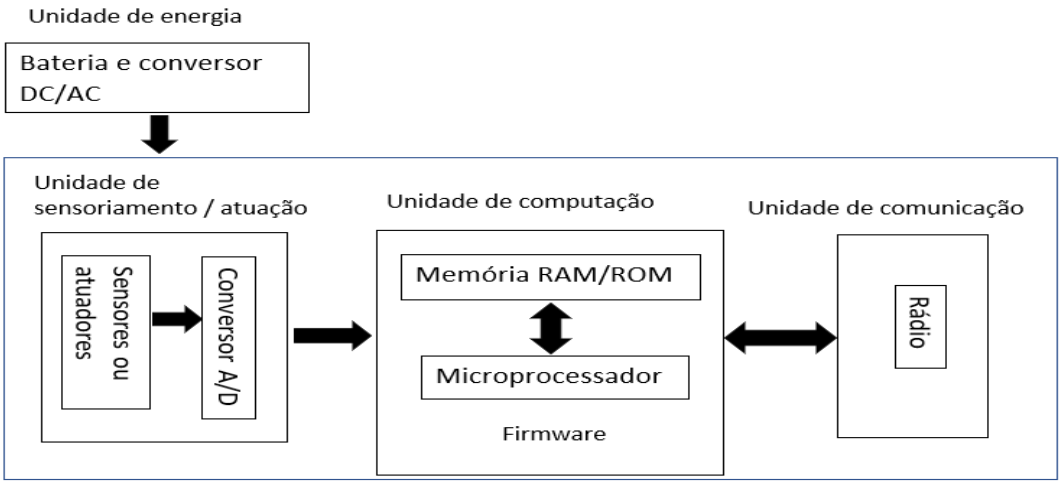
Fonte: Adaptado de (IAN F. AKYILDIZ, 2010)

2.1.1. Nó Sensor

Um NS é o componente que armazena um ou mais sensores, responsáveis por coletar grandezas do meio ambiente, tais como: temperatura, umidade, luminosidade, dentre outras. Além de coletar estas informações e entrega-las à ERB, os NS também possuem a função de roteamento (IAN F. AKYILDIZ, 2010), uma vez que muitas vezes, para que um NS leve as informações para a ERB, é necessário passar por outros NS.

Um NS é composto por memória, processador, atuador, dispositivo de comunicação e fonte e energia (VIEIRA et al., 2003). Na Figura 2, são mostrados os principais componentes de um NS.

Figura 2 - Modelo de NS com seus componentes principais



Fonte: Adaptado de (VIEIRA et al., 2003)

A seguir, são detalhados cada um dos componentes de um NS.

- **Unidade de Energia:** a fonte de energia em um NS é um fator de importância, pois ela mantém todos os outros componentes em funcionamento. Dependendo da localização, a alimentação dos NS se torna difícil pelo fato de não estarem próximos de alguma fonte de alimentação de energia. Um bom exemplo, seria uma RSSF para monitorar uma floresta (KARL; WILLIG, 2005);

- **Unidade de sensoriamento e atuação:** Esta unidade é composta de sensores e(ou) atuadores e um conversor analógico/digital. O atuador, cria uma alteração física no meio ambiente. Por exemplo, um NS pode abrir ou fechar um relé, permitindo a passagem e/ou configuração de um valor, assim acionando algum outro mecanismo, como um motor ou uma lâmpada. Já um sensor é um dispositivo físico capaz de ler uma grandeza do mundo físico (formato analógico) e convertê-la para o mundo virtual (formato digital), por intermédio de um conversor analógico-digital. Já o conversor analógico-digital (A/D) converte a saída de um sensor, que é contínua, ou seja, sinal analógica e um sinal digital (DARGIE; POELLABAUER, 2010).

- **Unidade de Computação:** Esta unidade é composta por um firmware, que contém um microprocessador e memórias RAM e ROM. O firmware é o *software* que gerência as ações de um NS, o qual é composto por memória e microprocessador (DARGIE; POELLABAUER, 2010). A memória possui a função de armazenar os dados coletados pelos sensores e suas aplicações, além dos pacotes a serem transmitidos. Os tipos de memória mais comuns em NS são a *Flash* para armazenamento não volátil dos dados e a EEPRON (Memória Somente Leitura Programável Apagável Eletricamente. O microprocessador é o componente que processa os dados. Este componente atua sobre os sensores, atuadores e o dispositivo de comunicação, além de ser responsável por escrever e recuperar dados na memória do NS. Normalmente o poder

computacional do microprocessador é pequeno a fim de colaborar com o consumo energético (KARL; WILLIG, 2005).

- **Unidade de comunicação:** responsável pela comunicação entre os NS e/ou a ERB. A comunicação sem fio se dá por Rádio Frequência, pois a mesma é que mais se adapta aos requisitos de aplicações para RSSF, devido a acessibilidade em locais no qual seria difícil a utilização de cabos para realizar a comunicação (RUSSELL; DUREN, 2016);

2.1.2. Estação Rádio Base (ERB)

A ERB consiste em um componente responsável por receber, armazenar e processar dados dos NS comuns (MOHAMMAD ILYAS; IMAD MAHGOUB, 2005). A ERB é ligada à um computador e se comunica com os NS sem a utilização de cabos. Portanto, a função da ERB é fazer a conexão entre a RSSF com uma rede cabeada.

2.1.3. Gateway

Um gateway é o elemento responsável por conectar as RSSF com uma rede externa, geralmente uma rede de computadores (OLIVEIRA, 2016b). O termo gateway geralmente é utilizado em grande parte dos trabalhos relacionados à RSSF, no entanto, é um termo pobre, pois seu único objetivo neste tipo de rede, é realizar uma passagem do meio de comunicação sem fio para um meio cabeado.

2.1.4. IoT Proxy Manager

Como dito no tópico anterior, tradicionalmente, as RSSF utilizam um *gateway* para trocar informações entre a RSSF e outras redes (SANTOS et al., 2016). No entanto, no caso deste trabalho, este elemento realiza funções adicionais de gerência na rede, como o tratamento de dados e participação no processo de cifragem. Devido a estas funções, este elemento será denominado como IoT Proxy Manager (IoT-PM). Um Proxy Manager (STALLINGS, 1998) é

um elemento capaz de receber parâmetros relacionados a gerência, solicitados por um Gerente de Rede e traduzi-los para um padrão específico.

Pelo fato do trabalho ser direcionado para a Internet das Coisas, este elemento recebe o nome IoT-PM. Este, atua tanto na pilha de protocolos do Gerente quanto na pilha de protocolos da rede, no qual os NS gerenciados estão localizados (OLIVEIRA, 2016). Sendo assim, estes elementos possuem funções de gerência da RSSF e dos dados coletados pelos NS, com funções específicas para atender a Internet das Coisas. Por fim, neste trabalho, o IoT-PM também possui participação importante nos métodos de cifragem dos dados.

2.2. Limitações das Redes de Sensores Sem Fio

As RSSF possuem algumas limitações importantes que devem ser levadas em consideração no momento de sua implementação, configuração e gerenciamento.

Os NS utilizados nas RSSF são geralmente, limitados em termos de capacidade de processamento, assim como também de armazenamento. Isto dificulta o desenvolvimento de algumas técnicas, como por exemplo, de segurança (BENTO, 2009).

Outro problema crítico é relacionado a questão de capacidade energética. É muito comum a implantação de RSSF em locais de difícil acesso, onde a substituição das baterias dos sensores é uma tarefa que será realizada o mínimo de vezes possível (PANDA, 2014).

2.2.1. Limitações das RSSF

As RSSF possuem problemas de quaisquer redes sem fio, onde os NS encaminham informações uns aos outros por meio da rede (BENTO, 2009). A falta de confiabilidade do canal de comunicação, a falta de estrutura física e problemas de colisão de pacotes ou danificação de pacotes são problemas comuns.

Com relação às interferências na transmissão de dados em uma RSSF, a ocorrência é muito maior em redes de dimensões elevadas, com milhares de NS

por exemplo, onde existe uma probabilidade elevada de ocorrer colisões de pacotes e latência (ZIA; ZOMAYA, 2006).

As interferências em redes sem fio são comuns, principalmente quando utiliza-se faixas de frequência não licenciadas em sua implantação, como a de 2,4 Ghz por exemplo. Alguns equipamentos que também utilizam estas faixas, pelo fato destas serem gratuitas, podem interferir no sinal de um equipamento em uma rede sem fio (RUFINO, 2014).

3. SEGURANÇA DA INFORMAÇÃO EM REDES DE SENSORES SEM FIO

Este capítulo aborda os principais conceitos de Segurança da Informação. Além disto, o capítulo tratará também da segurança nas RSSF, assim como dos tipos possíveis de ataques neste tipo de rede.

3.1. Segurança da Informação

A Segurança da Informação está diretamente relacionada com a proteção de um conjunto de informações com o objetivo de preservar o valor que possuem para um indivíduo ou uma organização (MACHADO, 2014). Este termo, pode ser definido como um processo para proteger a informação do mau uso tanto acidental como intencional, por pessoas internas ou externas à um ambiente (MACHADO, 2014).

Uma vulnerabilidade ou falha em uma rede pode ocorrer por vários fatores, como por exemplo, um problema em uma aplicação, um serviço, ocorrência de um vírus, ou ainda, um usuário que divulga a senha indiscriminadamente (CERT.BR, 2012). Existem também, ameaças mais perigosas, como alguém que sabota a base de dados de uma empresa por vingança, ou mesmo por um espião contratado para obter informações importantes da empresa através de sua rede (MORAES, 2010).

A grande maioria dos problemas relacionados a segurança em uma rede de computadores também existe nas RSSF (CHELLI, 2015). Alguns destes problemas, inclusive, são bem mais críticos neste tipo de ambiente, devido a fatores já citados em capítulos anteriores, como limitações de *hardware* e meio de comunicação utilizado.

Tendo em vista as áreas de gerência de redes definidas pela *International Standard Organization* (ISO) a ITU, por meio da Recomendação M.3400 (ITU-T, 2000), definiu 5 áreas funcionais de gerenciamento para redes, apresentadas na Tabela 1.

Tabela 1 - Áreas Funcionais de Gerenciamento de Redes (ISO)

Área	Definição
Desempenho	Monitoração do desempenho dos elementos da rede e da qualidade das atividades de comunicação.
Falha	Detecção e correção de operações anormais na rede.
Configuração	Gerenciamento dos dados de inicialização e desligamento da rede, bem como a configurações pertinentes a operação da mesma.
Contagem	Bilhetagem e controle de custos a utilização de elementos da rede.
Segurança	Controle e proteção dos recursos de rede.

Fonte: elaboração própria

Com base nas áreas de gerência mostradas na Tabela 1, o trabalho irá explorar a área de gerência de segurança, pois irá abordar mecanismos de proteção e segurança em RSSF.

O padrão internacional *Open System Interconnection* (OSI) é utilizado como referência para profissionais organizarem uma tarefa e prover segurança das informações (STALLINGS, 2008). A arquitetura de segurança OSI enfoca ataques, mecanismos e serviços de segurança. Estes podem ser definidos da seguinte forma (STALLINGS, 2008).

- **Ataque à segurança:** Qualquer ação que comprometa a segurança da informação pertencente a uma organização;
- **Mecanismos de segurança:** Um processo que é projetado para detectar, impedir ou permitir a recuperação de um ataque à segurança;
- **Serviço de segurança:** Um serviço de processamento ou comunicação que aumenta a segurança dos sistemas de transmissão.

A Segurança da Informação é um objetivo que deve ser buscado de forma constante. O autor (MORAES, 2010) afirma que não existe rede ou mesmo informação 100% segura. Todas as vezes que informações são disponibilizadas em uma rede e estas são transmitidas, existem riscos.

3.2. Segurança da Informação nas RSSF

Assim como ocorre na maior parte das redes, as informações transmitidas em uma RSSF necessitam de segurança (DENER, 2014). Como já citado, geralmente as RSSF não possuem mecanismos de segurança necessários para que as informações possam ser transmitidas de forma segura. Os mais importantes requerimentos de segurança em uma RSSF são listados a seguir (SEN, 2009):

- **Integridade dos dados**: visa garantir que todas as características originais dos dados geradas no sensoriamento do NS sejam mantidas durante todo o roteamento até a ERB em todo o ciclo de vida do dado; diferentes aplicações de RSSF correspondem a diferentes requisitos de segurança. Por exemplo, no monitoramento da umidade do solo de uma plantação, espera-se que os dados coletados reflitam a realidade do que foi medido nos pontos da plantação. Neste caso a **integridade** dos dados transmitidos é essencial.

- **Confidencialidade dos dados:** o direito de acessar a informação deve ser dado apenas ao NS que tem autorização para acessá-lo. Os dados obtidos sobre a umidade de uma plantação não são sigilosos, porém a divulgação do estado de equipamentos de uma empresa, por exemplo, poderia levar a mesma a prejuízos ou apresentar vantagens para empresas concorrentes. Neste caso, a **confidencialidade** dos dados também é essencial.

- **Autenticidade dos dados**: visa garantir que o dado vem realmente do NS no qual foi produzido, não sendo alvo de nenhum tipo de modificação no caminho. Se as RSSF forem utilizadas para a área da saúde por exemplo, na qual impacta diretamente na vida dos seres humanos, além dos dois itens anteriores, é essencial também garantir a **autenticidade** dos

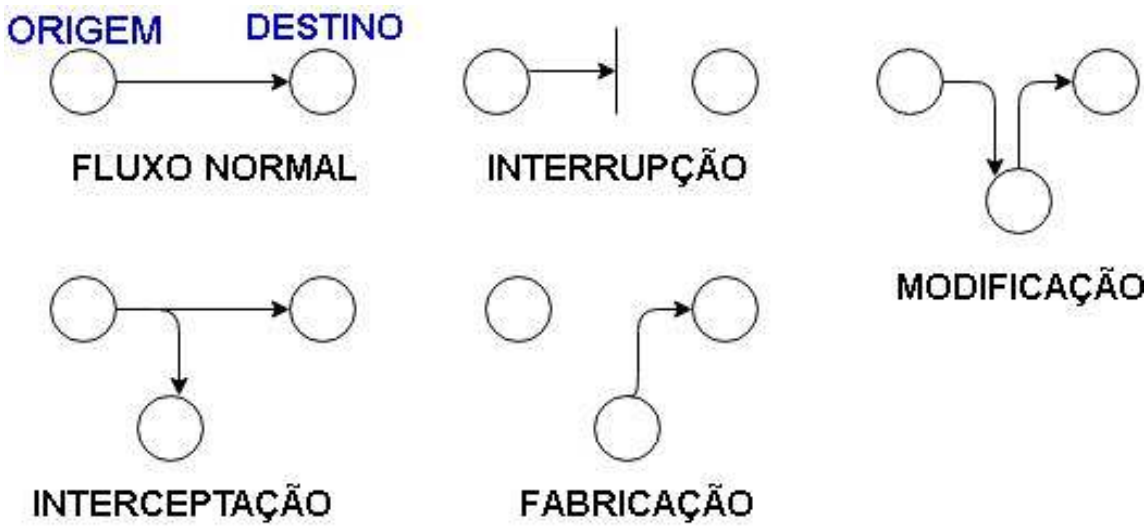
dados que são transmitidos e até mesmo a autenticidade dos NS. Os dados sempre são importantes e devem ser autênticos.

- **Disponibilidade dos dados:** permite que o dado esteja sempre disponível para os NS autorizados a acessá-lo. É importante garantir a **disponibilidade** da estrutura da RSSF. São necessários para esse fim, mecanismos para evitar ataques que porventura impeçam o funcionamento de um ou vários NS da rede.

3.3. Tipos de ameaças em uma RSSF

Existem várias formas de ameaças para a informações transmitidas em uma RSSF, que podem afetar a integridade parcialmente ou até mesmo totalmente dos dados, além de em alguns casos, também a disponibilidade, autenticidade e confidencialidade. Na Figura 3, são mostrados os tipos de ameaças em uma rede e que se adequam também as RSSF.

Figura 3 – Tipos de ameaças em uma RSSF



Fonte: Adaptado de (STALLINGS, 1998)

Os tipos de ameaças existentes em redes exibidas na Figura 3 são descritas a seguir (STALLINGS, 1998):

- **Fluxo normal:** é a comunicação sendo realizada normalmente entre o emissor e o receptor da mensagem.
- **Interrupção:** um recurso da RSSF é destruído e se torna indisponível. Isto ameaça a disponibilidade do ambiente. Um exemplo seria a ERB ou até mesmo um NS destruído fisicamente, impossibilitando sua utilização na rede.
- **Interceptação:** um indivíduo não autorizado de alguma forma acessa os dados transmitidos pela RSSF ou armazenados em algum de seus componentes. Este é um caso de ameaça à confidencialidade dos dados.

- **Modificação:** um destinatário não autorizado consegue acessar os dados transmitidos na RSSF para fins de efetuar alguma ação maliciosa. Com isto o atacante pode modificar os valores dos dados, alterando consequentemente o resultado correto.

- **Fabricação:** o invasor tem acesso de alguma forma à RSSF e insere dados falsos na mesma. Nesse tipo de ameaça, é possível inserir mensagens falsas para os NS para que os mesmos retornem alguma informação útil para o atacante.

3.4. Problemas nas RSSF

Uma RSSF, assim como outras redes, precisa ficar o maior tempo possível em funcionamento (SANTOS et al., 2016). Entretanto, alguns fatores podem fazer com que uma RSSF sofra algum dano, seja na rede em si ou na informação que a mesma transmite entre os NS.

Um mau funcionamento de uma RSSF pode ocorrer por alguma atividade feita por um atacante, de forma maliciosa, ou ainda, por algum outro fenômeno não necessariamente de origem maliciosa. É possível por exemplo, que algum equipamento próximo e que esteja utilizando o mesmo canal de frequência de um NS, interfira em seu sinal, o deixando assim inoperante (RUSSELL; DUREN, 2016).

3.4.1. Problemas nas RSSF sem origem maliciosa

Nem sempre um mau funcionalmente de uma RSSF está relacionado a uma ação maliciosa feita por um atacante (RUSSELL; DUREN, 2016). Em muitos casos, problemas físicos em algum NS ou entre eles também podem prejudicar o desempenho da RSSF ou até mesmo interromper seu funcionamento por completo. Alguns exemplos são citados a seguir (BENTO, 2009):

- **Nós parados:** quando um NS para de funcionar, poderá haver perda das informações enviadas para os nós agregados a ele em uma rede hierárquica. Os protocolos da RSSF devem ser robustos e elaborar uma rota alternativa de forma a reduzir o prejuízo na rede.

- **Nós avariados:** muitas vezes o NS pode não parar completamente, no entanto ele fica oscilando e instável em seu funcionamento, o que pode afetar a integridade dos dados. Em redes hierárquicas, em que os NS estejam em cluster, esta oscilação pode ter uma elevada relevância, pois consequentemente as informações provenientes dos NS inferiores poderão se danificadas ou perdidas;

Além disso, as RSSF apresentam alguns problemas característicos de quaisquer outras redes onde o método de transmissão é realizado sem a utilização de cabos, que são as possíveis interferência com outros equipamentos próximos que eventualmente possam utilizar frequência semelhante (LOUREIRO et al., 2003).

3.4.2. Problemas nas RSSF de origem maliciosa (ataques)

Como qualquer outro tipo de rede, uma RSSF pode sofrer ataques. Os atacantes em redes podem ser classificados como internos e externos. Os atacantes internos conseguem se passar como membros da RSSF, enquanto os externos podem influenciar, mas não participar da rede. A eficiência e as possibilidades de ataques variam de acordo com o acesso do intruso à rede. Os

ataques a RSSF poder ser divididos em passivos ou ativos (FRANCO; CARDOSO, 2014).

3.4.2.1. Ataques ativos

Os ataques ativos são aqueles em que o atacante cria, altera, descarta ou inviabiliza o uso de pacotes em trânsito (FRANCO; CARDOSO, 2014). Estes tipos de ataque envolvem alguma modificação do fluxo de dados ou a criação de um fluxo falso e apresentam características opostas aos ataques passivos. É muito difícil impedir ataques ativos em sua totalidade, devido à grande variedade de vulnerabilidades físicas, de *software* e de rede em potencial (STALLINGS, 2008).

Os ataques ativos são mais numerosos, podendo atuar em diferentes camadas. A seguir alguns exemplos:

- **Captura de Nós:** geralmente as RSSF são implantadas em locais onde as dificuldades de acesso impedem o contínuo monitoramento dos elementos físicos da rede. Nestes casos, é possível a remoção física de um nó da rede. Este nó pode revelar informações importantes da rede, como informações de criptografia ou outras informações de segurança, o que pode prejudicar o restante da RSSF (BENTO, 2009).

- **Nós falsos:** esta técnica consiste na introdução de um nó na rede com o objetivo de propagar informações maliciosas. Este nó irá receber informações dos outros nós. Geralmente este nó possui boa capacidade de processamento para comportar o volume de dados que recebe(CHELLI, 2015) (ADAM; SHEHU, 2017) .

- **Mensagens corrompidas:** Alterações de conteúdo de uma mensagem podem comprometer sua integridade. Caso um atacante consiga informações sobre a rede, o mesmo pode

alterar informações em mensagens ao redor da rede, podendo comprometê-la em sua totalidade (DENER, 2014).

- **Ataques de roteamento:** consistem em uma alteração, limitação ou até mesmo o impedimento do roteamento na RSSF. Estes ataques consistem em nós comprometidos pelos atacantes. Isto se deve muito ao fato dos protocolos utilizados para roteamento nas RSSF serem simples devido as limitações do *hardware* utilizado. Um ou vários sensores podem deixar de encaminhar o tráfego para o restante da rede. Um nó malicioso pode imitar um outro nó na rede apresentando várias identidades e o ataque de “Hello”, que consiste em um nó com elevada capacidade de transmissão de se fazer passar por vizinho de todos os nós da rede e assim, fazer com que o tráfego da rede passe por ele (SEN, 2009) (LATA; GUPTA, 2014).

- **Negação de Serviço:** é um tipo de ataque que ocorre na camada física. Este tipo de ataque visa interferência de sinal, levando a exaustão da bateria por exemplo. Um primeiro tipo é por **adulteração** para a captura física do NS. Os NS devem ser capazes de apagar qualquer dado criptográfico que existam na memória, que coloque em perigo a confidencialidade da rede. Para prevenir este tipo de ataque, uma opção seria camuflar ou ocultar os sensores. Um segundo tipo por **interferência** para impedir a comunicação dos NS. Para impedir este tipo de problema, uma alternativa seria uma porção de frequências distintas e não apenas um canal. Entretanto, um consumo energético e uma estrutura mais elaborada seriam necessários(ARAUJO et al., 2012) (ASSIS et al., 2015).

3.4.2.2. Ataques passivos

Os ataques passivos não afetam a operação da rede, sendo caracterizados pela espionagem dos dados sem alteração dos mesmos (MACHADO, 2014). Estes ataques possuem a natureza de bisbilhotar ou monitorar transmissões. O objetivo é obter informações que estão sendo transmitidas.

Os ataques passivos são muito difíceis de detectar, pois não envolvem alteração dos dados (STALLINGS, 2005). Normalmente, o tráfego de mensagens ocorre em um padrão aparentemente normal, e nem o emissor nem o receptor estão cientes de que um terceiro NS leu as mensagens ou observou o padrão de tráfego. Porém é viável impedir o sucesso desses ataques por meio de criptografia (STALLINGS, 2008). A seguir alguns exemplos de ataques passivos:

- **Captura passiva de informação:** um atacante pode capturar informações que trafegam por uma rede não protegida. Se a comunicação entre NS for realizada sem codificação, um atacante com um receptor pode capturar os dados. Nestas informações capturadas, podem estar a localização física dos NS, o que pode facilitar a destruição dos mesmos pelo atacante. São necessárias técnicas de criptografia para tornar estas informações inelegíveis aos atacantes (FERNANDES et al., 2006).

- **Análise de Tráfego:** Um atacante que pretende atacar a estação base, pode obter a localização da mesma através do fluxo de dados na rede, se aproveitando do fato das RSSF transmitirem dados de forma assimétrica, pois o propósito dos NS é a coleta dos dados. Estas características são uteis para a localização da estação base, mesmo que as mensagens sejam transmitidas de maneira criptografada (MARGI; JR; BARRETO, 2009)

3.5. Desafios para implementação de segurança em RSSF

Uma vez que as RSSF são compostas de NS com características como processamento, armazenamento, comunicação e energia limitados, os mecanismos de segurança utilizados devem ser ponderados no sentido do consumo destes recursos, para que a estrutura da rede continue funcionando com desempenho satisfatório. Segundo (HU; SHARMA, 2005), os principais problemas que tornam a implementação de soluções de segurança em RSSF, uma questão desafiadora:

- **Conflito entre minimizar consumo de recursos e maximizar a segurança:** neste quesito, os recursos em questão são memória, processador e energia. Para garantir o sigilo das mensagens, existe a necessidade de criptografar os dados, o que pode aumentar o tamanho da mensagem transmitida, o que poderia ocasionar um aumento na utilização destes recursos. A criptografia é uma operação que é executada no NS de origem e em cada NS que precise ler o conteúdo da mensagem cifrada.
- **Topologia de RSSF suscetível a ataques ao enlace:** ao contrário das redes via cabo, as redes *wireless* não possuem proteção física ao meio de transmissão. Com isto, diversos ataques ao meio são possíveis. Existe também um grande número de NS com possibilidade de mobilidade em alguns casos, que normalmente não são monitorados. É possível assim, que NS possam ser substituídos por NS maliciosos, comprometendo a segurança da RSSF.
- **Características da comunicação sem fio:** a comunicação sem fio traz grandes desafios às RSSF. Alguns deles são: a distância limitada de transmissão, baixa largura de banda, além de possíveis problemas na entrega dos pacotes devido a interferência ou atenuação. Tais fatores tornam os enlaces muitas vezes não confiáveis, o que requer métodos de gerência e segurança distintos dos utilizados em redes cabeadas.

3.6. Cifragem e Criptografia

Existe uma diferença importante entre os termos cifragem e criptografia. Essas distinções precisam ser entendidas para uma melhor compreensão do trabalho.

3.6.1. Cifragem

No âmbito da Segurança da Informação, uma cifragem é o ato de alterar a mensagem original, por meio de ordem, aparência, tipos de letras ou caracteres, ou ainda através de algum algoritmo de criptografia, de modo a torna-la ininteligível para quem a interceptar e não souber como reproduzir a mensagem original (WHITE, 2012). Uma cifragem pode simplesmente substituir um caractere ou grupo de caracteres por um caractere ou grupos de caracteres diferentes, sem utilizar para isto, nenhuma chave de criptografia (WHITE, 2012). Na Tabela 2, é mostrado um exemplo simples de cifragem, no qual os caracteres originais são substituídos por caracteres diferentes.

Tabela 2 - Exemplo simples de cifragem com alteração dos caracteres

Texto simples	a	b	c	d	e	f	g
Texto cifrado	P	O	I	U	Y	T	R

Fonte: Adaptado de (WHITE, 2012)

3.6.2. Criptografia

A criptografia por sua vez, é considerada uma ciência e a arte de escrever mensagens em forma de códigos. É um dos principais mecanismos de segurança para proteger os dados em uma rede (CERT.BR, 2012).

O termo “criptografia” (em grego: kryptós, "escondido", e gráphein, "escrita"), é a ciência que utiliza algoritmos matemáticos para criptografar/encriptar dados (texto claro), de forma não legível (texto cifrado) e recuperá-los (descriptografá-los) (MORAES, 2010). Com esta ciência, o custo de

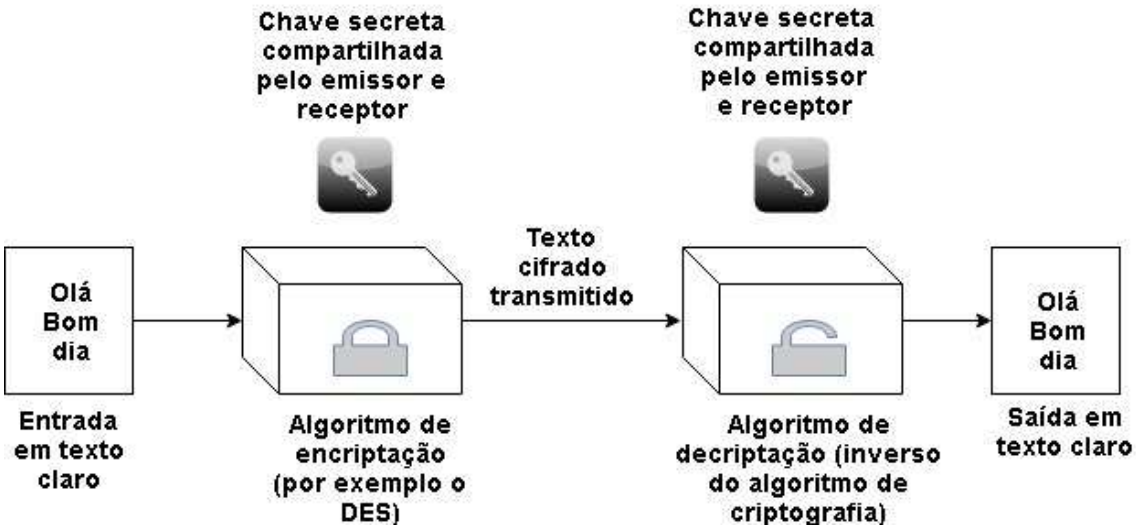
tentar descobrir o conteúdo das mensagens cifradas torna-se maior do que o potencial ganho com os dados. Alguns termos são importantes quando se trata de criptografia (JAMES F. KUROSE; KEITH W. ROSS, 2010):

- **Encriptação:** é o processo de transformação de dados claros em uma forma ilegível, ou seja, encriptados. O propósito é garantir privacidade, mantendo a informação escondida para qualquer um que não seja o destinatário da mensagem, mesmo que ele possa ter acesso às informações criptografadas.
- **Decriptação:** é o processo reverso da criptografia; é a transformação dos dados encriptados de volta à forma de texto claro.
- **Texto claro:** é a mensagem a ser enviada. Se for interceptada em uma comunicação, pode ser compreendida pelo interceptador porque não se encontra criptografada.
- **Texto cifrado:** é uma mensagem que passou por um processo de encriptação e, se for interceptada em uma comunicação, não pode ser compreendida pelo interceptador, desde que ele não conheça a chave e o algoritmo criptográfico utilizados

3.6.2.1. Criptografia simétrica

A criptografia simétrica utiliza uma única chave para codificar, como para decodificar as informações (CERT.BR, 2012), sendo usada principalmente para garantir a confidencialidade dos dados. Exemplos de métodos criptográficos simétricos são: AES, RC4 e o DES (CERT.BR, 2012), este último foi escolhido para este trabalho e o motivo será detalhado no Capítulo 5. A criptografia simétrica é mais utilizada do que a assimétrica (STALLINGS, 2005). Na Figura 4, é mostrado um modelo simplificado de criptografia simétrica.

Figura 4 - Modelo simplificado de criptografia simétrica



Fonte: Adaptado de (STALLINGS, 2005)

Em janeiro de 1977, o governo dos Estados Unidos adotou uma cifra de produto desenvolvida pela IBM como seu padrão oficial para informações não confidenciais (TANENBAUM, 2011). A cifra, DES (*Data Encryption Standard* — padrão de criptografia de dados), foi amplamente adotada pelo setor de informática para uso em produtos de segurança. O texto simples é criptografado em blocos de 64 bits, produzindo 64 bits de texto cifrado (SCHNEIER, 1996a). O algoritmo foi projetado para permitir que a decodificação fosse feita com a mesma chave da codificação, uma propriedade necessária em qualquer algoritmo de chave simétrica. As etapas são simplesmente executadas na ordem inversa (STALLINGS, 2005).

O algoritmo DES foi o escolhido para compor um dos métodos de cifragem deste trabalho pelo fato de ser a opção de criptografia simétrica que mais se ajustou com dois fatores importantes para que o método funcionasse. O primeiro é a estrutura do pacote da plataforma Radiuino (“Radiuino”, 2018), que foi a plataforma utilizada para a realização dos experimentos. Já o segundo motivo é a simplicidade do algoritmo comparado com outras opções disponíveis, o que torna seu processamento mais leve e mais viável para o âmbito das RSSF, que conforme já citado, possuem muitas limitações neste sentido.

3.6.2.2. Criptografia assimétrica

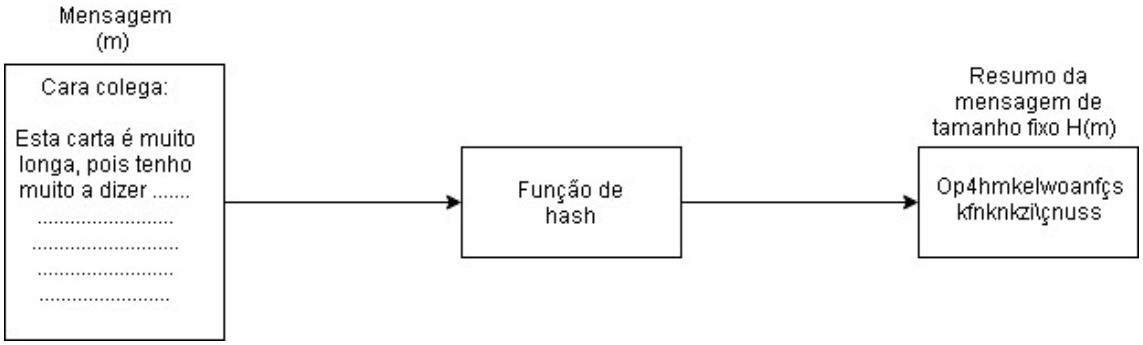
A criptografia assimétrica envolve o uso de duas chaves separadas, uma chave pública e outra privada, ao contrário da criptografia simétrica (MORAES, 2010). A chave privada pode ser armazenada de diferentes maneiras, como um arquivo no computador, um *smart card* ou um *token* (CERT.BR, 2012).

Existe um grande processamento computacional dos atuais esquemas de criptografia assimétrica (CERT.BR, 2012), o que torna esse tipo de solução inviável em alguns sistemas e em algumas redes, como no caso das RSSF.

3.6.2.3. Função *Hash*

Uma função *hash* é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. Uma função *hash* recebe uma entrada m, computa um resumo de tamanho fixo H(m), conhecida como *hash* (JAMES F. KUROSE; KEITH W. ROSS, 2010). A função *hash* não utiliza chave secreta como entrada. Para autenticar a mensagem, o resumo da mensagem é enviado junto com a mensagem (STALLINGS, 2005). Na Figura 5, é mostrado um exemplo simples de funcionamento de uma função *hash*.

Figura 5 - Exemplo simples de funcionamento de uma função *hash*



Fonte: Adaptado de (JAMES F. KUROSE; KEITH W. ROSS, 2010)

Existem algumas opções de função *hash* disponíveis para utilização, sendo as mais utilizadas o MD5 e o SHA-1 (TANENBAUM, 2011). O SHA-1 é uma função *hash* que produz chaves de 160 bits de tamanho e é normalmente tratado como um número hexadecimal de 40 dígitos. O SHA-1 foi projetado pela Agência de Segurança Nacional dos Estados Unidos (SCHNEIER, 1996b).

O MD5 opera desfigurando os bits de uma forma tão complicada que todos os bits de saída são afetados por todos os bits de entrada (TANENBAUM, 2011). O MD5 gera um valor fixo de 128 bits (JAMES F. KUROSE; KEITH W. ROSS, 2010). Por questões de adequação à estrutura do pacote da plataforma Radiuino, a função *hash* MD5 foi a escolhida para o desenvolvimento do segundo método de cifragem, em detrimento ao SHA-1, que gera um valor fixo de 160 bits como resultado.

3.7. Interceptação de tráfego

Interceptação de tráfego, ou *sniffing*, é uma técnica que consiste em inspecionar os dados trafegados em redes, por meio do uso de programas específicos chamados de *sniffers*. Esta técnica pode ser utilizada de duas formas (CERT.BR, 2012):

- Legítima: por administradores de redes, para detectar problemas, analisar desempenho e monitorar atividades maliciosas relativas aos dispositivos ou redes por eles administrados.
- Maliciosa: por atacantes, para capturar informações sensíveis, como senhas, conteúdo de arquivos confidenciais que estejam trafegando por meio de conexões inseguras, ou seja, sem criptografia.

As informações capturadas por esta técnica são armazenadas na forma como trafegam, ou seja, informações que trafegam criptografadas apenas serão úteis ao atacante se ele conseguir decodifica-las. Para comprovar a eficácia dos métodos de cifragem deste trabalho, foi desenvolvido um *sniffer* para cada solução. O funcionamento de cada *sniffer* será detalhado no Capítulo 5.

4. TRABALHOS RELACIONADOS

Uma vez que este trabalho apresenta soluções para a cifragem dos dados em RSSF, é necessário por pesquisar na literatura, trabalhos com tais características.

Em (ASSIS et al., 2015), é apresentada uma abordagem a respeito dos aspectos de segurança em RSSF. O artigo mostra componentes, conceitos e aspectos operacionais para implementação de criptografia e protocolos seguros para RSSF. O trabalho exibe detalhes de como os sensores operam, processam e transmitem informações baseadas no processo de tomada de decisão, de acordo com regiões de processamento, e como esta comunicação pode e deve ocorrer com segurança. A criptografia nas RSSF pode anular ou diminuir a gravidade da grande maioria dos tipos de ataque apresentados no trabalho. Entretanto, devido às limitações de recursos de *hardware*

Em (THIRUPPATHY KESAVAN; RADHAKRISHNAN, 2012), é realizada uma abordagem de segurança baseada em criptografia de chave secreta. Os NS são assumidos como tendo a mesma capacidade de processamento e comunicação. O estudo visa detectar ataques Replay, Sybil e DoS de forma efetiva. Cada NS é distribuído com dois pares de parâmetros que são então utilizados para o cálculo de nove diferentes chaves de criptografia. Um valor único é definido em cada NS no momento da implementação, que é usado para inicializar o contador. A abordagem proposta tem alta exigência de memória dos elementos da RSSF devido aos parâmetros armazenados, como contador e grande número de chaves. Além disto, o método também calcula nove chaves de criptografia, o que aumenta a sobrecarga computacional nos NS e, portanto, reduz a vida útil da bateria também.

Em (KHAN et al., 2017), foi desenvolvido um método de criptografia em um modelo de comunicação onde NS do remetente coleta dados, passa para o módulo de criptografia. Depois que os dados são criptografados, eles são codificados, em seguida, o sinal é modulado. O sinal modulado passa pelo meio de transmissão onde fica distorcido devido à adição de ruído gaussiano branco aditivo ao fenômeno de desvanecimento de sinal. O pacote é recebido no NS

receptor e é desmodulado. O sinal é decodificado e descriptografado. Neste trabalho, foi realizado um estudo sobre várias chaves simétricas de criptografia, no qual foi escolhido o algoritmo de criptografia AES que trabalha com 256 bits. No entanto o tamanho excessivo da chave gerou um aumento de procesamento dos elementos da RSSF e um esgotamento rápido da alimentação de energia.

Em (SAIF UR; CUI; BAO, 2014), foi desenvolvido um método de criptografia com base em um algoritmo que pode gerar um grande número de chaves aleatórias em um arquivo relativamente pequeno. O arquivo é distribupido para os NS na RSSF de forma segura antes do mesmo ser implantado. O arquivo de origem é considerado seguro a partir da captura durante a distribuição antes da implantação, semelhante a segurança de chaves secretas. A teoria por trás do algoritmo proposto é comprovada matemeticamente. Os cálculos mostram que o algoritmo pode ser usado para gerar um número muito grande de chaves de criptografia exclusivas com o conjunto correto de parâmetros. Dependendo dos requisitos de segurança necessários na RSSF, os NS podem atualizar periodicamente suas chaves. Éste processo é realizado por um nó transmitindo uma “chave de atualização”, já esta mensagem contém um vetor de inicialização aleatória.

No trabalho (ELGENAIDI et al., 2016), os autores utilizam um gerenciamento de chaves inicialmente *offline* nos elementos da RSSF. No processo, cada NS mantém sua própria chave simétrica, chamada de chave adjacente, que é utilizada para criptografar os dados que serão transmitidos. Além disto, alguns NS, chamados de NS Líderes, são carregados com uma chave mestra inicial. A operação de rechaveamento é combinada apenas na chave de um NS Líder e uma chave adjacente. Inicialmente, todo NS envia um mecanismo chave de regeneração para criar uma nova chave. Além disso, para a implementação de *hardware* fornecida pelo Libelium, um dispositivo em particular foi usado, que é uma plataforma Waspmote. Este *hardware* é integrado a diferentes radiofrequências e protocolos.

Um método de criptografia para a plataforma ZigBee foi desenvolvido (MOH’D et al., 2012) utilizando um esquema de codificação dos dados. Neste trabalho, é utilizado o algoritmo de criptografia AES para gerar o texto cifrado. É utilizada uma criptografia baseada em *hardware*, o que torna o método eficiente

quanto ao consumo de energia. O método de criptografia pode operar em dois modos: convencional e compacto: O modo convencional mantêm a segurança das informações no cabeçalho. Já no modo compacto, é utilizado se o remetente envia mais de um pacote.

Este trabalho, difere dos outros citados neste capítulo, em primeiro lugar, no sentido de proporcionar ao administrador da RSSF duas opções de cifragem para RSSF, um método mais simples, porém que consume menos recursos da RSSF e outro mais complexo, porém que consume mais recursos da RSSF, como será explicado no Capítulo 7. Este comparativo de desempenho da RSSF com a utilização dos diferentes métodos de cifragem desenvolvidos é importante, uma vez que coma utilização de quaisquer métodos de segurança, o desempenho da rede tende a diminuir. Por este motivo, esta comparação é importante para avaliar a possibilidade de utilização de cada método nos mais variados ambientes nos quais as RSSF podem sem implementadas.

5. PROPOSTAS DE CIFRAGEM PARA REDES DE SENSORES SEM FIO

A proposta deste trabalho é apresentar duas soluções de cifragem, implementadas em RSSF com a finalidade prover segurança na transmissão de dados neste tipo de rede. Em ambos os métodos, todos os componentes das RSSF participam do processo de cifragem. Tal fato, descentraliza a responsabilidade para efetuar tal tarefa entre os elementos da RSSF, o que faz com que cada elemento tenha sua importância no processo de cifragem. Além disto, como todos os elementos participarem do processo de cifragem, ocorre uma divisão na sobrecarga de consumo de *hardware* entre os elementos da RSSF.

Em ambos os métodos desenvolvidos, as informações transmitidas entre os elementos serão protegidas contra três tipos de ameaças mostradas na Figura 3: modificação, interceptação e fabricação. Para modificar algum dado capturado, o invasor precisa primeiro, ter acesso ao texto puro. Uma interceptação também é inútil ao atacante sem a possibilidade de leitura original dos dados. Já na fabricação de algum dado de origem de algum elemento externo à RSSF, será identificado pelo NS de destino, uma vez que ele estará programado para decifrar os dados recebidos, portanto estes devem ter sido previamente cifrados pelo elemento de origem do pacote.

5.1. Método XOR

O primeiro método de cifragem desenvolvido para proteção dos dados em RSSF é realizado utilizando a operação matemática *XOR*, também conhecida como *or* exclusivo (SCHNEIER, 1996a), que é realizada em todos os bits (e consequentemente nos *bytes*) da informação que foi coletada pelos NS com a senha previamente configurada. Na Tabela 3, são mostradas as combinações possíveis em uma operação *XOR*.

Tabela 3 - Combinações da operação XOR

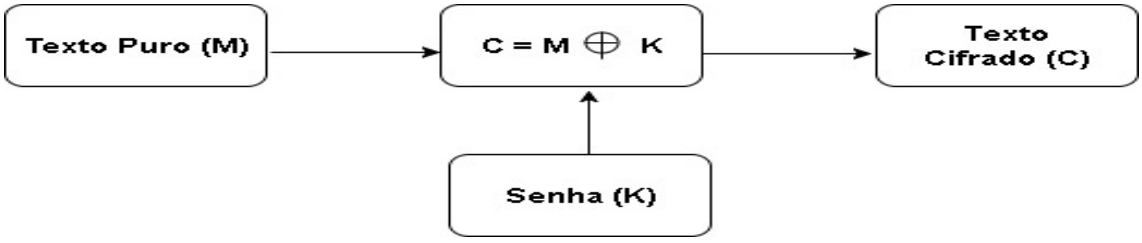
Primeiro bit	Segundo bit	Resultado da operação XOR
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: elaboração própria

5.1.1. Processo de cifragem XOR

O XOR é uma cifragem polialfabética (SCHNEIER, 1996a). Esta solução se refere à um algoritmo de cifragem simétrico, no qual o texto puro é lido e comparado com a senha. Na Figura 6, é mostrado o funcionamento da cifragem utilizando o XOR.

Figura 6 - Funcionamento da cifragem com o or exclusivo



Fonte: elaboração própria

No processo de cifragem, o texto puro (M) é gerado pelo elemento emissor. Antes do envio da mensagem, com o auxílio de uma senha (K), é realizada a operação XOR, gerando assim o texto cifrado (C). Na Tabela 4, é mostrado e emulação de uma combinação de cifragem desta operação.

Tabela 4 - Emulação cifragem com a operação XOR

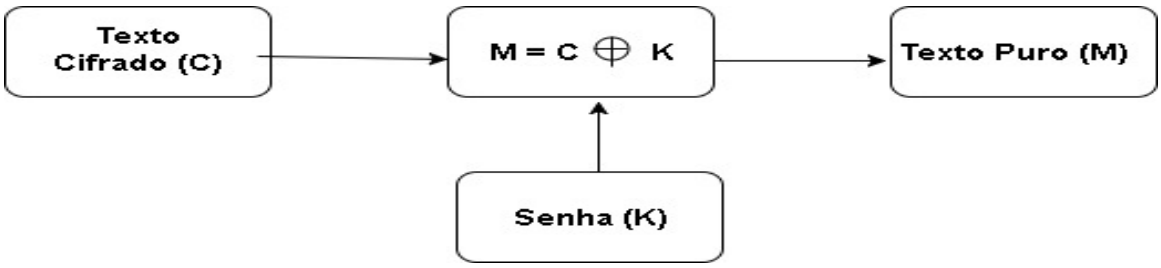
Texto puro (M)	1000110
Chave (K)	1100011
Texto cifrado (C)	0100101

Fonte: elaboração própria

5.1.2. Processo de decifragem XOR

Uma vez que o texto é cifrado pelo emissor da mensagem, o destinatário realiza o processo inverso, utilizando a mesma senha e o texto recebido de forma cifrada, para assim conseguir ler o texto puro. O conteúdo resultante do XOR realizado pelo destinatário é exatamente idêntico à mensagem original. Na Figura 7, é mostrado o processo realizado pelo destinatário para decifrar a mensagem.

Figura 7 - Funcionamento da decifragem com o XOR



Fonte: elaboração própria

No processo de decifragem, o texto cifrado (C) é recebido pelo elemento destinatário. Este, decifra a mensagem com o auxílio da mesma chave (K) utilizada na cifragem, realizada pelo emissor. Utilizando a operação or exclusivo, a combinação do texto cifrado com a chave, irá resultar no texto puro (M), que foi gerado pelo elemento emissor da mensagem, como mostra a Tabela 5.

Tabela 5 - Operação de decifragem com a operação XOR

Texto cifrado (C)	0100101
Chave (K)	1100011
Texto puro (M)	1000110

Fonte: elaboração própria

5.2. Método One Time Pad

A técnica de criptografia One Time Pad (OTP) consiste em um algoritmo que age caractere por caractere, combinando ao texto claro original a uma chave secreta aleatória que deve ter pelo menos, o mesmo número de caracteres do

seu tamanho original. Para garantir a integridade da criptografia, a chave deve ser utilizada apenas uma única vez, sendo imediatamente destruída após o uso (SCHNEIER, 1996a).

O OTP foi criado por Frank Miller em 1882 (SLATER, 1888) e é uma melhoria com relação a outra chave de uso único, a cifra de Vernam. O sistema de Vernam consistia em uma cifra que combinava uma mensagem com uma chave lida em uma fita perfurada. Em sua forma original, o sistema de Vernam era vulnerável porque a fita com as chaves era reutilizada ao chegar ao fim. O uso único veio apenas posteriormente, quando se percebeu que se a fita fosse totalmente aleatória, a decifragem da chave poderia ser impossível (KAHN, 1996).

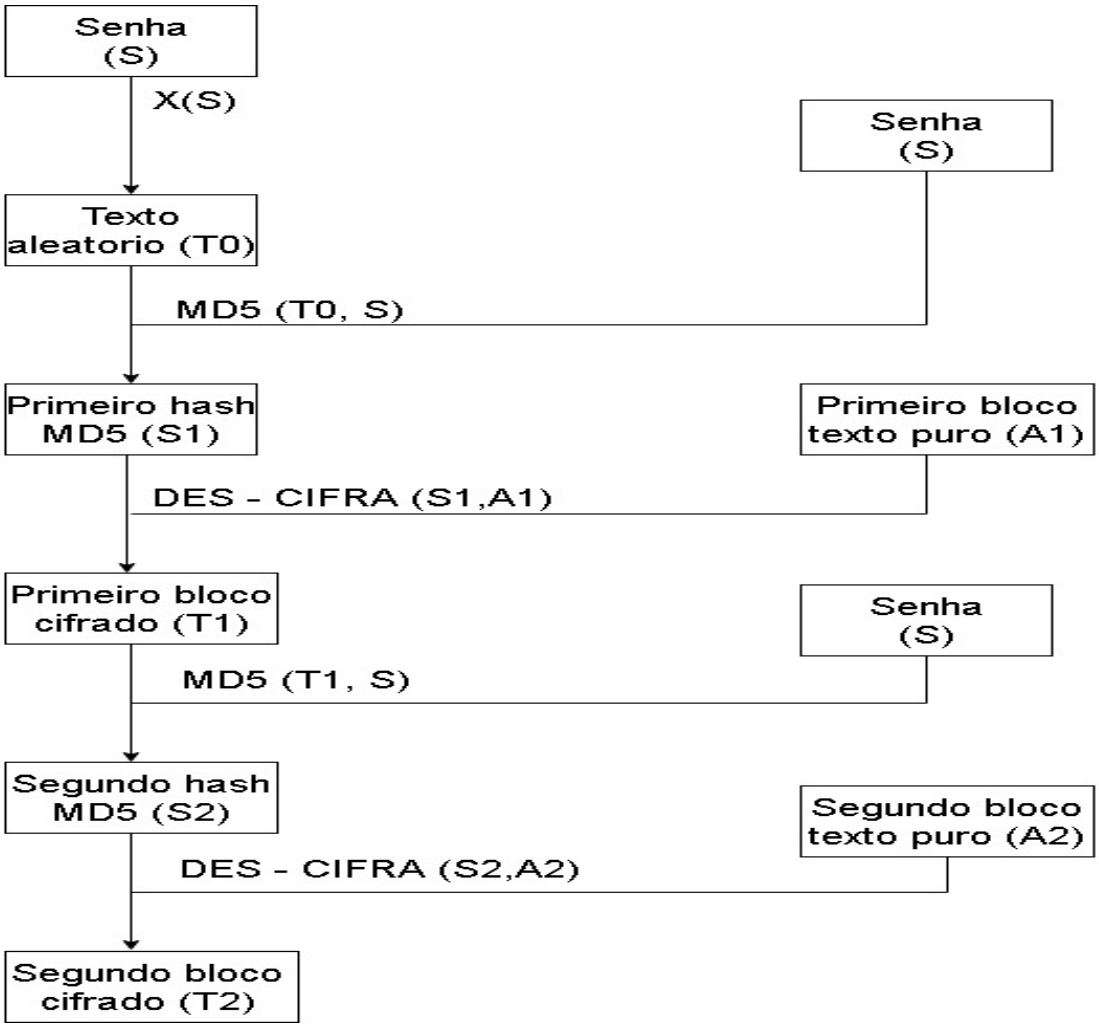
Existem algumas contradições na terminologia referente a esse tipo de criptografia, já que alguns autores usam indistintamente os termos “cifra de Vernam” e “cifra de uso único”, enquanto outros utilizam a expressão “cifra de Vernam” para qualquer cifra de fluxo (KAHN, 1996).

O método OTP, gera uma sequência aleatória de chaves, cada chave é usada apenas uma vez para cada bloco, o receptor usa a mesma seqüência de chaves para decifrar os blocos. Terceiros podem decifrar todos os blocos somente quando eles obtiverem a seqüência das chaves. Se não houver uma seqüência de chaves, é impossível decifrar o texto cifrado (TANG; LIU, 2012). Este método será explicado melhor mais adiante neste trabalho.

5.2.1. Processo de cifragem OTP

Conforme já citado anteriormente, foi proposto um procedimento de cifra de uso único. No procedimento de cifragem, será utilizada a função de *hash* MD5 para a geração das chaves, assim como o método DES para efetuar criptografia. Na Figura 8, é mostrado o processo de cifragem OTP.

Figura 8 - Processo de cifragem OTP



Fonte – Adaptado de [TANG]

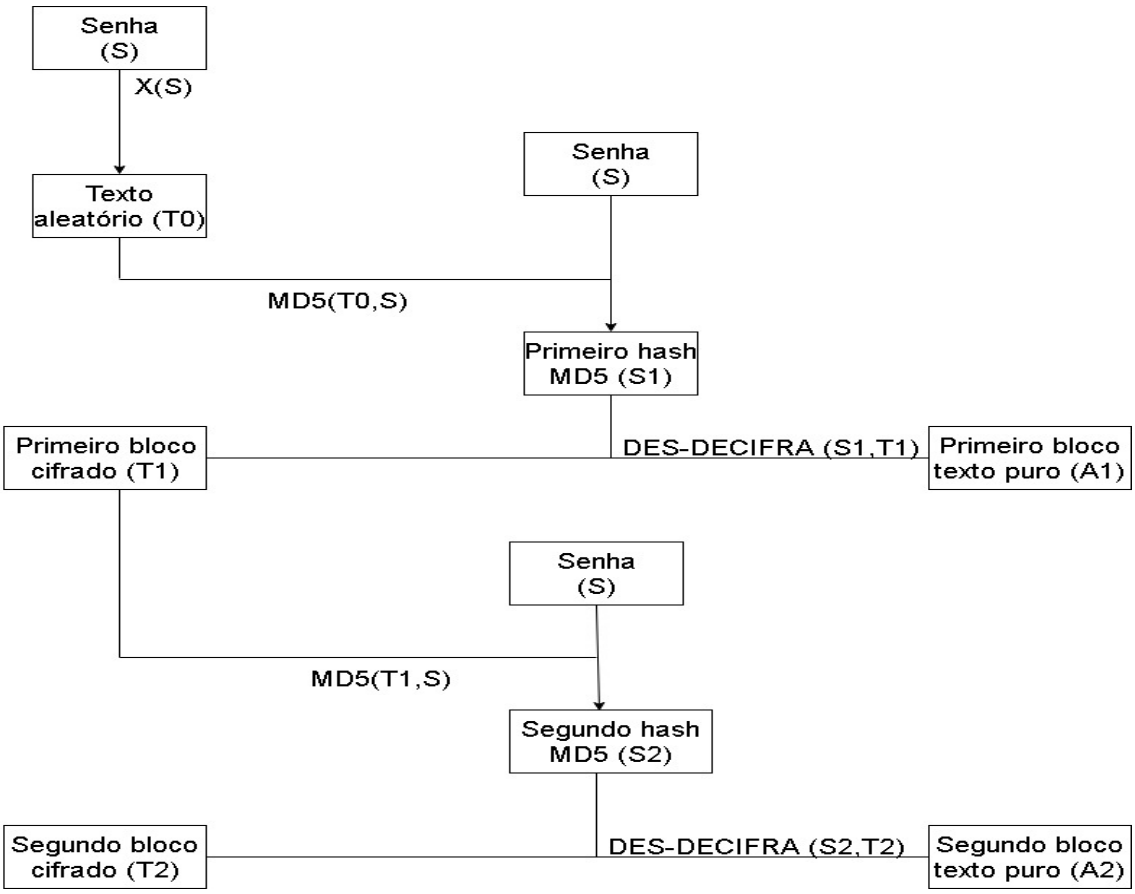
No processo de cifragem mostrado na Figura 8, o primeiro passo é transformar a entrada de dados (S) em um primeiro bloco de texto simples (T0), através da função X(S), que contém uma operação XOR simples. Em seguida, é utilizada a função *hash* MD5 (T0,S) para obter a chave criptográfica do primeiro bloco (S1). A função MD5 conecta T0 a S e faz uma operação, gerando S1. Na próxima etapa, a função DES_CIFRA (S1, A1) é acionada para criptografar o primeiro bloco. Esta função utiliza algoritmo de criptografia simétrica DES e utiliza A1, que é o primeiro bloco sem cifragem, como sua entrada, gerando o primeiro bloco de texto cifrado (T1).

O processo descrito irá se repetir com todos os blocos de dados do pacote a ser transmitido.

5.2.2. Processo de decifragem OTP

O elemento que receber um pacote cifrado com o OTP precisa ter a mesma chave inicial do emissor da mensagem, além de também utilizar o mesmo algoritmo simétrico de criptografia, no caso o DES e a mesma função *hash*, no caso o MD5. Na Figura 9, é mostrado o processo de decifragem OTP.

Figura 9 - Processo de decifragem OTP



Fonte – Adaptado de (TANG; LIU, 2012)

O primeiro passo é transformar a chave do usuário (S) em um bloco de texto simples (T0) através da função de transformação X(S). Esta função contém algumas operações XOR. Em seguida, é utilizada a função MD5 (T0,S) para obter a chave de criptografia do primeiro bloco (S1). A próxima etapa, a função DES_DECIFRA é acionada para descriptografar e utiliza a chave do primeiro bloco (S1) e conjunto com o primeiro bloco cifrado (T1) para gerar o primeiro bloco de texto simples (A1), que é o primeiro bloco sem cifragem.

Todo o processo descrito irá se repetir até todos os blocos que chegarem de forma criptografada, sejam descriptografados.

5.3. Etapas dos métodos propostos

Como citado no início deste capítulo, todos os elementos que compõem a RSSF participam das etapas de quaisquer um dos métodos propostos neste trabalho, ou seja, cada elemento da RSSF terá sua participação como é mostrado na Figura 10.

Figura 10 - Etapas do processo de cifragem na RSSF

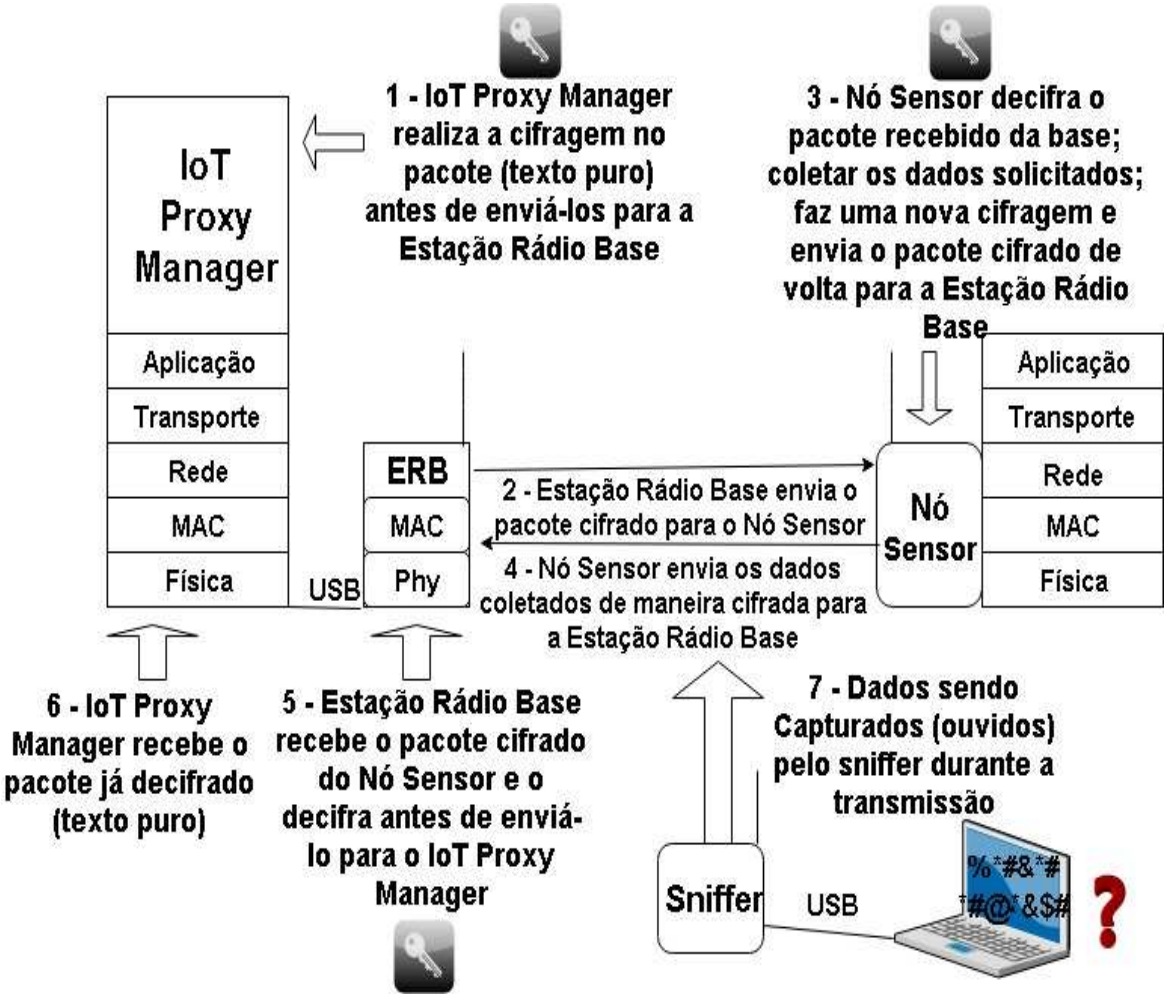
Fonte: elaboração própria

5.4. *Sniffer* para RSSF

Para a realização deste trabalho, foram desenvolvidos *sniffers* para cada solução de cifragem. Nas redes tradicionais, existem várias soluções de mercado que realizam esta função, porém, tais tecnologias não são compatíveis com as RSSF. Sendo assim, para cada método de cifragem desenvolvido, um *sniffer* foi inserido como um elemento adicional da rede. Este componente é capaz de capturar os pacotes transmitidos entre a ERB e o NS, em ambas as direções da comunicação. O objetivo da utilização do *sniffer*, é emular um

invasor que possui acesso físico à RSSF e tenta interceptar as informações transmitidas entre os dois elementos citados. Na Figura 11, é mostrado o cenário com a utilização de quaisquer um dos dois métodos de cifragem, utilizando o *sniffer.*

Figura 11 - Processos de cifragem na RSSF com o *sniffer*



Fonte: elaboração própria

6. MATERIAL E MÉTODOS

Para implementar os métodos de cifragem, foram utilizados *hardware* e *software* de código aberto (*open-hardware* e *open-software*). A princípio, serão detalhados o *hardware* e os *softwares* utilizados no IoT-PM. Em seguida, o mesmo será realizado para a ERB, o NS e o *sniffer*. E por fim, o computador no qual o *sniffer* será conectado terá seu hardware e *software* detalhado.

6.1. Plataforma Radiuino

A plataforma Radiuino é uma biblioteca desenvolvida para o Arduino (“Radiuino”, 2018). O Radiuino utiliza linguagem de programação C e *hardware* e *software* de código aberto, o que foi de fundamental importância para o desenvolvimento dos métodos de cifragem deste trabalho. O fato do Radiuino consistir em um código aberto, propicia também a criação de novos recursos nas RSSF.

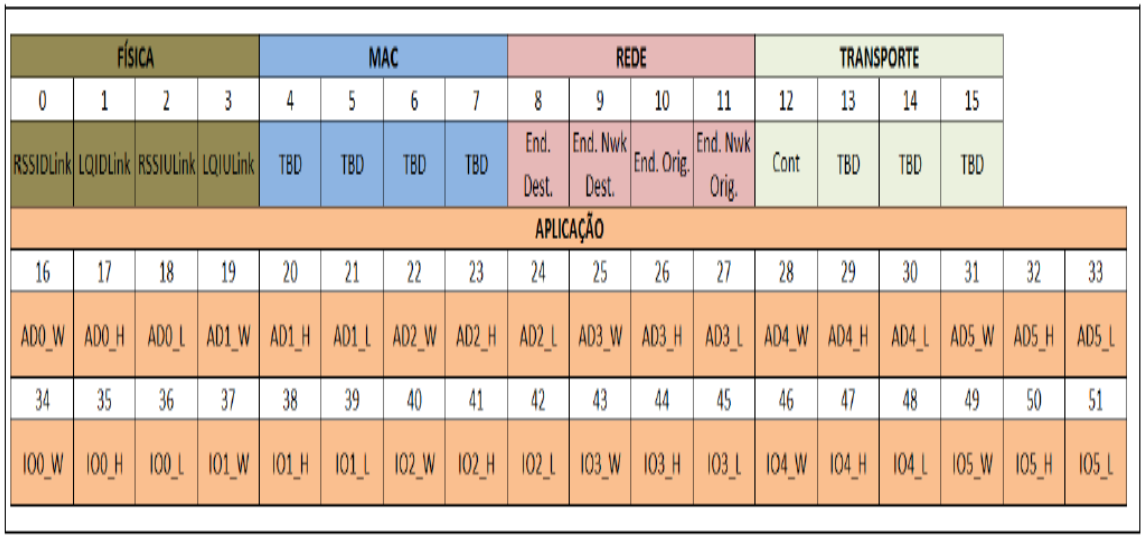
6.1.1. Composição do pacote

A plataforma Radiuino define um pacote contendo 52 *bytes*, os quais são divididos em cinco camadas (“Radiuino”, 2018), da seguinte forma:

- Física (4 *bytes*);
- MAC (*Media Access Control*) (4 *bytes*);
- Redes (4 *bytes*);
- Transporte (4 *bytes*);
- Aplicação (36 *bytes*) (carga útil, contendo 6AD (*Analogic Digital*) e 6 IO (*Input / Output*);

Na Figura 12, é mostrado o mapa do pacote original do Radiuino. Nela, é possível observar a quantidade de *bytes* disponíveis para cada camada.

Figura 12 - Mapa do pacote original do Radiuino



FÍSICA				MAC				REDE				TRANSPORTE					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
RSSIDLink	LQIDLink	RSSIULink	LQIULink	TBD	TBD	TBD	TBD	End. Dest.	End. Nwk Dest.	End. Orig.	End. Nwk Orig.	Cont	TBD	TBD	TBD		
APLICAÇÃO																	
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
AD0_W	AD0_H	AD0_L	AD1_W	AD1_H	AD1_L	AD2_W	AD2_H	AD2_L	AD3_W	AD3_H	AD3_L	AD4_W	AD4_H	AD4_L	AD5_W	AD5_H	AD5_L
34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
IO0_W	IO0_H	IO0_L	IO1_W	IO1_H	IO1_L	IO2_W	IO2_H	IO2_L	IO3_W	IO3_H	IO3_L	IO4_W	IO4_H	IO4_L	IO5_W	IO5_H	IO5_L

Fonte: Adaptado de (“Radiuino”, 2018)

6.2. IoT-Proxy Manager

Para a escolha do *hardware* do IoT-PM, foram levados em consideração alguns fatores importantes (OLIVEIRA, 2016) :

- Capacidade de processamento;
- Capacidade de Memória RAM;
- Capacidade de armazenamento;
- Conexões USB, Wireless e HDMI;
- Custo;
- Tamanho;

Utilizando como parâmetro, as características citadas acima, foi realizada uma pesquisa para verificar os principais computadores de pequeno porte disponíveis no mercado, voltados para atuar em aplicações de IoT (MAKSIMOVIĆ et al., 2014). Foi detectada através da pesquisa realizada, que os dispositivos com estas características mais utilizados atualmente são: Raspberry Pi 3 (Raspberry Pi 3, 2018), BeagleBone Black (“Beaglebone Black”, 2018) e o Arduino Uno (“Arduino Uno”, 2018). Na Tabela 6, é mostrado um comparativo entre estes computadores nos quesitos listados anteriormente.

Tabela 6 - Comparativo dos computadores de pequeno porte

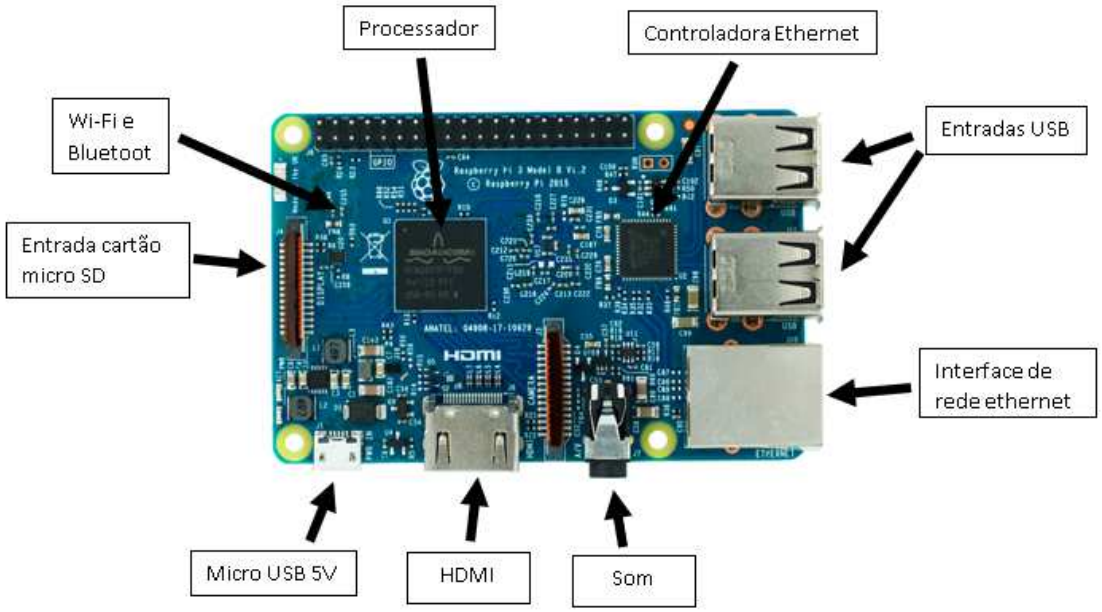
Computador	Processamento	Memória	Armazenamento	USB	Rede	HDMI	Custo	Tamanho
Arduino Uno	ATmega328P	2KB	32KB	1	Não Possui	Não Possui	R$ 84,48	68.6 x 53.4 mm
Beaglebone Black	AM335x 1Ghz ARM Cortex-A8	512MB	4GB	2	Ethernet 10/100 Mbps	Possui	R$ 399,90	86.3 x 53.3 mm
Raspberry Pi 3	Quad Core 1.2GHz BCM2837	1GB	Não possui nativamente. Entrada para cartão SD	4	Ethernet 10/100 Mbps Wireless 802.11 b/g/n Bluetooth 4.1	Possui	R$ 199,90	85.6 x 53.9 mm

Fonte – Elaboração própria

Embora sejam comuns os comparativos entre o Arduino Uno e os outros dois elementos demonstrados na Tabela 6, é importante citar que este é utilizado geralmente para outras finalidades. Isto se deve pelo fato do Arduino Uno ser uma placa com microcontrolador, que pode executar apenas um programa por vez (“Arduino Uno”, 2018). Já o Beaglebone Black e o Raspberry Pi 3 são computadores com capacidade de executar vários programas, portanto, são geralmente utilizados para aplicações mais complexas (“Diferenças entre Arduino, Raspberry Pi e Beaglebone”, 2018).

Tendo em vista o melhor poder computacional, memória RAM, entradas USB (*Universal Serial Bus)* e acesso à rede, optou-se pelo *Raspberry Pi* 3 para hospedar o IoT-PM. Embora não haja de maneira nativa um armazenamento interno, neste equipamento, devido à entrada para cartão SD (*San Disk*), foi possível facilmente inserir armazenamento para alocar o Sistema Operacional e aplicações, assim como os dados coletados. No caso, foi utilizado um cartão SD de 16GB (“Sandisk”, 2018). O preço do equipamento escolhido se justifica quando se é levado em consideração as melhores características de *hardware.* Na Figura 13, é mostrado uma foto do Raspberry Pi 3 e seus componentes.

Figura 13 - Raspberry Pi 3 e seus componentes



Fonte: adaptado de (“Raspberry Pi - Teach, Learn, and Make with Raspberry Pi”)

Como Sistema Operacional no IoT-PM optou-se por utilizar o Raspbian, que é uma distribuição Linux customizada para o Raspberry (“Raspbian”, 2018), no qual oferece uma boa documentação.

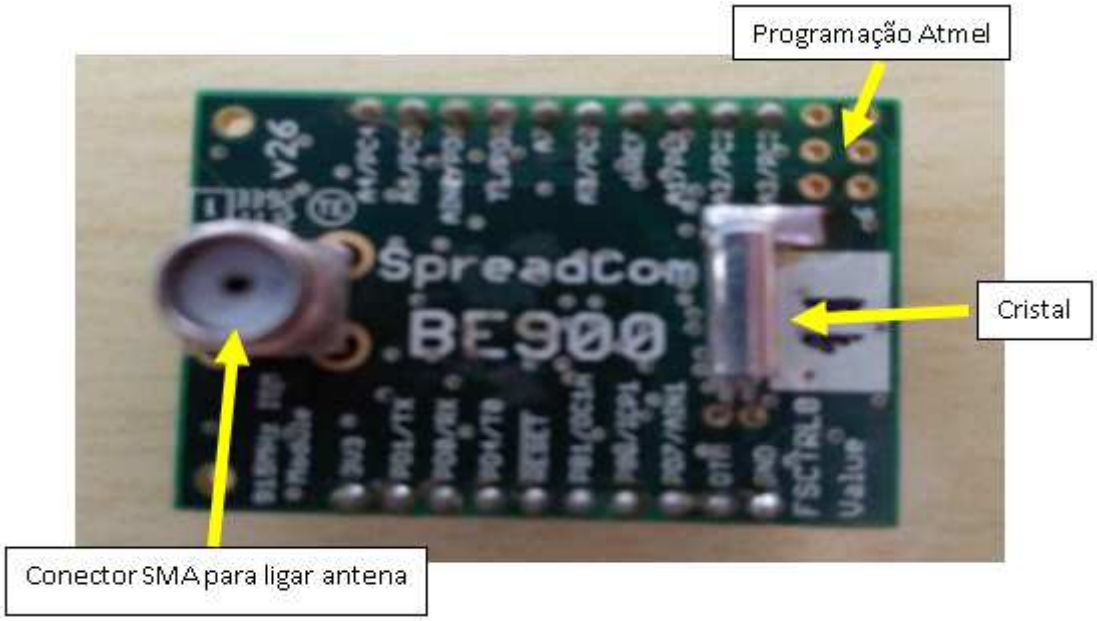
Para gerenciar as informações recebidas pelo IoT-PM, foi instalado no Raspberry a ferramenta Zabbix. Este *software* Open Source (HORST; PIRES; DÉO, 2015) consegue armazenar e tratar os dados coletados pelos NS e enviados para o IoT-PM através do *software* em Python. Além disto, o Zabbix é capaz de exibir as informações recebidas de várias formas, inclusive no formato de gráficos (HORST; PIRES; DÉO, 2015), o que auxiliou os testes e comparações dos experimentos neste trabalho.

Já para o *software*, usado para coletar as informações enviadas pelos NS e para enviar as informações para o Zabbix, optou-se pela linguagem *Python* (“Python”, 2018). O *Python* é uma linguagem *open-source*, interpretada e compilada, com diversas bibliotecas disponíveis para as mais diversas necessidades, inclusive para a segurança, que é o foco deste trabalho.

6.3. Nó Sensor, Estação Rádio Base e *Sniffer*

Para o NS, o ERB e o *sniffer*, foi utilizada uma plataforma desenvolvida em pilha de camadas (Aplicação, Transporte, Rede, MAC e Física). Como já citado, esta plataforma é o Radiuino, que é uma biblioteca desenvolvida para o Arduino (“Radiuino”, 2018), utiliza linguagem de programação C e *hardware* e *software* de código aberto (“História do Radiuino - Radiuino”, [s.d.]), o que proporcionou o desenvolvimento do método de cifragem, assim como propicia também a criação de novos recursos. Tanto o NS, a ERB e o *sniffer,* utilizam módulos de rádio BE-900, que é um transceptor que atende às regulamentações da ANATEL (“Radioit”, [s.d.]). Os módulos trabalham na faixa de frequência de 902-907,5MHz e 915-928MHz, utiliza a modulação 2FSK e possui taxa de transmissão de até 250 Kbps (“Radioit”, [s.d.]). O BE-900 utiliza o micro controlador ATmega328 e um transceptor CC1101 (TRANSCEIVER; CC, 2010). Na Figura 14, é mostrado o transceptor BE-900.

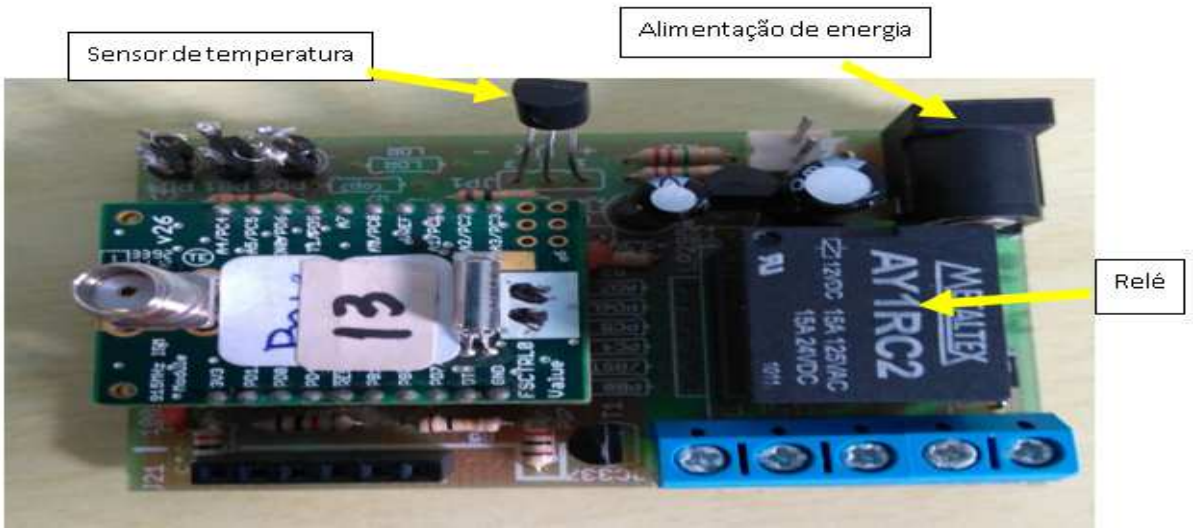
Figura 14 - Transceptor BE-900



Fonte – elaboração própria

No NS, foi utilizada uma placa de desenvolvimento DK-101, que é alimentada por uma fonte de energia de 5V. Esta placa possui embutido os sensores de temperatura e luminosidade. Na Figura 15, é mostrada a placa do DK-101.

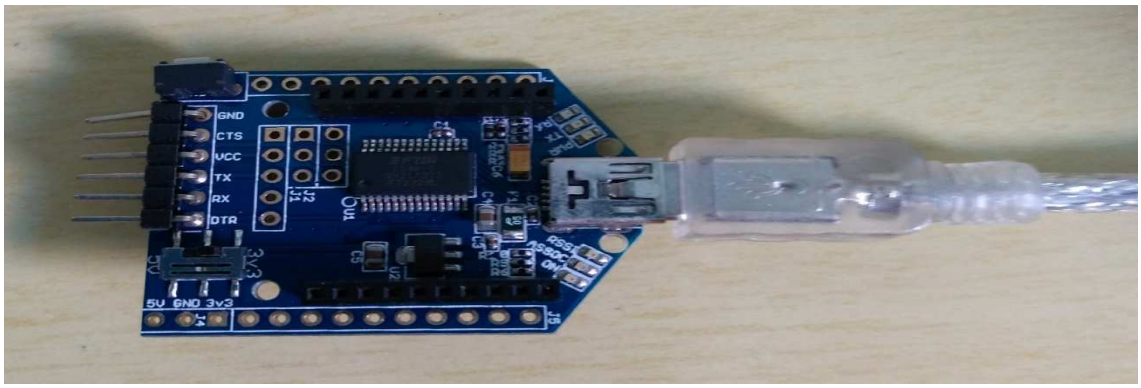
Figura 15 - Placa de desenvolvimento DK-101



Fonte – elaboração própria

A placa DK-101 não possui saída USB para que seja possível a comunicação com um computador. Por este motivo, neste trabalho, foi utilizado na ERB (conectado ao IoT-PM) e no *sniffer* (conectado ao notebook), foi utilizado o módulo conversor USB para RS232, chamado UasrtSBee (SEEEDSTUDIO, [s.d.]). Este equipamento possui saída mini USB. Na Figura 16, é mostrado o módulo UasrtBee.

Figura 16 - Módulo UartSBee



Fonte: elaboração própria

6.4. Computador conectado ao *sniffer*

Conforme já citado anteriormente, para a realização deste trabalho, um *sniffer* foi desenvolvido para emular um possível atacante na RSSF. Este elemento, assim como a ERB precisa estar conectada em um computador via

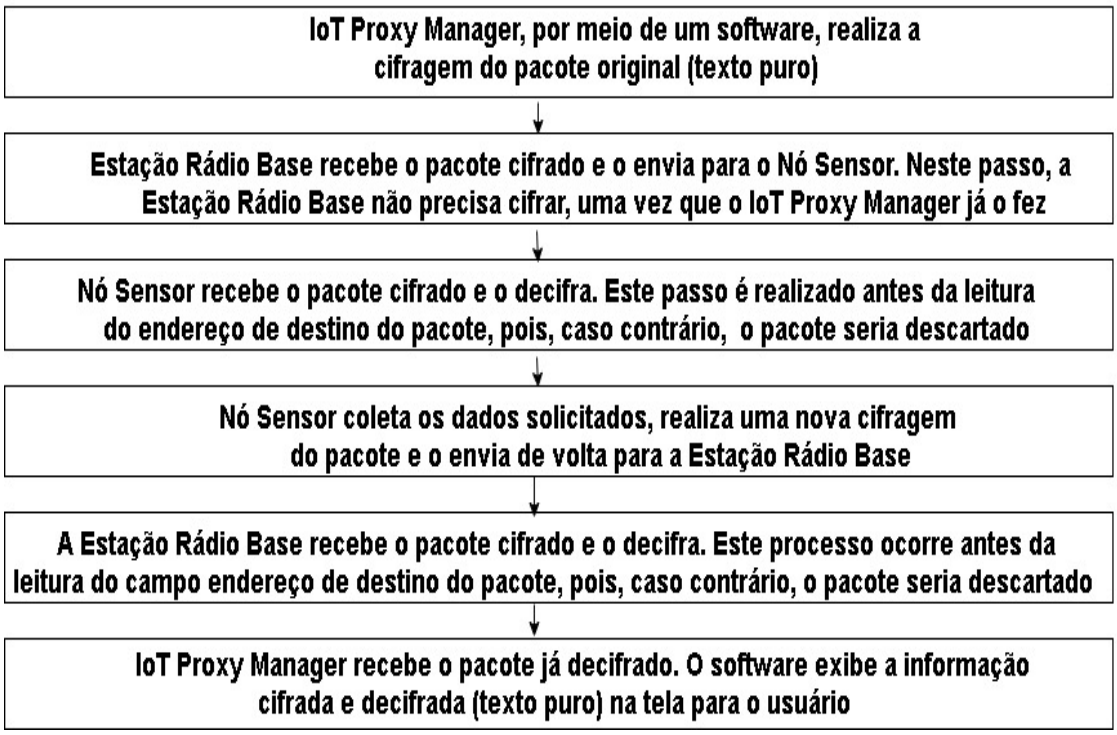
USB para que possa enviar os pacotes capturados ao longo da RSSF. Como a função do computador interligado ao *sniffer* é de apenas armazenar e exibir as informações capturadas, e não há neste caso uma preocupação com o seu tamanho físico, foi utilizado um notebook com a seguinte configuração:

- 512 GB de disco rígido;
- 4 GB de memória RAM;
- Processador Intel Core I5;
- Sistema Operacional Windows 10;

6.5. Implementação da cifragem XOR

O método de cifragem desenvolvido para o trabalho utilizando o XOR, tem como objetivo embaralhar todos os bits que compõem 52 *bytes* do pacote Radiuino, utilizando uma senha, que é inserida nos firmwares do NS e da ERB, assim como no *software Python*, no IoT-PM. Como já citado, estes três componentes da RSSF participam do processo de cifragem. Por questão de clareza, a Figura 17 mostra novamente as etapas do processo de cifragem.

Figura 17 - Etapas do processo de cifragem na RSSF



Fonte: elaboração própria

O IoT-PM é o elemento que inicia a comunicação na RSSF e consequentemente, é também o dispositivo que começa o processo de cifragem. O *software Python* que foi desenvolvido para o IoT-PM com a cifragem XOR é mostrado no Apêndice A.

Uma vez o pacote cifrado pelo IoT-PM, a ERB o envia para o NS. Este, quando recebe o pacote, precisa decifrá-lo. Como mostrado na Figura 12, a plataforma Radiuino possui um *byte* (*byte* 10) reservado para identificar a origem do pacote transmitido na RSSF. Sendo assim, a decifragem do pacote precisa ser realizada antes da leitura deste *byte*, pois, em caso contrário, o NS não conseguiria ler esta informação e por consequência, descartaria este pacote. O *firmware* do NS sem a cifragem pode ser obtido em (“Radiuino”, 2018). As linhas do código com o processo de decifragem são mostradas no Apêndice B.

Como mostrado em (“Radiuino”, 2018), o *firmware* do NS possui duas funções importantes que foram levadas em consideração na estratégia. A função **RECEIVE**, que recebe o pacote e a função **SEND**, que envia o pacote. No caso, as linhas de código de decifragem do firmware do NS (Apêndice B), foram inseridas dentro da função **RECEIVE**.

Após a decifragem, o NS coleta a grandeza solicitada pelo *software Python*, e antes de enviá-la de volta, deve realizar novamente a cifragem do pacote. As linhas utilizadas no processo de cifragem feito pelo NS antes do envio do pacote, são mostradas no Apêndice C e foram inseridas dentro da função **SEND**.

Como já dito anteriormente, a ERB não faz nenhuma ação no processo de cifragem, no envio do pacote. No entanto, quando ele recebe o pacote de volta do NS, precisa obrigatoriamente realizar sua decifragem, pois, assim como ocorre no recebimento por parte do NS, se a ERB não realizar a decifragem antes da leitura do campo de identificação do destino do pacote (*byte* 10), descartará o pacote, pois não conseguiria ler esta informação. As linhas de decifragem mostradas no Apêndice D, são acrescentadas dentro da função **RECEIVE.** Ao contrário do NS, a ERB não tem nenhuma linha de código de

cifragem dentro da função **SEND,** pois, como já dito, no envio do pacote, a ERB não faz nenhuma cifragem.

A ERB, uma vez que decifra o pacote, o envia para o IoT-PM. O *software Python*, não faz nenhum processo do método de cifragem ao receber o pacote, apenas exibindo-o.

6.5.1. *Sniffer* no método XOR

Com relação ao *sniffer*, desenvolvido para o método XOR, foi utilizado um *hardware* idêntico a ERB, no qual seu *firmware* foi modificado para que este elemento pudesse capturar as informações na RSSF de forma decimal. O Apêndice E mostra as linhas de código para exibição das informações em formato decimal, no *sniffer.* Neste caso, foram escolhidos os pacotes AD0 [0], AD0 [1] e AD0 [2], que no transceptor BE-900, são responsáveis pela coleta da grandeza temperatura. Estes campos, correspondem aos bytes de número 16, 17 e 18 do protocolo Radiuino (Figura 12).

Um detalhe importante do *sniffer*, é que ele deve capturar quaisquer pacotes que estejam no seu alcance dentro da RSSF. Para este fim, as linhas de código do *firmware* da ERB, disponível em (“Radiuino”, 2018) que verificam o endereço de destino do pacote, foram comentadas. Com isto, o *sniffer* não faz checagem alguma desta informação e “ouve” todos os pacotes transmitidos na RSSF.

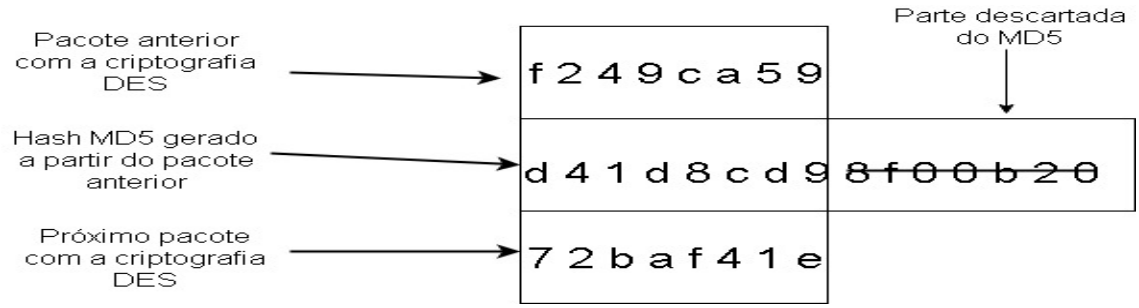
6.6. Implementação da cifragem OTP

Diferentemente do método XOR, no qual todos os bits de todo pacote original são embaralhados com o auxílio de uma senha, no segundo método, é utilizada uma técnica de criptografia simétrica (DES) juntamente com uma função de *hash* (MD5), formando o método de cifragem OTP.

Para o desenvolvimento deste método, foram utilizadas bibliotecas no Arduino para o algoritmo de criptografia DES e para o *hash* MD5 no NS e na ERB. Bibliotecas do DES e do MD5 também foram utilizadas no *software* em *Python* no IoT-PM.

Neste processo, o pacote de 52 *bytes* gerado pela plataforma Radiuino é dividido em pacotes menores de 64 bits (8 *bytes*) cada um. Este tamanho de 64 bits na divisão dos pacotes se dá, pelo fato da criptografia simétrica DES realizar a cifragem com esta quantidade exata de bits. O sistema de criptografia simétrica DES irá atuar juntamente com a função *hash* MD5. Um detalhe importante é que a função MD5 gera *hash* de tamanho de 128 bits, portanto para a realização deste método, o processo leva em consideração apenas os primeiros 64 bits do MD5 para se adequar ao tamanho do DES, ignorando assim os últimos 64 bits do *hash*. Um exemplo é mostrado na Figura 18.

Figura 18 - Parte do hash MD5 ignorada no processo



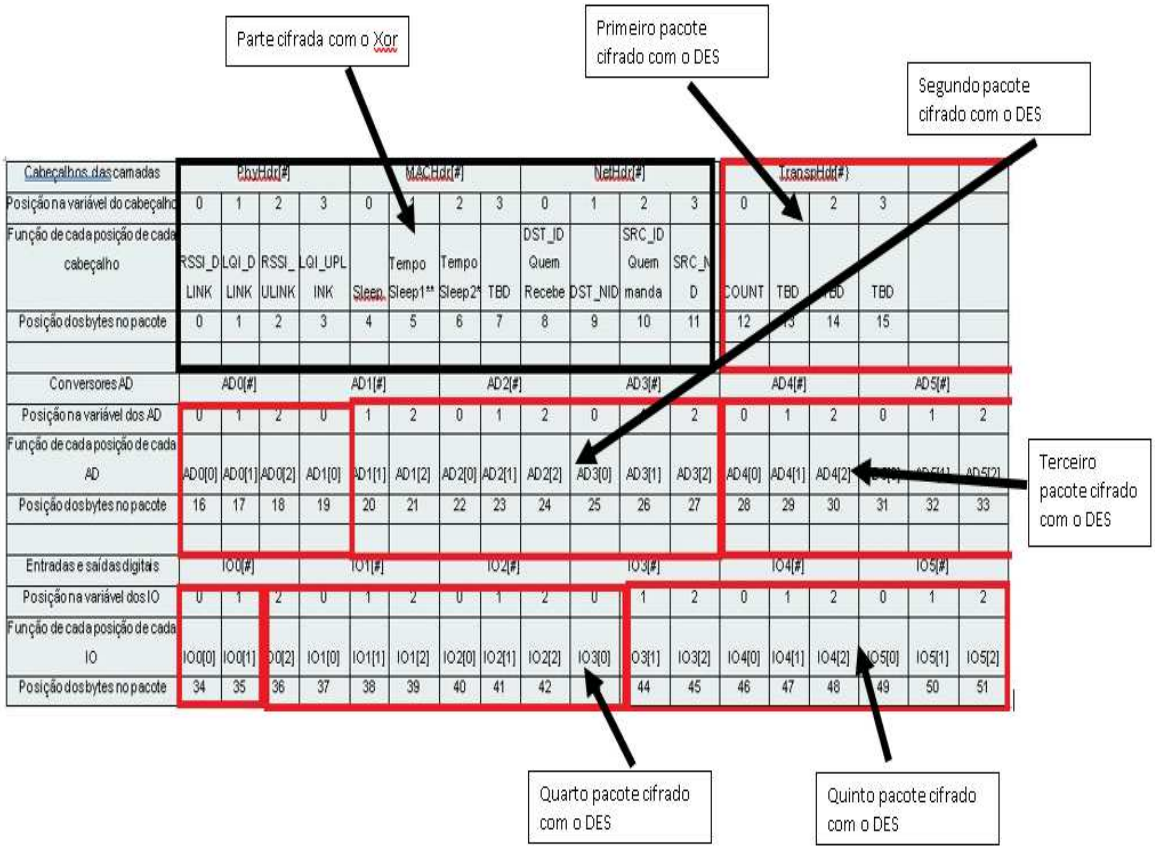
Fonte – elaboração própria

No desenvolvimento do método, percebeu-se que não seria possível realizar esta divisão proposta no OTP de 64 bits (8 *bytes*) ao longo de todo o pacote do Radiuino. Isto se deu pelo fato da plataforma Radiuino gerar pacotes de 52 *bytes* de tamanho (“Radiuino”, 2018) e este número não ser múltiplo de 8 *bytes* (64 bits). Sendo assim, optou-se por implementar o processo OTP nos últimos 40 bytes do pacote (*byte* 12 ao *byte* 51), gerando assim um total de 5 pacotes criptografados com o DES, cada um deles realizado com uma função *hash* distinta.

Já os primeiros 12 *bytes* do pacote Radiuino (*byte* 0 ao *byte* 11), serão cifrados utilizando o método idêntico à primeira solução de cifragem deste trabalho. Com isto, um NS intermediário, que não é o destino de um pacote, uma vez que decifrar o XOR nos 12 primeiros *bytes*, irá ler o identificador de destino do pacote (*byte* 10 do Radiuino). Neste caso, se este não for o destino do pacote, não irá utilizar *hardware* (processamento, memória, energia, etc...) para decifrar o restante do pacote, que estará cifrado com o OTP. Isto é interessante, pois

poupa recursos dos NS nas RSSF, que são limitados. Na Figura 19, é mostrado como ficou a divisão do pacote Radiuino com o método OTP.

Figura 19 - Divisão de cifragem do OTP no pacote Radiuino



Fonte – elaboração própria

Portanto, enquanto no primeiro método de cifragem, a operação XOR é realizada em todos os bits dos 52 *bytes* do pacote transmitido, no método OTP, este processo ocorre apenas nos 12 primeiros *bytes*. O intuito foi o desenvolvimento de métodos de cifragem que pudessem atender diferentes níveis de necessidade de segurança nas RSSF. Enquanto no método XOR, ocorre apenas um embaralhamento das informações, no método de cifragem com o OTP, algoritmos de criptografia são utilizados no processo, o que o torna mais complexo do que o primeiro método.

O Apêndice F mostra o código principal em Python (pybaseotp.py) do IoT-PM para o método de cifragem OTP. Uma observação importante é que por questões de simplificação deste código na linguagem de programação Python, o mesmo foi dividido em dois códigos, no qual um deles realiza as tarefas de

criptografia, chamado neste trabalho de pyotp.py, que é mostrado no Apêndice G, e este, é acionado pelo código principal.

Como mostrado em (“Radiuino”, 2018), o *firmware* do NS possui duas funções importantes que foram levadas em consideração na estratégia. A função **RECEIVE**, que recebe o pacote e a função **SEND**, que envia o pacote. No caso, as linhas de código de decifragem do firmware do NS (Apêndice H), foram inseridas dentro da função **RECEIVE**.

Após a decifragem, o NS coleta a grandeza solicitada pelo *software Python*, e antes de enviá-la de volta, deve realizar novamente a cifragem do pacote. As linhas utilizadas no processo de cifragem feito pelo NS antes do envio do pacote, são mostradas no Apêndice I e foram inseridas dentro da função **SEND**.

Como já dito anteriormente, a ERB não faz nenhuma também ação no processo de cifragem, no envio do pacote neste método OTP. No entanto, quando ele recebe o pacote de volta do NS, precisa obrigatoriamente realizar sua decifragem, pois, assim como ocorre no recebimento por parte do NS, se a ERB não realizar a decifragem antes da leitura do campo de identificação do destino do pacote (*byte* 10), descartará o pacote, pois não conseguiria ler esta informação. As linhas de decifragem mostradas no Apêndice J, são acrescentadas dentro da função **RECEIVE.** Ao contrário do NS, a ERB não tem nenhuma linha de código de cifragem dentro da função **SEND,** pois, como já dito, no envio do pacote, a ERB não faz nenhuma cifragem.

A ERB, uma vez que decifra o pacote, o envia para o IoT-PM. O *software Python*, não faz nenhum processo do método de cifragem ao receber o pacote, apenas exibindo-o.

6.6.1. *Sniffer* no método OTP

Com relação ao *sniffer*, desenvolvido para o método OTP, assim como no *sniffer* para o método XOR, também foi utilizado um *hardware* idêntico a ERB, no qual seu *firmware* foi modificado para que este elemento pudesse capturar as informações na RSSF de forma decimal. O Apêndice K mostra as linhas de

código para exibição das informações em formato decimal, no *sniffer.* Neste caso, também foram escolhidos os pacotes AD0 [0], AD0 [1] e AD0 [2], que no transceptor BE-900, são responsáveis pela coleta da grandeza temperatura. Estes campos, correspondem aos bytes de número 16, 17 e 18 do protocolo Radiuino (Figura 12).

Um detalhe importante do *sniffer*, é que ele deve capturar quaisquer pacotes que estejam no seu alcance dentro da RSSF. Para este fim, as linhas de código do *firmware* da ERB, disponível em (“Radiuino”, 2018) que verificam o endereço de destino do pacote, foram comentadas. Com isto, o *sniffer* não faz checagem alguma desta informação e “ouve” todos os pacotes transmitidos na RSSF.

É necessário avaliar os ambientes nos quais a RSSF vai ser implementada, para mitigar situações de segurança na transferência de dados e assim, optar por um dos dois métodos descritos.

7. TESTES REALIZADOS E RESULTADOS

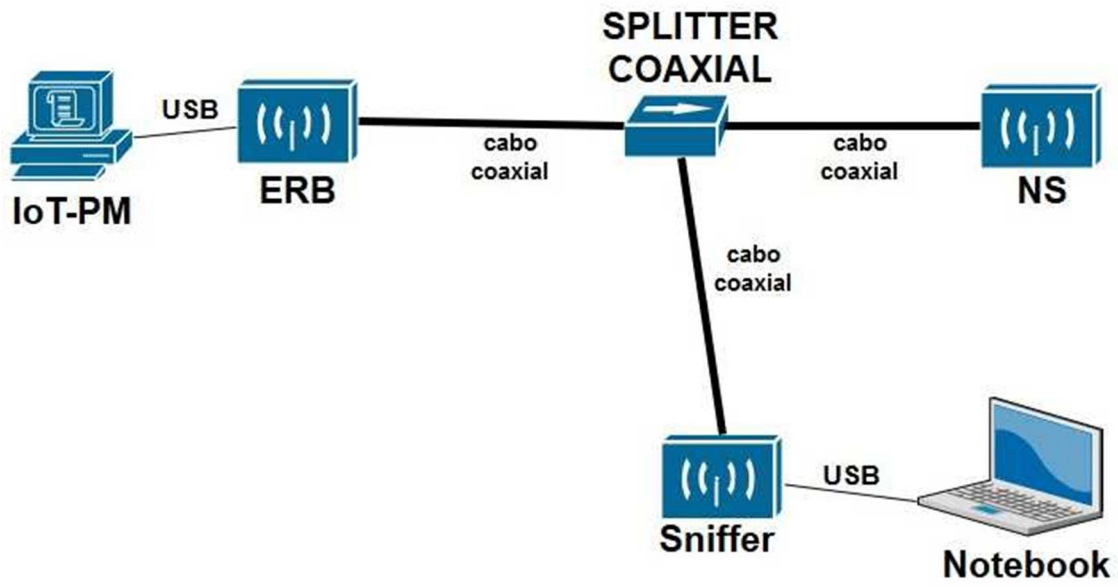
Para a realização dos testes para este trabalho, foi montada uma bancada de emulação, na qual foi utilizada um *splitter* coaxial para fazer a conexão entre os dispositivos da rede. Como o objetivo do trabalho é mostrar o funcionamento dos métodos de cifragem desenvolvidos e o consumo de recursos computacionais de cada método, entendeu-se que o meio de transmissão em um primeiro momento não seria relevante. Este cenário foi escolhido por ser um ambiente no qual a comunicação entre os elementos ocorre de forma satisfatória, sem interferência ou qualquer outro problema de conectividade, uma vez que o objetivo do trabalho não era avaliar o desempenho da interface rádio.

No cenário montado para os testes, foi conectada a ERB ao IoT-PM. A ERB coleta os dados da RSSF e os envia para o IoT-PM. O IoT-PM por sua vez, por meio do software de gerência de redes Zabbix, tem a possibilidade de realizar o tratamento destes dados. Além disto, como mostrado na Figura 11, o IoT-PM participa dos dois métodos de cifragem

O NS será responsável por capturas as grandezas na RSSF. Para realização dos testes para este trabalho, a temperatura foi a grandeza escolhida. Estas informações estão armazenadas nos *bytes* AD0[16], AD0[17] e AD0[18] de acordo com a combinação de pinagens do kit DK-101, utilizado no NS. O NS tem participação fundamental nos métodos de cifragem desenvolvidos, uma vez que em ambos, terá que decrifrar os pacotes ao recebe-lo, e realizar uma nova cifragem antes do seu envio.

O *sniffer* que foi desenvolvido para capturar os pacotes na RSSF e exibir as informações capturadas, foi conectado à um notebook. Como já dito, o objetivo do desenvolvimento do *sniffer* é ouvir as informações cifradas transmitidas pelos elementos da RSSF. Por este motivo, ele foi conectado a um notebook, simulando um possível invasor dentro da RSSF. Na Figura 20, é mostrada a bancada de emulação, sem a utilização de atenuadores.

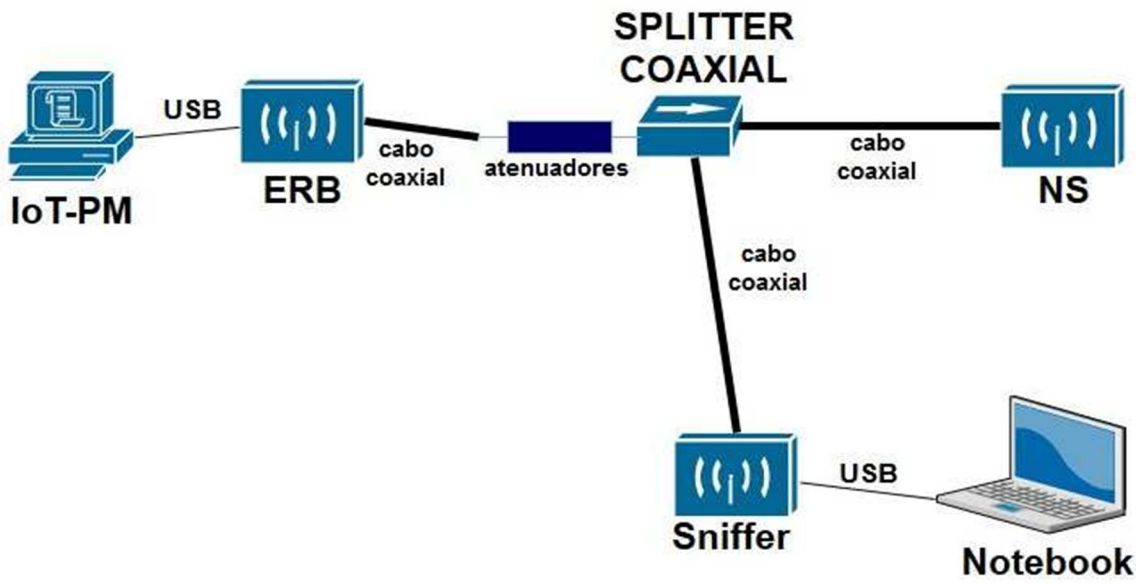
Figura 20 – Estrutura da bancada de emulação sem atenuadores



Fonte – elaboração própria

Já na Figura 21, é mostrada a estrutura da bancada de emulação, com a utilização dos atenuadores.

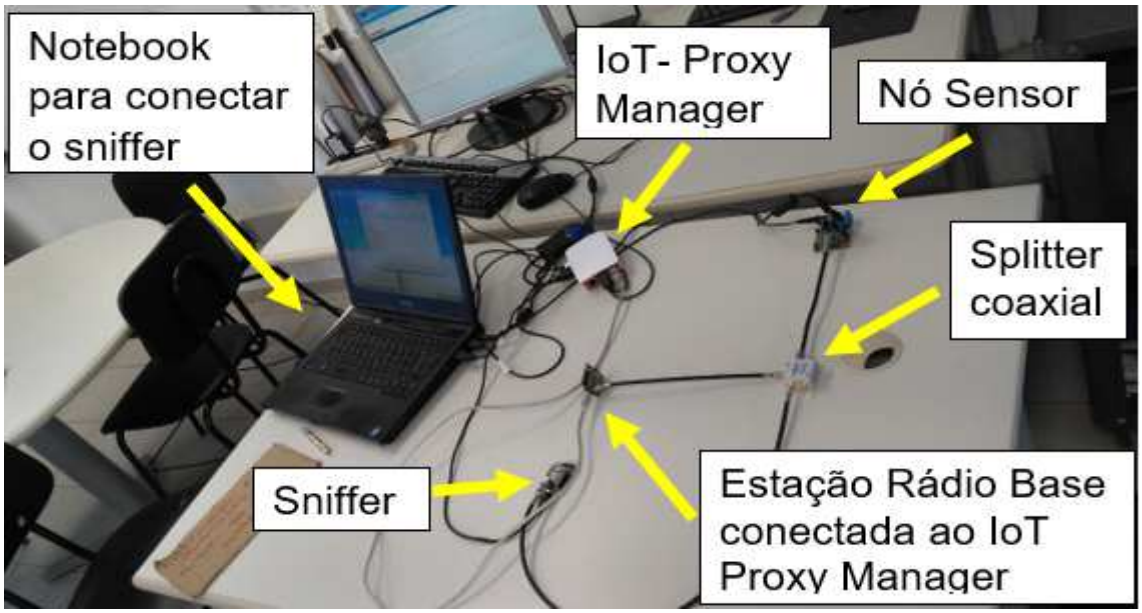
Figura 21 – Estrutura da bancada de emulação com atenuadores



Fonte – elaboração própria

Na Figura 22, é mostrada uma foto da bancada de emulação criada para a realização dos experimentos.

Figura 22 - Bancada de emulação criada para os experimentos



Fonte – elaboração própria

Durante os experimentos, foram realizadas medições de algumas métricas da RSSF, como: taxa de erro de pacote (PER), tempo de processamento do NS e tempo de resposta da RSSF. O objetivo é comparar estas características com o intuito de demonstrar o nível de impacto de desempenho, que os métodos de cifragem desenvolvidos neste trabalho podem gerar na RSSF, e assim, verificar a possibilidade de utilização de cada solução. Em todos os experimentos, o *sniffer* realizou a interceptação do pacote transmitido na RSSF em ambas as direções.

Nos três cenários de testes realizados (sem a utilização de cifragem, com a cifragem XOR e com o OTP), foram realizadas coletas com a utilização de atenuadores para conexão coaxial, mas também sem a utilização dos atenuadores. O objetivo é emular um certo nível de atenuação e assim, comparar as métricas com as medidas coletadas sem e com a utilização dos atenuadores. Na Figura 23, são mostrados os atenuadores que foram utilizados nos experimentos.

Figura 203 - Atenuadores utilizados nos experimentos

Fonte – elaboração própria

O transceptor utilizado no rádio BE-900 é o CC1101. Este elemento possui ums sensibilidade de -112 dBm (TRANSCEIVER; CC, 2010). A finalidade dos testes em um primeiro momento, não era medir a qualidade de sinal e sim avaliar como o fato de trabalhar com baixa potência, poderia afetar a estabilidade e desempenho dos códigos de cifragens desenvolvidos. Para este motivo, foi utilizada a atenuação de 80 dBm, neste caso, não haverá perda de pacotes em função das perdas do canal.

Em todas as situações possíveis, foram realizadas 50.000 coletas. Optou-se por este número para demonstrar a estabilidade e robustez dos sistemas e da comunicação entre os elementos da RSSF. Os testes tiveram duração de 6 dias, nos quais houve transição de temperatura (grandeza escolhida nos testes) e consequentemente alterações nos valores transmitidos com a utilização das cifragens e sem nenhuma segurança.

Em todos os testes realizados, foram calculados o menor tempo, o maior tempo e a média de tempo em milissegundos. Além destas informações, foram calculadas também o desvio padrão e o coeficiente de variação das coletas.

O desvio padrão é uma medida de dispersão em torno da média de uma variável aleatória (MULHOLLAND, 2007). Já o coeficiente de variação é uma medida muito útil para caracterizar, com maior rigor possível a dispersão dos conjuntos. O coeficiente de variação está sempre relacionado ao valor médio de um conjunto (MULHOLLAND, 2007). A seguinte fórmula é utilizada para calcular o coeficiente de variação:

$$CV = \frac{desvio\ padrão}{média} \times 100$$

O coeficiente de variação é uma medida expressa em porcentagem, por isto, é multiplicado por 100 (MULHOLLAND, 2007).

Portanto, o objetivo de se calcular o desvio padrão e o coeficiente de variação, foi mostrar que o sistema é estável, com pouca variação de tempo entre as coletas.

7.1. Testes sem a utilização de cifragem

Como explicado na Figura 17, o IoT-PM, conectado à ERB, inicia a comunicação na RSSF, solicitando informações ao NS. Um *sniffer* atua entre estes elementos, realizando a captura dos pacotes transmitidos pela RSSF em ambas as direções da comunicação, emulando um possível invasor. Na Figura 24, é mostrado o resultado da uma amostragem de capturas realizadas pelo *sniffer* na RSSF, sem a utilização de cifragem.

Figura 24 - Captura dos pacotes realizada pelo *sniffer* na RSSF, sem utilizar cifragem

```
Radiuino! Sniffer
Temperatura: 0
Temperatura: 251
Temperatura: 0
Temperatura: 250
```

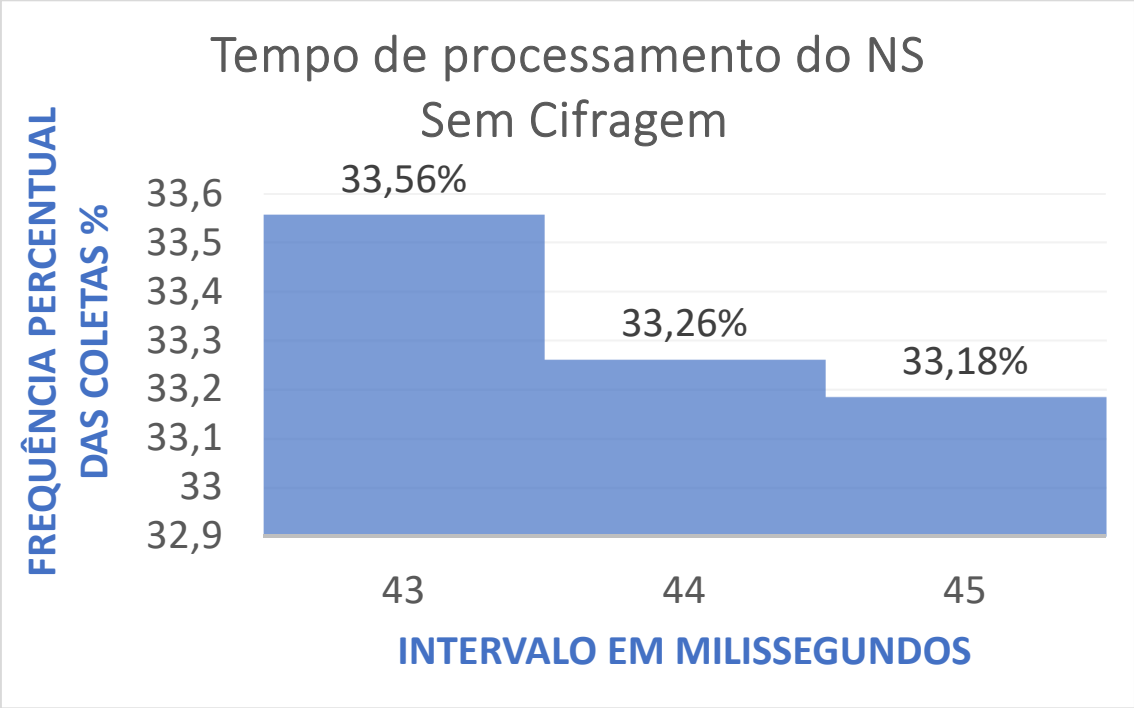
Fonte: elaboração própria

Na Figura 24 observa-se que a temperatura no momento da medição está variando entre 25,00 °C e 25,10 ºC. As linhas que constam “Temperatura: 0” são capturas desta medida, realizadas pelo *sniffer* no sentido IoT-PM/ERB para o NS, no qual os *bytes* referentes à temperatura estão sem esta informação, uma vez que a mesma não foi ainda coletada pelo NS.

O tempo de processamento do NS também foi testado. Esta métrica leva em consideração, o tempo gasto pelo NS desde quando ele recebe o pacote, até o momento em que ele o envia de volta para ao IoT-PM/ERB. Na medição desta

métrica, não faz diferença a utilização ou não de atenuação. Na Figura 25, é mostrado um gráfico com as informações referentes ao tempo de processamento do NS, sem utilização de cifragem. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de processamento do NS em milissegundos.

Figura 25 - Tempo de processamento do NS, sem a utilização de cifragem



Fonte: elaboração própria

Neste teste, o tempo de processamento mínimo no NS foi de 43 milissegundos, o tempo de processamento máximo 45 milissegundos, o tempo médio de processamento foi de 44 milissegundos, o desvio padrão foi de 0,81 milissegundos e o coeficiente de variação foi de 1,83%. Na Figura 26, é mostrado o NS exibindo uma amostragem do tempo de processamento.

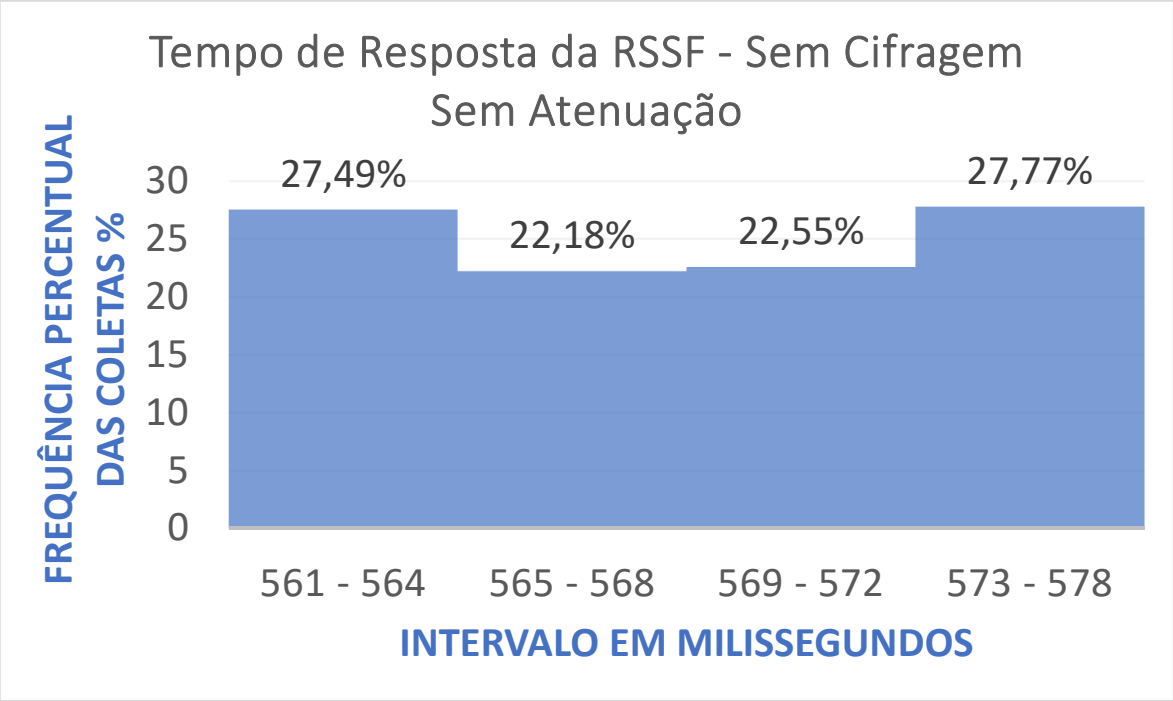
Figura 26 - Tempo de processamento exibida pelo NS

```
Radiuino! Sensor
Tempo:  43
Tempo:  43
Tempo:  45
Tempo:  43
```

Fonte: elaboração própria

O tempo de resposta da RSSF também foi medido nos testes realizados. Esta métrica, calcula o tempo gasto na transmissão, desde que o pacote sai do IoT-PM, passa pela ERB, chega no NS e volta no caminho inverso, até chegar novamente no IoT-PM. Na Figura 27, é mostrado o gráfico do tempo de resposta da RSSF sem cifragem e sem atenuação. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de processamento do NS em milissegundos.

Figura 27 - Tempo de resposta da RSSF sem cifragem e sem atenuação

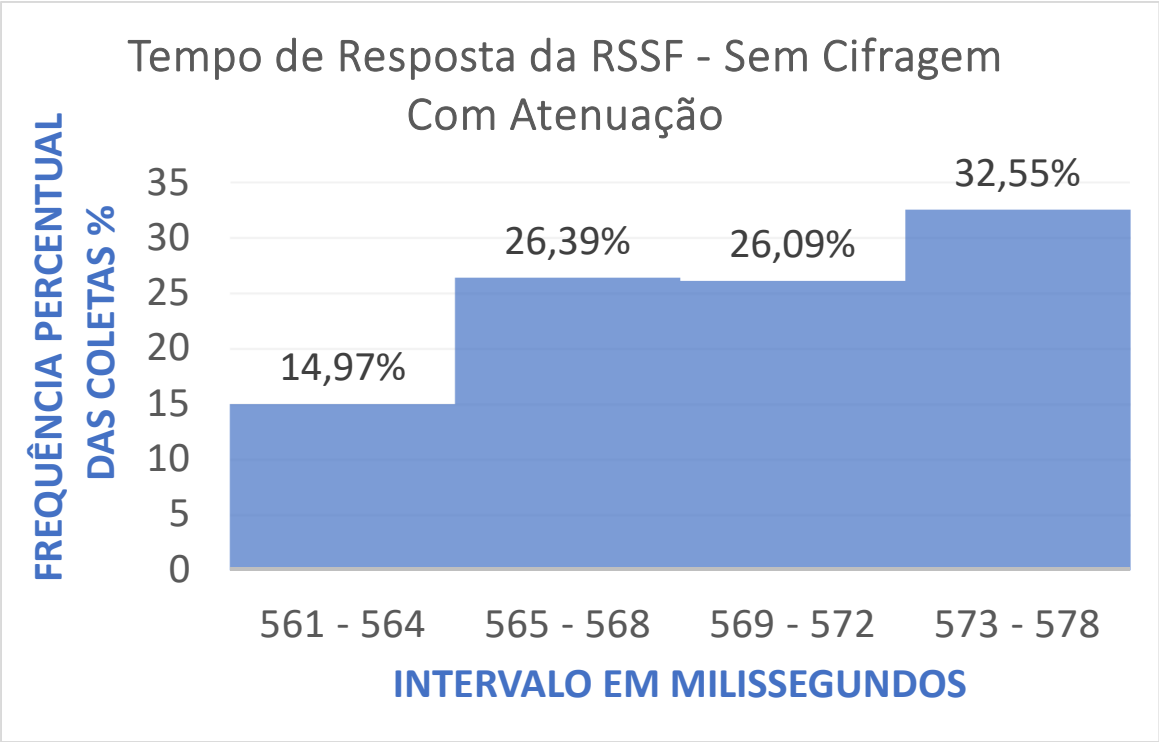


Fonte: elaboração própria

Neste teste, o tempo de resposta da RSSF mínimo foi de 561 milissegundos, o tempo de processamento máximo foi de 578 milissegundos, o tempo médio de processamento foi de 569,61 milissegundos, o desvio padrão foi de 5,18 milissegundos e o coeficiente de variação foi de 0,91%.

O mesmo teste foi realizado com atenuação. Na Figura 28, é mostrado o gráfico de tempo de resposta da RSSF sem cifragem e com atenuação. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de processamento do NS em milissegundos.

Figura 28 - Tempo de resposta da RSSF sem cifragem e com atenuação



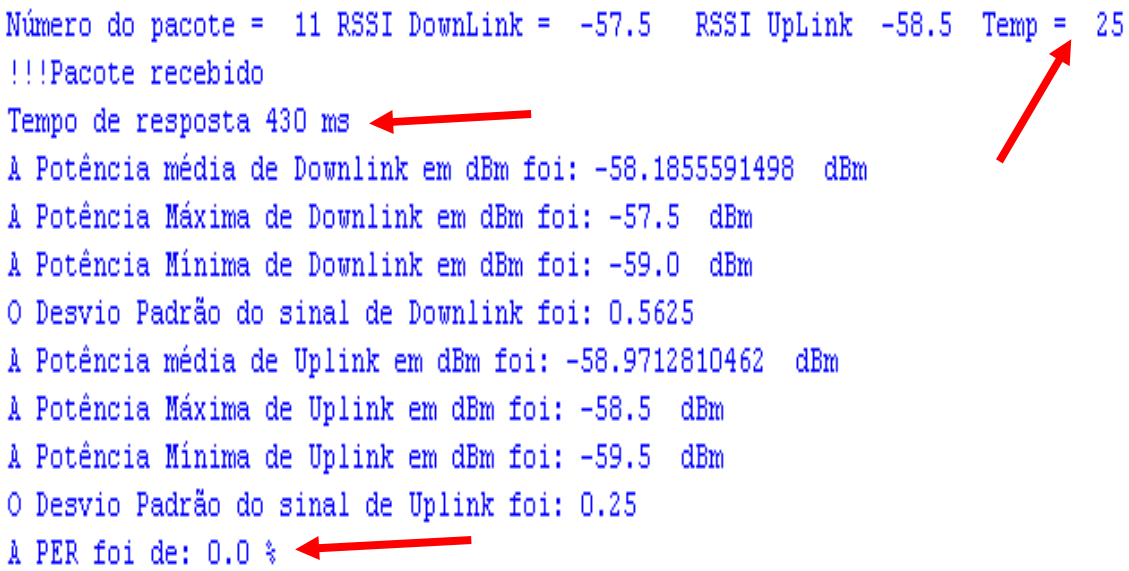
Fonte: elaboração própria

Neste teste, o tempo de resposta da RSSF mínimo também foi de 561 milissegundos, o tempo de processamento máximo foi também de 578 milissegundos, portanto, idênticos aos testes sem atenuação. No entanto, houve um aumento no tempo médio de processamento, que foi de 570,77 milissegundos. O desvio padrão foi de 4,52 milissegundos e o coeficiente de variação foi de 0,79%.

Pode-se perceber que no tempo de processamento da RSSF sem cifragem e sem a utilização de atenuação, o tempo foi mais homogêneo, com uma proporção maior de coletas nas faixas de 561 até 564 milissegundos e de 573 até 578 milissegundos, havendo uma distância maior da média de 569,61 milissegundos. Já nos testes sem cifragem e com atenuação, pode-se perceber que houve uma proporção menor na faixa de 561 até 564 milissegundos, uma proporção maior na faixa de 573 até 578 milissegundos e houve uma homogeneidade mais próxima à média destas coletas, que foi de 570,77 milissegundos.

O *software* desenvolvido em *Python* no IoT-PM, exibe uma série de informações do pacote recebido, inclusive com a temperatura original, o tempo de resposta da RSSF e ainda a taxa de erro de pacote (PER). Na Figura 29, é mostrado um exemplo de saída do *software Python,* que foi realizada antes dos testes, na bancada de emulação.

Figura 29 - Saída do *software* no IoT-PM

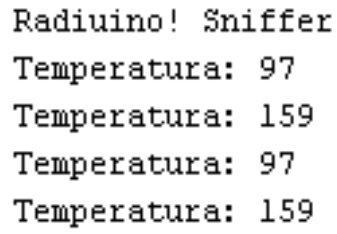


Fonte: elaboração própria

7.2. Testes com utilização da cifragem XOR

Os mesmos testes realizados sem a utilização de cifragem, foram feitos com o método de cifragem XOR. Na Figura 30, é mostrado uma amostra de captura de pacotes realizadas pelo *sniffer*.

Figura 30 - Captura dos pacotes realizada pelo *sniffer* na RSSF, com o XOR

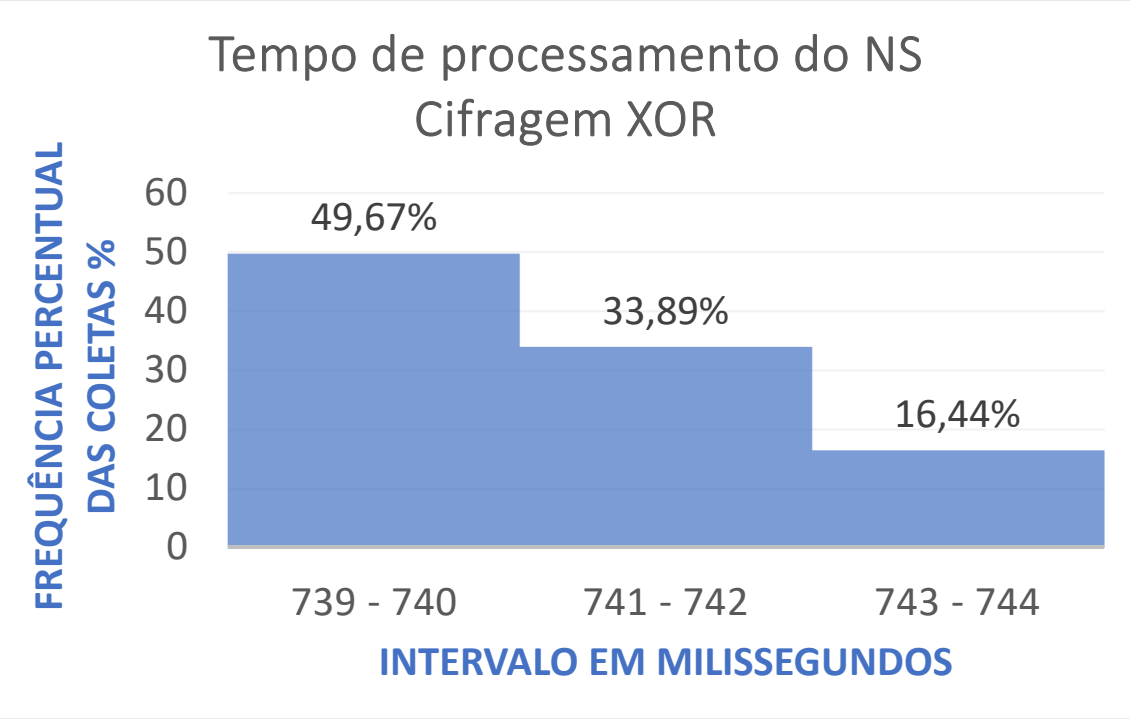


Fonte: elaboração própria

Na Figura 30 observa-se que, ao contrário do que ocorreu, quando o *sniffer*, capturou os pacotes sem a utilização de cifragem, as informações coletadas por meio da interceptação estão modificadas e inelegíveis para um elemento invasor na RSSF que não participa do processo de cifragem.

O tempo de processamento do NS foi testado também com a utilização deste método de cifragem. Na Figura 31, é mostrado o tempo de processamento do NS com a cifragem XOR. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de processamento do NS em milissegundos.

Figura 31 - Tempo de processamento do NS, com a utilização da cifragem XOR



Fonte: elaboração própria

Neste teste, o tempo de processamento mínimo no NS foi de 739 milissegundos, o tempo de processamento máximo 744 milissegundos, o tempo médio de processamento foi de 741,5 milissegundos, o desvio padrão foi de 1,70 milissegundos e o coeficiente de variação foi de 0,23%.

Na Figura 32, é mostrado o NS exibindo uma amostragem do tempo de processamento e processo do método XOR realizado pelo NS. Esta figura mostra também todo o processo que o NS faz: recebe o pacote de forma cifrada,

decifra o pacote, realiza a ação solicitada (no caso, coletar a temperatura), realiza uma nova cifragem, e envia os dados de volta para o IoT-PM/ERB.

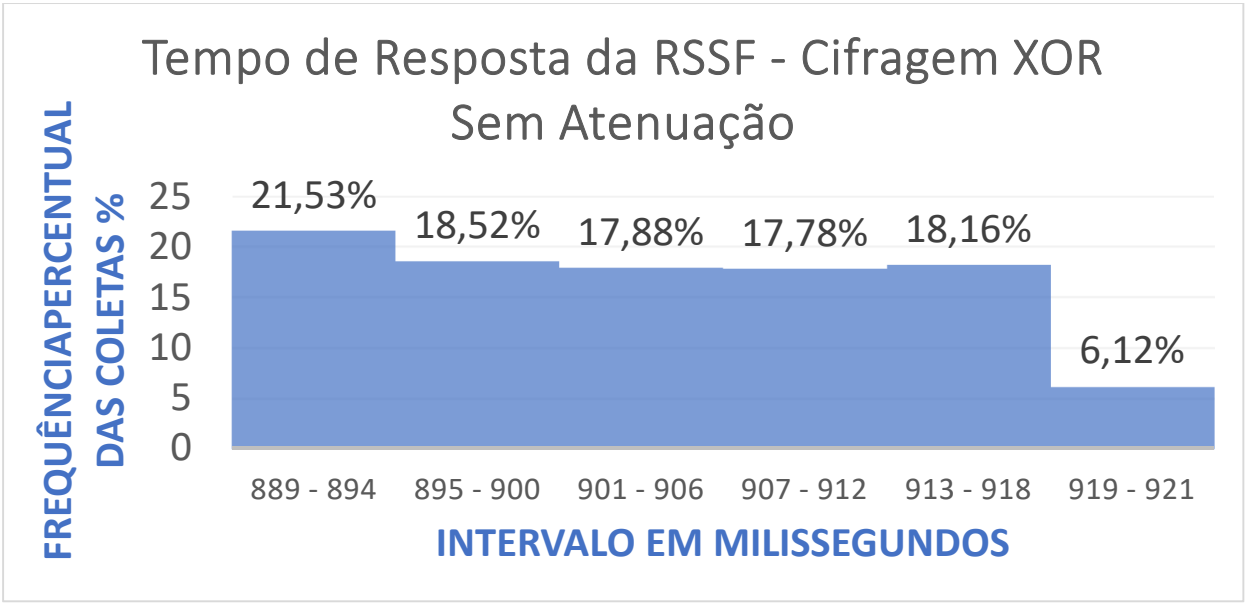
Figura 32 - Tempo de processamento e processos do método XOR realizados pelo NS

```
Radiuino! Sensor
Pacote recebido
70 6F 6E 74 69 66 69 63 60 75 6E 69 76 65 72 73 69 64 61 64 65 70 6F 6E 74 69 66 69 63 61 75 6E 69 76 65 72 73 69 64 61 64 65 70 6F 6E 74 69 66 69 63 61 75
Pacote recebido - apÃ³s xor
00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Enviando pacote - Antes XOR
21 81 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 FD 01 00 00 02 01 01 03 00 02 00 01 6A 00 01 E1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Enviando pacote - Depois XOR
51 EE 6E 74 69 66 69 63 61 75 6F 69 76 65 72 73 69 64 9C 65 65 70 6D 6F 75 6A 66 6B 63 60 1F 6E 68 97 65 72 73 69 64 61 64 65 70 6F 6E 74 69 66 69 63 61 75
Tempo:741
```

Fonte: elaboração própria

O tempo de resposta da RSSF também foi medido nos testes realizados tanto sem atenuação como também com atenuação. Na Figura 33, é mostrado o tempo de resposta da RSSF com a cifragem XOR, sem atenuação. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de resposta da RSSF em milissegundos.

Figura 33 - Tempo de resposta da RSSF com XOR e sem atenuação

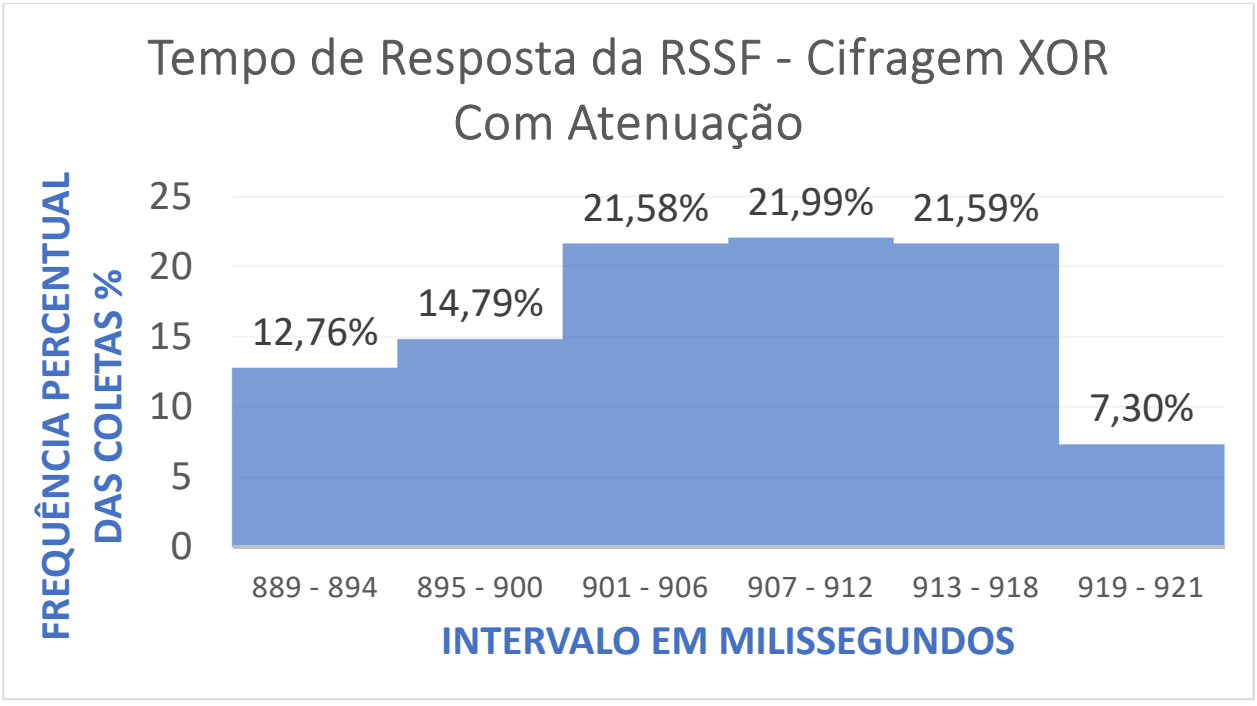


Fonte: elaboração própria

Neste teste, o tempo de resposta da RSSF mínimo também foi de 889 milissegundos, o tempo de processamento máximo foi de 921 milissegundos, o tempo médio de processamento foi de 904,91 milissegundos, o desvio padrão foi de 9,55 milissegundos e o coeficiente de variação foi de 1,05%.

O tempo de resposta da RSSF com a cifragem XOR, também foi medido com a utilização de atenuação. Na Figura 34, é mostrado o tempo de resposta da RSSF com a cifragem XOR, com atenuação. NA vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de resposta da RSSF em milissegundos.

Figura 34 - Tempo de resposta da RSSF com XOR e com atenuação



Fonte: elaboração própria

Neste teste, o tempo de resposta da RSSF mínimo foi de 889 milissegundos, o tempo de resposta máximo foi de 921 milissegundos, o tempo médio de resposta foi de 907,16 milissegundos, o desvio padrão foi de 8,81 milissegundos e o coeficiente de variação foi de 0,97%.

Pode-se perceber que no tempo de resposta da RSSF com a utilização da cifragem XOR e sem a utilização de atenuação, o tempo foi mais homogêneo, com uma proporção maior de coletas nas faixas de 889 até 894 milissegundos e

uma proporção bem menor na faixa de 919 até 921 milissegundos, no entanto, houve uma homogeneidade nos intervalos intermediários de tempo, inclusive o intervalo próximo da média de 904,91 milissegundos. Já nos testes com a cofragem XOR e com atenuação, pode-se perceber que também houve uma proporção menor na faixa de 889 até 894 milissegundos, no entanto, as maiores proporções foram percebidas entre as faixas 901 até 918 milissegundos, intervalos este onde se encontra a média deste teste, que foi de 907,16 milissegundos.

O *software* desenvolvido em *Python* no IoT-PM, exibe as informações do pacote recebido com uma série informações sobre o pacote, como a temperatura real, a temperatura cifrada, o tempo de resposta e a taxa de erro de pacote (PER) da RSSF. Na Figura 35, é mostrado um exemplo de saída do *software Python*, que foi realizada antes dos testes, na bancada de emulação.

Figura 35 – Exemplo de saída do software no IoT-PM com XOR

```
Tempo de resposta 1018 milissegundos
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 52
70 6f 6e 74 69 66 69 63 60 75 6e 69 76 65 72 73 69 64 61 64 65 70 6f 6e 74 69 66 69 63 61 75 6e 69 76 65 72
---*****---
!!!!!Pacote recebido!!!!!
24 82 23 82 00 00 00 00 00 00 01 00 0b 00 00 00 00 00 ff 01 00 00 02 01 01 03 00 02 00 00 f0 00 01 5f 00 00
Real:  25
Cifrada:  97
A PotÃªncia mÃ©dia de Downlink em dBm foi: -56.0  dBm
A PotÃªncia MÃ¡xima de Downlink em dBm foi: -56.0  dBm
A PotÃªncia MÃ-nima de Downlink em dBm foi: -56.0  dBm
O Desvio PadrÃ£o do sinal de Downlink foi: 0.0
A PotÃªncia mÃ©dia de Uplink em dBm foi: -56.5  dBm
A PotÃªncia MÃ¡xima de Uplink em dBm foi: -56.5  dBm
A PotÃªncia MÃ-nima de Uplink em dBm foi: -56.5  dBm
O Desvio PadrÃ£o do sinal de Uplink foi: 0.0
A PER foi de: 0.0 %
```

Fonte: elaboração própria

O *software* Zabbix captura as informações coletadas pelo NS e enviadas para o IoT-PM e as exibem. Na Figura 36, são mostradas as informações coletadas pelo Zabbix, que foi feita antes dos testes, na bancada de emulação.

Figura 36 - Dados exibidos pelo Zabbix no teste com a cifragem XOR

Fonte: elaboração própria

Conforme mostrado no *software* no IoT-PM (Figura 35) e no Zabbix (Figura 36), não houve perda de pacotes (PER) nos testes com a utilização do método de cifragem XOR.

7.3. Testes com utilização da cifragem OTP

Os mesmos testes realizados sem a utilização de cifragem e com a utilização da cifragem XOR, foram feitos com o método de cifragem OTP. Na Figura 37, é mostrada uma captura de pacotes realizadas pelo *sniffer*

Figura 37 - Captura dos pacotes realizada pelo *sniffer* na RSSF, sem cifragem

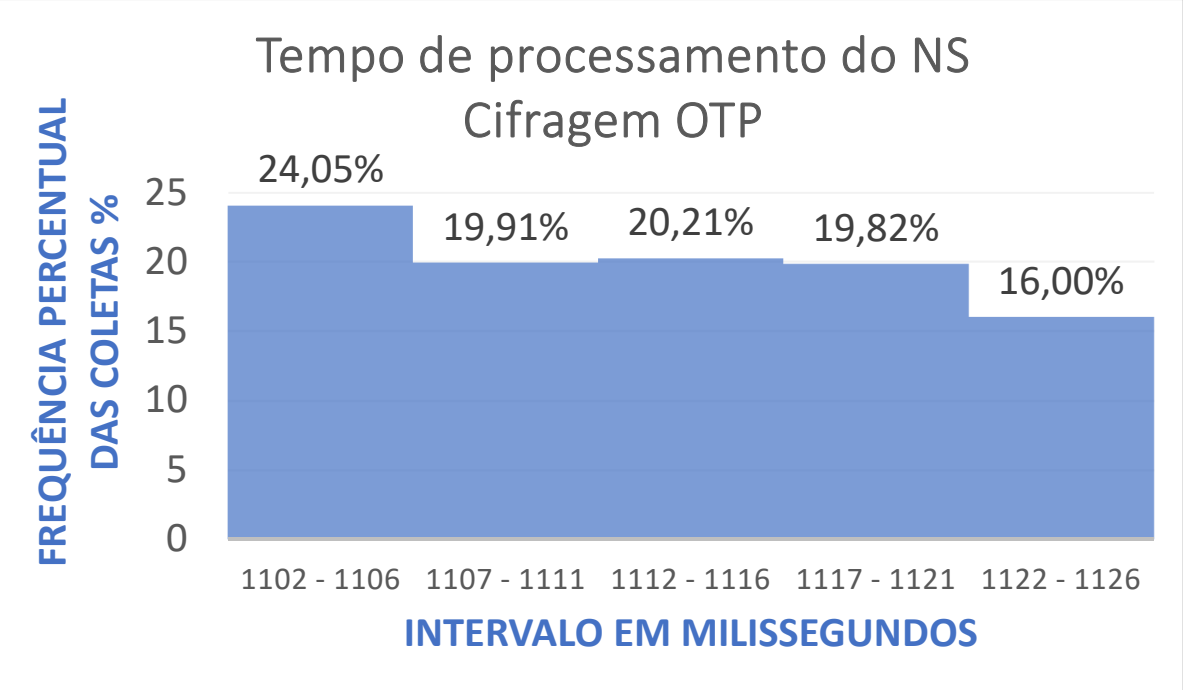
```
Radiuino! Sniffer
Temperatura criptografada: -4138.38
Temperatura criptografada: 398.00
Temperatura criptografada: -4138.38
Temperatura criptografada: 7971.40
Temperatura criptografada: -4138.38
Temperatura criptografada: 7038.67
```

Fonte: elaboração própria

Na Figura 37 observa-se também que o *sniffer*, capturou os pacotes de forma cifrada. As informações coletadas pelo *sniffer* por meio da interceptação estão modificadas e inelegíveis para um elemento invasor na RSSF que não participa do processo de cifragem.

O tempo de processamento do NS foi testado também com a utilização deste método. NA Figura 38 mostrado um gráfico com o tempo de processamento do NS, com utilização da cifragem OTP.

Figura 38 - Tempo de processamento do NS, com a utilização da cifragem OTP



Fonte: elaboração própria

Neste teste, o tempo de processamento mínimo no NS foi de 1102 milissegundos, o tempo de processamento máximo 1186 milissegundos, o tempo médio de processamento foi de 1144,04 milissegundos, o desvio padrão foi de 24,52 milissegundos e o coeficiente de variação foi de 2,14%.

Na Figura 39, é mostrado o NS exibindo uma amostragem do tempo de processamento, chave gerada, pacote recebido de forma cifrada, pacote recebido após a decifragem, pacote em cifragem antes do envio, e nova cifragem realizada antes do envio.

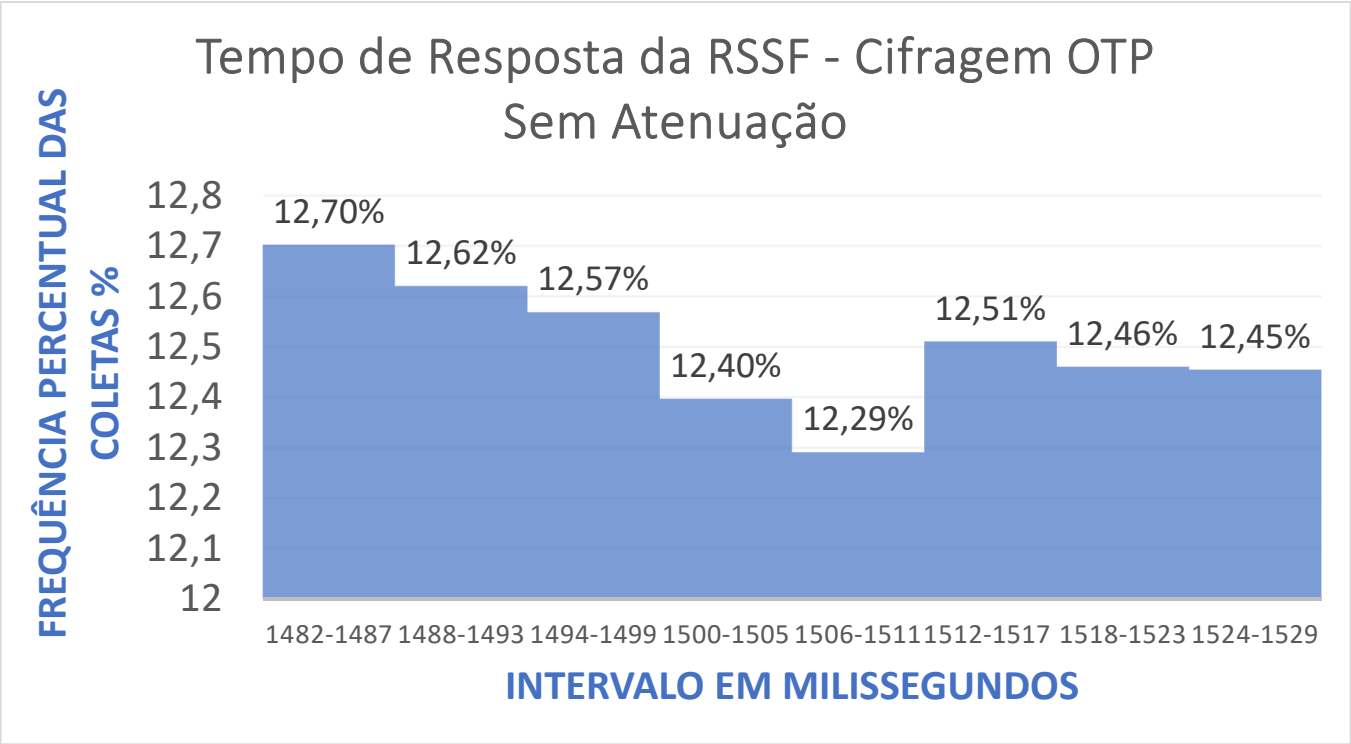
Figura 39 - Tempo de processamento e processos de cifragem exibidos pelo NS

```
Radiuino! Sensor!!
Pacote:
242 97 218 65 177 75 65 185 243 97 218 65 239 156 169 100 60 206 115 143 60 228 012 01 255 36 237 171 117 205 2
-------------
Chave
242 97 218 65 177 75 65 185
-------------
Descriptografado
00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-------------
Pacote:
16 130 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 253 01 00 00 02 01 01 03 00 02 00 01 97 00 01 210 00 00
-------------
Chave
242 97 218 65 177 75 65 185
-------------
Criptografado
226 227 218 65 177 75 65 185 242 97 219 65 015 87 21 203 122 133 164 86 197 34 107 43 247 126 06 199 161 43 22
Tempo:1147
```

Fonte: elaboração própria

Assim como ocorreu nos testes de comunicação sem a utilização de cifragem e utilizando o método XOR, o tempo de resposta da RSSF também foi medido, nos testes utilizando o método de cifragem OTP, sem nenhuma atenuação e com a utilização de atenuação. Na Figura 40, é mostrado o tempo de resposta da RSSF com a cifragem OTP, sem atenuação. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de resposta da RSSF em milissegundos.

Figura 40 - Tempo de resposta da RSSF com OTP e sem atenuação



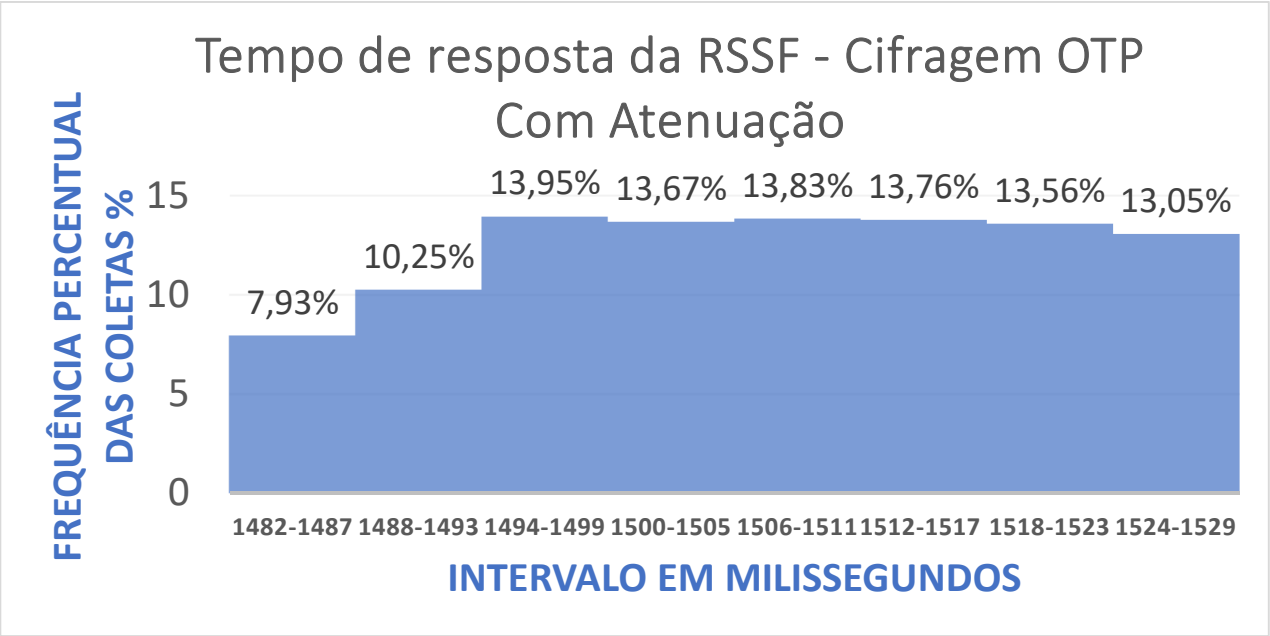
Fonte: elaboração própria

Neste teste, o tempo de resposta da RSSF mínimo foi de 1482 milissegundos, o tempo máximo de resposta da RSSF foi de 1529 milissegundos, o tempo médio de resposta da RSSF foi de 1505,39 milissegundos, o desvio padrão foi de 13,88 milissegundos e o coeficiente de variação foi de 0,92%.

O mesmo teste foi feito com o tempo de resposta da RSSF com a cifragem OTP, utilizando atenuação. Na Figura 41, é mostrado o tempo de resposta da RSSF com a cifragem OTP, com atenuação. Na vertical (eixo y), é apresentada a frequência percentual das coletas e na horizontal (eixo x), é mostrado o tempo de resposta da RSSF em milissegundos.

.

Figura 41 - Tempo de resposta da RSSF com OTP e com atenuação



Fonte: elaboração própria

Neste teste, o tempo de resposta da RSSF mínimo foi de 1482 milissegundos, o tempo máximo de resposta da RSSF foi de 1529 milissegundos, o tempo médio de resposta da RSSF foi de 1507,08 milissegundos, o desvio padrão foi de 13,04 milissegundos e o coeficiente de variação foi de 0,86%.

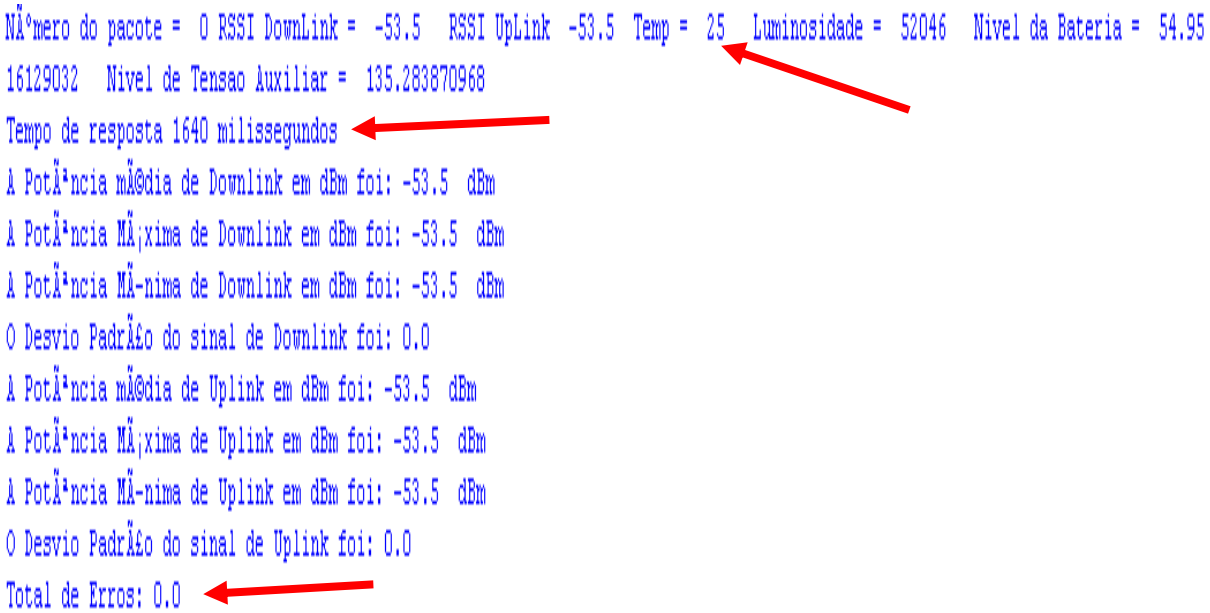
Pode-se perceber que no tempo de processamento da RSSF com a utilização da cifragem OTP e sem a utilização de atenuação, houve uma proporção maior de coletas nas faixas de 1482 até 1487 e uma proporção bem menor na faixa de 1506 até 1511 milissegundos. Já nos testes com a cifragem OTP e com atenuação, pode-se perceber que também houve uma proporção bem menor na faixa de 1482 até 1487 milissegundos, no entanto, as maiores proporções foram percebidas entre as faixas 1494 até 1529 milissegundos, intervalos este onde se encontra a média deste teste, que foi de 1507,08 milissegundos, havendo uma homogeneidade maior no tempo.

Como na média, houve uma tempo maior de resposta da RSSF, nos experimentos nas três situações possíveis no trabalho (sem cifragem, com a

cifragem XOR e com a cifragem OTP, suponha-se que no *Radio Control*, que controla o rádio, que é realizado por um microcontrolador interno ao transceptor CC1101 utilizado no rádio BE-900, de alguma maneira, quando trabalha com um potência mais baixa, faz com o que seu processamento demore mais tempo. É de se supor, que talvez a modulação e demodulação que é necessária na conversão de sinal analógico para digital, seja mais rápida com uma potência mais alta do que com uma potência mais baixa.

O *software* desenvolvido em *Python* no IoT-PM, exibe as informações do pacote recebido com uma série informações sobre o pacote, como a temperatura real, a temperatura cifrada, o tempo de resposta e a taxa de erro de pacote (PER) da RSSF, com a utilização do método OTP. Na Figura 42, é mostrado um exemplo de saída do *software Python*, que foi realizada antes dos testes, na bancada de emulação.

Figura 42 – Exemplo de saída do software no IoT-PM com OTP

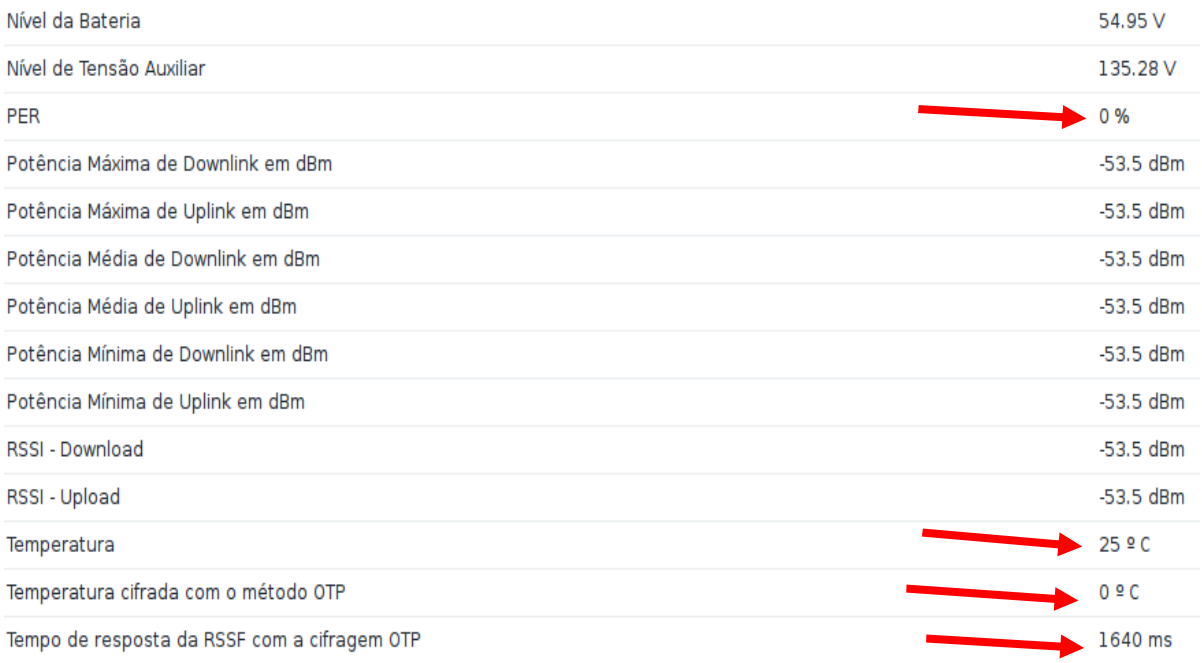


Fonte: elaboração própria

O *software Zabbix* captura as informações coletadas pelo NS e enviadas para o IoT-PM e as exibem. A Figura 43 mostra as informações

coletadas pelo Zabbix que foram feitas antes dos testes, na bancada de emulação.

Figura 43 – Exemplo dos dados exibidos pelo Zabbix com a cifragem OTP



Nível da Bateria	54.95 V
Nível de Tensão Auxiliar	135.28 V
PER	0 %
Potência Máxima de Downlink em dBm	-53.5 dBm
Potência Máxima de Uplink em dBm	-53.5 dBm
Potência Média de Downlink em dBm	-53.5 dBm
Potência Média de Uplink em dBm	-53.5 dBm
Potência Mínima de Downlink em dBm	-53.5 dBm
Potência Mínima de Uplink em dBm	-53.5 dBm
RSSI - Download	-53.5 dBm
RSSI - Upload	-53.5 dBm
Temperatura	25 ºC
Temperatura cifrada com o método OTP	0 ºC
Tempo de resposta da RSSF com a cifragem OTP	1640 ms

Fonte: elaboração própria

Conforme mostrado no Python (Figura 42) e no Zabbix (Figura 43), não houve perda de pacotes (PER) nos testes realizados com a utilização do método de cifragem OTP. Um detalhe é que a temperatura cifrada com o método OTP está com valor 0 (zero) pois não foi possível ainda enviá-la para o Zabbix.

7.4. Comparativos das métricas

Embora as informações estejam descritas nos tópicos anteriores neste capítulo, para melhor compreensão e comparativo entre os dados medidos referente as métricas escolhidas para os testes no trabalho, tabelas serão exibidas para fins de concentração das informações.

Com relação ao tempo de processamento do NS, houve nitidamente, um aumento bastante significativo de tempo com a implementação dos métodos de cifragem, como mostrado na Tabela 7.

Tabela 7- Comparativo do tempo de processamento do NS

TEMPO DE PROCESSAMENTO DO NS	Sem cifragem	Método XOR	Método OTP
Tempo Mínimo	43 ms	739 ms	1102 ms
Tempo Máximo	45 ms	744 ms	1186 ms
Tempo Médio	44 ms	741,5 ms	1126 ms
Desvio Padrão	0,81 ms	1,70 ms	24,52 ms
Coeficiente de Variação	1,86%	0,23%	2,14%

Fonte: elaboração própria

Houve um aumento de tempo médio de 1585,2% do processamento do NS com a utilização do método de cifragem XOR em comparação as medidas realizadas sem cifragem. Já com a utilização do método OTP, o aumento em comparação as medidas realizadas sem cifragem foi de 2459%. Estes aumentos significativos se justificam pelo fato de o NS é o único elemento no experimento, que faz ambas as etapas dos métodos de cifragem, ou seja, fazem a decifragem quando recebe o pacote, realiza a tarefa solicitada e precisa realizar uma nova cifragem antes de enviar novamente os pacotes para a ERB.

Já com relação ao tempo de resposta da RSSF, estes aumentos de tempo foram menos discrepantes, como mostra a Tabela 8

Tabela 8 - Comparativo de Tempo de Resposta da RSSF

TEMPO DE RESPOSTA DA RSSF	Sem Cifragem Sem Atenuação	Sem Cifragem Com Atenuação	Método XOR Sem Atenuação	Método XOR Com Atenuação	Método OTP Sem Atenuação	Método OTP Com Atenuação
Tempo Mínimo	561 ms	561 ms	889 ms	889 ms	1482 ms	1482 ms
Tempo Máximo	578 ms	578 ms	921 ms	921 ms	1529 ms	1529 ms
Tempo Médio	569,51 ms	570,77 ms	904,91 ms	907,16 ms	1505,39 ms	1507,08 ms
Desvio Padrão	5,18 ms	4,52 ms	9,56 ms	8,81 ms	13,88 ms	13,04 ms
Coeficiente de Variação	0,91%	0,79%	1,05%	0,97%	0,92%	0,86%

Fonte: elaboração própria

Com relação ao tempo médio de resposta da RSSF sem a utilização de atenuação, utilizando o método XOR, ouve um aumento de 58,89% com relação à média sem a utilização de cifragem. Já com a utilização do método OTP, ouve um aumento de 164,33% em relação à média sem a utilização de cifragem.

Já com relação ao tempo médio de resposta da RSSF com a utilização de atenuação, utilizando o método XOR, ouve um aumento de 58,93 com relação à média sem a utilização de cifragem. Já com a utilização do método OTP, ouve um aumento de 164,04% em relação à média sem a utilização de cifragem.

Como já dito anteriormente, não houve em nenhum experimento, perda de pacotes na transmissão. Sendo assim, uma tabela com a PER, a quarta métrica medida neste trabalho, e torna desnecessária.

7.5. Escolha do método de cifragem a ser utilizado na RSSF

Como já citado, as RSSF possuem limitações computacionais (PRANATA; ATHAUDA; SKINNER, 2012). Estas limitações impedem que boa parte das tecnologias eficientes relacionadas à segurança sejam implementadas com sucesso neste tipo de rede. Ao desenvolver os dois métodos de cifragem, pensou-se em proporcionar soluções com distintos níveis de complexidade e consequentemente, com diferentes níveis de consumo computacional dos elementos que compõem a RSSF.

Conforme mostrado anteriormente, o método XOR atua alterando os bits originais da informação, causando apenas um embaralhamento destes bits. Este método, não utiliza nenhum algoritmo de criptografia. Por este motivo, se trata de uma solução com um nível de segurança inferior ao método OTP, no entanto, consome uma quantidade de recursos inferior a este método. Embora tenha havido um aumento significativo no tempo de resposta da RSSF e do tempo de processamento do pacote por parte do NS, em relação à transmissão sem nenhuma cifragem, este método proporciona transmissões mais rápidas do que o método OTP.

Já o método OTP utiliza algoritmo criptográfico simétrico em conjunto com uma função hash com chaves aleatórias. A utilização destes recursos, realizando a quebra dos pacotes, como mostrado na Figura 19, fazem com que

este método tenha um nível de complexidade maior do que o XOR. O método OTP, pelo fato de utilizar estes recursos citados, exige um maior consumo de processamento dos elementos da RSSF na cifragem, ocasionando assim, uma maior lentidão na transmissão dos pacotes, em relação ao método XOR.

Sendo assim, é necessário identificar o grau de segurança necessária, em função da aplicação desejada, para optar por utilizar alguma das soluções desenvolvidas, ou ainda, optar por não utilizar nenhuma delas. O nível necessário de segurança da informação está diretamente relacionado a importância que ela tem para a rede ou para a corporação. É importante assim, ponderar o nível de complexidade da cifragem utilizada na segurança da informação com o consumo de processamento e consequentemente, o tempo de transmissão destas informações na rede. Pode-se considerar três níveis de necessidade de segurança para uma RSSF.

- **Nível baixo:** existem alguns tipos de RSSF, nas quais a segurança das informações transmitidas pelos seus elementos não tem relevância alguma, e o vazamento destas não ocasionariam nenhum impacto negativo. Um exemplo seria uma RSSF que coleta a umidade do solo de um campo de futebol, ou ainda uma RSSF que colete a temperatura de uma residência. Estas informações não trazem nenhum atrativo para algum invasor. Desta forma, não existe a necessidade de nenhuma implementação de segurança.

- **Nível médio:** Em muitas RSSF, as informações transmitidas possuem uma certa importância e mantê-las em um bom grau de sigilo seria interessante. Sendo assim, pode-se considerar um exagero a implementação de um método complexo de cifragem em uma rede simples, dado que existe uma exigência maior de processamento da RSSF, com a utilização de algoritmos mais complexos. Um exemplo seria um ambiente acadêmico, onde dados dos alunos ou professores necessitariam de um certo grau de segurança, mas não de algo muito complexo, mantendo o

desempenho e velocidade da RSSF. Para este tipo de situação, o método XOR seria o mais indicado, pois possui estas características.

- **Nível alto:** Já em alguns ambientes, as informações transmitidas por meio da RSSF podem possuir um maior grau de sigilo, necessitando-se assim, de uma implementação mais complexa e bem elaborada. Um exemplo seria uma RSSF que possui NS que medem a temperatura de uma caldeira em uma indústria, ou ainda uma RSSF que é utilizada em um hospital, transmitindo informações vitais de pacientes. Neste tipo de situação, seria negligencia não utilizar nenhuma segurança, ou ainda, utilizar um método mais simples de cifragem, ao mesmo tempo, muitas vezes, o desempenho da RSSF no que diz respeito a velocidade na transmissão dos pacotes não é primordial. Neste tipo de RSSF, o método OTP seria o mais indicado, pois possui uma maior robustez do que o XOR, sendo assim, mais difícil de um possível invasor decifrar as informações.

8. CONSIDERAÇÕES FINAIS

Este trabalho apresentou dois métodos de cifragem para implementar segurança nos pacotes transmitidos por meio das RSSF. Em ambos os métodos, todos os elementos da RSSF participam de alguma forma dos processos, fato este, que descentraliza a responsabilidade da cifragem em apenas um dispositivo, dificultando assim, a ação de algum elemento de origem maliciosa. A descentralização dos processos, também divide a carga de utilização de processamento dos elementos da RSSF. Os resultados obtidos, com base nas métricas escolhidas, mostram que os métodos de segurança não afetam a integridade das informações transmitidas, uma vez que, mesmo com implementação das cifragems nos pacotes, não ocorreram quaisquer alterações nas informações transmitidas na RSSF e estas chegaram de forma íntegra ao destino.

Com relação às outras duas métricas escolhidas, ficou evidente um aumento do tempo de resposta da RSSF, desde a origem da transmissão, até seu retorno. Pode-se notar que embora todos os elementos da RSSF participem do método de cifragem, o NS foi o maior responsável pelo aumento de tempo nos experimentos com ambos os métodos, pois, conforme descrito no trabalho, ele precisa fazer ambas as etapas do processo, ou seja, a decifragem, quando recebe o pacote e uma nova cifragem, antes de enviar o pacote de volta, além disto, possui um baixo poder de processamento. Entretanto, frente ao tempo total de processamento, este tempo não é relevante no âmbito das RSSF, uma vez que geralmente, as coletas de informações são realizadas em intervalos maiores de tempo.

Já os *sniffers* desenvolvidos para o trabalho para testar o funcionamento dos dois métodos de cifragem, mostraram de forma clara, que de fato, utilizando qualquer um dos métodos, os pacotes são transmitidos pela RSSF de forma cifrada e qualquer invasor que porventura conseguir a captura estes pacotes, não terá acesso a informação correta.

Foi observado um aumento no tempo médio de resposta da RSSF com a utilização de atenuação. No entanto, mesmo com a utilização, os códigos das cifragens desenvolvidas, continuaram estáveis e homogêneos, pois, além de não

haver perda de pacotes, o desvio padrão e o coeficiente de variação, em todos os testes, mantiveram um valor baixo.

O termo segurança da informação deve ser contiuamente estudado no âmbito das RSSF, uma vez que a utilização deste tipo de rede cresce a cada dia, fator este que pode atrair cada vez mais elementos maliciosos e ação de invasores.

Como perspectiva futura, propõe-se testes com estes métodos com uma maior número de NS e saltos na RSSF. Propões-se também realizar um comparativo de consumo de energia dos NS com a utilização dos métodos de cifragens desenvolvidos.

9. REFERÊNCIAS BIBLIOGRÁFICAS

ADAM, M. M.; SHEHU, N. M. WIRELESS SENSOR NETWORKS : SECURITY WIRELESS SENSOR NETWORKS : SECURITY ISSUES. n. December, 2017.

ARAUJO, A. et al. Security in cognitive wireless sensor networks. Challenges and open problems. **EURASIP Journal on Wireless Communications and Networking**, v. 2012, n. 1, p. 48, 2012.

Arduino Uno. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>.

ASSIS, D. et al. Segurança em Redes de Sensores sem Fio – Desafios, Tendências e Orientações Security in Wireless Sensor Networks -Challenges, Trends and Guidelines. **Edição Especial**, v. 13, p. 402–411, 2015.

Beaglebone Black. Disponível em: <https://beagleboard.org/black>.

BENTO, J. **Segurança em Redes de Sensores Wireless**. [s.l.] Faculdade de Engenharia da Universidade do Porto, 2009.

CERT.BR. **Cartilha de Segurança para Internet, versão 4.0**. [s.l: s.n.].

CHELLI, K. Security Issues in Wireless Sensor Networks. **Proceedings of the World Congress on Engineering**, v. 1, 2015.

DARGIE, W.; POELLABAUER, C. **Fundamentals of Wireless Sensor Networks. Theory and Practice**. USA: [s.n.].

DENER, M. Security analysis in wireless sensor networks. **International Journal of Distributed Sensor Networks**, v. 2014, 2014.

Diferenças entre Arduino, Raspberry Pi e Beaglebone. 20018.

ELGENAIDI, W. et al. Memory storage administration of security encryption keys for line topology in maritime wireless sensor networks. **Proceedings of the International Conference on Sensing Technology, ICST**, 2016.

FEKI, M. A. et al. G u e s t Edito rs' In t rodu c t io n. **IEEE Computer Society**, v. 35, p. 24–25, 2013.

FERNANDES, N. C. et al. Ataques e Mecanismos de Segurança em Redes Ad Hoc. 2006.

FRANCO, D. P.; CARDOSO, N. M. Exploração de vulnerabilidades em redes de sensores sem fio: uma descrição detalhada e ilustrada dos principais ataques. **Amazônia em Foco, Castanhal, v.3, n.1, p. , jan/jun,** p. 34–47, 2014.

FUNDATION, R. P. **Raspberry Pi 3**. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.

GANESHAN, R. K. Realizing a Forensic cognizant environment for future IoT Research and Development by designing a secure framework for ... Roshan Kolar Ganeshan Topic Realizing a Forensic cognizant environment for future IoT Research and Development Course MSc Computer S. **Faculty of Creative Arts, Technologies & Science Department of Computer Science & Technology**, n. May, 2017.

História do Radiuino - Radiuino. Disponível em: <http://radiuino.cc/sobre-o-radiuino/>. Acesso em: 8 mar. 2018.

HORST, A. S.; PIRES, A. DOS S.; DÉO, A. L. B. **De A a Zabbix**. São Paulo: [s.n.].

HU, F.; SHARMA, N. K. Security considerations in ad hoc sensor networks. **Ad Hoc Networks**, v. 3, n. 1, p. 69–89, 2005.

IAN F. AKYILDIZ, M. C. V. **Wireless Sensor Networks**. 1º ed. [s.l: s.n.].

ITU-T. ITU-T Recommendation M.3400: TMN Management Functions. **International Telecommunication Union**, v. 3400, p. 110, 2000.

JAMES F. KUROSE; KEITH W. ROSS. **Redes de Computadores e a Internet: Uma abordagem Top-Down**. 5° Ed. ed. São Paulo: [s.n.].

KAHN, D. A. **The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet**. Nova Yorque, Estados Unidos: [s.n.].

KARL, H.; WILLIG, A. **Protocols and Architectures for Wireless Sensor and Architectures for Wireless Sensor**. [s.l: s.n.].

KHAN, A. et al. Secret key encryption model for Wireless Sensor Networks. **Proceedings of 2017 14th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2017**, n. 1, p. 809–815, 2017.

LATA, S.; GUPTA, S. Security in Wireless Sensor Networks Using Broadcasting. v. 4, n. November, p. 31–34, 2014.

LOUREIRO, A. A. F. et al. **Redes de Sensores Sem Fio**. XXI Simpósio Brasileiro de Redes de Computadores. **Anais**...2003Disponível em: <http://homepages.dcc.ufmg.br/~loureiro/cm/docs/sbrc03.pdf>

MACHADO, F. N. R. **Segurança da Informação: Princípios e Controle de Ameaças**. 1° ed. [s.l: s.n.].

MAKSIMOVIĆ, M. et al. Raspberry Pi as Internet of Things hardware : Performances and Constraints. **Design Issues**, v. 3, n. JUNE, p. 8, 2014.

MARGI, M. S.; JR, M.; BARRETO, T. C. M. B. C. Segurança em Redes de Sensores Sem Fio. **Simpósio Brasileiro em Segurança da Informação**, p. 149–194, 2009.

MOH'D, A. et al. C-Sec: Energy efficient link layer encryption protocol for Wireless Sensor Networks. **2012 IEEE Consumer Communications and Networking Conference (CCNC)**, p. 219–223, 2012.

MOHAMMAD ILYAS; IMAD MAHGOUB. **Handbook of Sensor Networks : Compact Wireless and Wired Sensing Systems Edited by M OHAMMAD I LYAS AND I MAD M AHGOUB**. [s.l: s.n.].

MORAES, A. F. DE. **Segurança em Redes: Fundamentos**. 1° ed. São Paulo: [s.n.].

MULHOLLAND, T. M. **Governo Federal Universidade de Brasília – UnB**. Brasília: [s.n.].

NUNO, V. et al. **Performance Measurement in Wireless Sensor Networks**. [s.l.] Universidade de Coimbra, 2016.

OLIVEIRA, R. F. DE. **PROPOSTA DE UM PROXY MANAGER PARA INTERNET DAS COISAS**. [s.l.] Pontífica Universidade Católica, 2016a.

PANDA, M. Security in Wireless Sensor Networks using Cryptographic Techniques. **American Journal of Engineering Research (AJER)**, n. 01, p. 50–56, 2014.

PHOHA, S.; LAPORTA, T.; GRIFFIN, C. **Sensor Network**. [s.l: s.n.].

PINHEIRO, D.; MAGALHÃES, N. Implementação De Segurança Em Redes De Sensores Sem

Fio: Uma Descrição Detalhada E Ilustrada Dos Principais Tipos De Criptografia E Protocolos De Segurança. n. 31, p. 59–75, 2014.

PRANATA, I.; ATHAUDA, R. I.; SKINNER, G. Securing and Governing Access in Ad-Hoc Networks of Internet of Things. **Engineering and Applied Science**, n. September 2015, 2012.

Python. Disponível em: <python.org>.

Radioit. Disponível em: <http://www.radioit.com.br/>. Acesso em: 8 mar. 2018.

Radiuino. Disponível em: <www.radiuino.cc>. Acesso em: 7 fev. 2018.

Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. Disponível em: <https://www.raspberrypi.org/>. Acesso em: 8 mar. 2018.

Raspbian. Disponível em: <https://www.raspbian.org/>.

RUFINO, N. M. DE O. **Segurança em Redes Sem Fio**. 4° ed. Brasil: [s.n.].

RUSSELL, B.; DUREN, D. VAN. **Practical Internet of Things Security**. Birmingham: [s.n.].

SAIF UR, R.; CUI, G.; BAO, J. Post deployment encryption key generation for a fully connected and secure wireless sensor network. **Proceedings - 2014 World Ubiquitous Science Congress: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014**, p. 453–458, 2014.

Sandisk. Disponível em: <https://www.sandisk.com.br/home/memory-cards/microsd-cards/ultra-microsd>.

SANTOS, B. et al. **Internet das Coisas: da Teoria à Prática.**, 2016. (Nota técnica).

SCHNEIER, B. **Applied Cryptography: Protocols, algorithms and Source Code in C**. 2° Edição ed. [s.l: s.n.].

SEEDSTUDIO. **UartSBee V4 - Bees - Seeed Studio**. Disponível em: <https://www.seeedstudio.com/UartSBee-V4-p-688.html>. Acesso em: 16 mar. 2018.

SEN, J. A Survey on Wireless Sensor Network Security. **Computer Networks**, v. 1, n. 2, p. 55–78, 2009.

SLATER, R. **Telegraphic code, to ensure secrecy in the transmission of telegrams**. 3° ed. Toronto, Canadá: [s.n.].

STALLINGS, W. **Redes e Sistemas de Comunicação de Dados**. 5° Edição ed. Rio de Janeiro: [s.n.].

STALLINGS, W. **Criptografia e Segurança de Redes**. 4° ed. São Paulo: Prentice Hall, 2008.

TABRIZI, S. S.; IBRAHIM, D. Security of the Internet of Things. **Proceedings of the 2016 International Conference on Communication and Information Systems - ICCIS '16**, p. 146–150, 2016.

TANENBAUM, A. S. **Redes de Computadores**. 5º ed. Amsterdã: [s.n.].

TANG, S.; LIU, F. A one-time pad encryption algorithm based on one- way hash and conventional block cipher. **Network Security**, v. 2, n. Affiliation 1, p. 72–74, 2012.

THIRUPPATHY KESAVAN, V.; RADHAKRISHNAN, S. Secret Key Cryptography based Security Approach for Wireless Sensor Networks. **2012 International Conference on Recent Advances in Computing and Software Systems**, p. 185–191, 2012.

TRANSCEIVER, L. S.-G. R. F.; CC, E. Cc1101 Cc1101. p. 1100–1102, 2010.

VIEIRA, M. A. M. et al. **Survey on wireless sensor network devices**. IEEE Conference on Emerging Technologies and Factory Automation, 2003

WHITE, C. M. **Redes de Computadores e Comunicação de Dados**. 6° Edição ed. São Paulo: [s.n.].

WILLIAM STALLINGS. **SNMP, SNMPv2, SNMPv3, and RMON 1 and 2**. 3° ed. Boston: [s.n.].

YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. **Computer Networks**, v. 52, n. 12, p. 2292–2330, 2008.

ZIA, T.; ZOMAYA, A. A security framework for wireless sensor networks. **the proceedings of IEEE Sensor Applications Symposium (SAS06)**, n. February, p. 7–9, 2006.

10. Apêndices

APÊNDICE A - *Software* Python – IoT Proxy Manager com a cifragem Xor

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# PROGRAMA PARA APLICAÇÃO DO AP1
import serial
import math
import time
import struct
import array
#import pyotp # funções de geração de chave e criptografia
from time import localtime, strftime
from pyzabbix import ZabbixSender,ZabbixMetric

# Configura a serial
############
# Linux #
############
#n_serial = "/dev/ttyUSB0"
#ser = serial.Serial(n_serial, 9600,
timeout=0.5,parity=serial.PARITY_NONE) # seta
valores da serial
############
# Windows #
############
n_serial = raw_input("Digite o número da serial = ") #seta a serial
ser = serial.Serial("com"+n_serial, 9600,
timeout=1.5,parity=serial.PARITY_NONE) #
seta valores da serial

# Identificação da base
ID_base = raw_input('ID_base = ')

#Identificação do sensor
ID_sensor = raw_input('ID_sensor = ')

# Cria o vetor Pacote
Pacote = {}

# Cria o vetor Pacote
PacoteTX = {}
PacoteRX = {}

# Intervalo entre as medições
#TEMPO1 = 5

#Cria o vetor para salvar os valores das potências
listaPotDesviod ={}
listaPotDesviou ={}


# Cria Pacote de 52 bytes com valor zero em todas as posições
for i in range(0,52): # faz um array com 52 bytes
Pacote[i] = 0
PacoteTX[i] = 0
PacoteRX[i] = 0
opcao='n'
```

```
53
54 ####LOOP INFINITO
55
56 while True:
57 try:
58 #Inicialização das variaveis
59
60 contador_tot = 0
61 contador_pot = 0
62 potmediad = 0.0
63 potacumulad = 0.0
64 potmeddbd = 0.0
65 contador_err = 0
66 potmediau = 0.0
67 potacumulau = 0.0
68 potmeddbu = 0.0
69 PER = 0
70 count = 0
71
72 AcumDPd = 0
73 AcumDPu = 0
74 AcumVad = 0
75 AcumVau = 0
76 MedDPd = 0
77 MedDPu = 0
78 DPd = 0
79 DPu = 0
80 rssid = 0
81 rssiu = 0
82
83
84 nb0 = 0
85 nb1 = 0
86 nta0 = 0
87 nta1 = 0
88
89 PotMaxd = -200
90 PotMind = 10
91
92 PotMaxu = -200
93 PotMinu = 10
94
95
96 # Imprime na tela o menu de opções
97 print 'Escolha um comandos abaixos e depois enter'
98 print '1 - Realiza medidas:'
99 print 's - Para sair:'
100
101 Opcao = raw_input('Entre com a Opção = ')
102
103 # Limpa o buffer da serial
104 ser.flushInput()
105
106 # Coloca no pacote o ID_sensor e ID_base
107 for i in range(52):
108 PacoteTX[i]=0
109
110 PacoteTX[8] = int(ID_sensor)
111 PacoteTX[10] = int(ID_base)
112
```

```
113 # Leitura de temperatura e luminosidade
114 if Opcao == "1":
115 num_medidas = raw_input('Entre com o número de medidas = ')
116 w = int(num_medidas)
117
118 for j in range(0,w): #Inicializa uma lista para gravar as
potências e
calcular o desvio padrão
119 listaPotDesviod[j] = 0
120 listaPotDesviou[j] = 0
121 contador_tot = contador_tot + 1
122
123 filename1 = strftime("Sensor_%Y_%m_%d_%H-%M-%S.txt")
124 print "Arquivo de log: %s" % filename1
125 S = open(filename1, 'w')
126 senha='pontificauniversidade'
127 for j in range(0,w):
128 #pega os milisegundos iniciais para posterior comparação
129 temporespostai = int(round(time.time() * 1000))
130 TEMP=''
131 #criptografar
132 j=0;
133 for k in range(0,52): # transmite pacote
134 if (j == len(senha)):
135 j=0
136 TXbyte = chr(PacoteTX[k] ^ ord(senha[j]))
137 ser.write(TXbyte)
138 j+=1
139
140 # Aguarda a resposta do sensor
141 time.sleep(0.5)
142
143 line = ser.read(52) # faz a leitura de 52 bytes do buffer que
recebe
da serial pela COM
144 if len(line) == 52:
145
146 rssid = ord(line[0]) # RSSI_DownLink
147 rssiu = ord(line[2]) # RSSI_UpLink
148
149 #RSSI Downlink
150 if rssid > 128:
151 RSSId=((rssid-256)/2.0)-74
152
153 else:
154 RSSId=(rssid/2.0)-74
155
156 #RSSI Uplink
157 if rssiu > 128:
158 RSSIu=((rssiu-256)/2.0)-74
159
160 else:
161 RSSIu=(rssiu/2.0)-74
162
163 count = ord(line[12]) # contador de pacotes enviados pelo
sensor
164
165 # Leitura do AD0
166 ad0t = ord(line[16]) # tipo de sensor - no caso está medindo
temperatura
```

```
167 ad0h = ord(line[17]) # alto
168 ad0l = ord(line[18]) # baixo
169 AD0 = ad0h * 256 + ad0l
170 #Vout = 0.003223 * AD0
171 #TEMP = (250/10)#(Vout*100)-50
172
173 # Leitura do AD1
174 ad1t = ord(line[19]) # tipo de sensor - no caso está medindo LDR
175 ad1h = ord(line[20]) # alto
176 ad1l = ord(line[21]) # baixo
177 AD1 = ad1h * 256 + ad1l
178
179 #Leitura do Nível da Bateria
180 nb0 = ord(line[23]) # alto
181 nb1 = ord(line[24]) # baixo
182 NB = nb0 * 256 + nb1
183 NB = float ((NB * 3.3)) / 1023
184
185 #Leitura do Nível de Tensão Auxiliar
186 nta0 = ord(line[26]) # alto
187 nta1 = ord(line[27]) # baixo
188 NTA = nta0 * 256 + nta1
189 NTA = float ((NTA * 3.3)) / 1023
190
191 if RSSId > PotMaxd:
192 PotMaxd = RSSId
193
194 if RSSId < PotMind:
195 PotMind = RSSId
196
197 if RSSIu > PotMaxu:
198 PotMaxu = RSSIu
199
200 if RSSIu < PotMinu:
201 PotMinu = RSSIu
202
203
204 listaPotDesviod[contador_pot]= RSSId #Grava a potência de
downlink para cálculo do desvio padrão
205 listaPotDesviou[contador_pot]= RSSIu #Grava a potência de
uplink para cálculo do desvio padrão
206
207 contador_pot=contador_pot+1 #incrementa o contador utilizado
para a média de potência e para o desvio padrão
208
209 potmwd = pow(10,(RSSId/10)) #converte a potência de downlink
em dBm para mW.
210 potacumulad = potacumulad + potmwd #Soma a potência em mW em um
acumulador
211
212 potmwu = pow(10,(RSSIu/10)) #converte a potência de uplink em
dBm para mW
213 potacumulau= potacumulau + potmwu
214
215
216 # Milisegundos final para calcular
217 temporespostaf = int(round(time.time() * 1000))
218 #print '!!!!!Pacote recebido!!!!!'
219 print 'Tempo de resposta',
220 print temporespostaf - temporespostai,
```

```
221 print 'milissegundos'
222 B= temporespostaf - temporespostai
223
224 print'Número do pacote = ',count, 'RSSI DownLink = ', RSSId, '
RSSI UpLink ', RSSIu, ' Temp = ', AD0, ' Luminosidade = ',AD1,
' Nivel da Bateria = ',NB, ' Nivel de Tensao Auxiliar = ',NTA,
' Tempo de Resposta da RSSF = ',B
225 print >>S,time.asctime(),' Número do pacote = ',count, 'RSSI
DownLink = ', RSSId, ' RSSI UpLink ', RSSIu, ' Temp = ', AD0,
' Luminosidade = ',AD1, ' Nivel da Bateria = ',NB, ' Nivel de
Tensao Auxiliar = ',NTA, ' Tempo de Resposta da RSSF = ',B
226
227 # Define as chaves dos itens no Zabbix
228 L1=["temperatura", "luminosidade", "RSSId", "RSSIu", "NB",
"NTA", "dht_temp_xor", "tempo_resp_xor"]
229
230 # Define as variáveis do Script que alimentarão os itens
231
232 #B= temporespostaf - temporespostai
233 C = PacoteTX[18]
234 L2=[AD0,AD1,RSSId,RSSIu,NB,NTA,C,B]
235
236 T = AD0
237
238 print 'Temperatura Real: ', T
239 print 'Temperatura Cifrada: ', C
240
241 # Faz um loop para enviar cada valor de métrica para cada item
no Zabbix
242 x = 0
243 while x<len(L1):
244 # Envia Temperatura para o Zabbix
245 metrics = []
246 m = ZabbixMetric('DK 101', L1[x], L2[x])
247 metrics.append(m)
248 zbx = ZabbixSender(zabbix_server='192.168.0.198',
zabbix_port=10051, use_config=None)
249 zbx.send(metrics)
250 x+=1
251
252 for l in range(0,contador_pot):
253 AcumVad =AcumVad+ listaPotDesviod[l] #acumula o valor da lista
para calcular a média
254 AcumVau =AcumVau+ listaPotDesviou[l] #acumula o valor da lista
para calcular a média
255
256 MedDPd = float (AcumVad)/float(contador_pot)
257 MedDPu = float (AcumVau)/float(contador_pot)
258
259 for m in range(0,contador_pot):
260 AcumDPd =AcumDPd+ pow((listaPotDesviod[m]- MedDPd),2) #acumula o
valor da variancia
261 AcumDPu =AcumDPu+ pow((listaPotDesviou[m]- MedDPu),2) #acumula o
valor da variancia
262
263 DPd = float (AcumDPd)/float(contador_pot) #termina o calculo da
variancia
264 DPu = float (AcumDPu)/float(contador_pot) #termina o calculo da
variancia
265
```

```
266 potmediad = potacumulad /contador_pot
267 potmeddbd = 10*math.log10(potmediad)
268 #print ' A Potência média de downlink foi:', potmediad , ' mW'
269 print 'A Potência média de Downlink em dBm foi:', potmeddbd,' dBm'
270 print 'A Potência Máxima de Downlink em dBm foi:', PotMaxd,' dBm'
271 print 'A Potência Mínima de Downlink em dBm foi:', PotMind,' dBm'
272 print 'O Desvio Padrão do sinal de Downlink foi:', DPd
273
274 print >>S,time.asctime(),' A Potência média de Downlink em dBm
foi:',
potmeddbd,' dBm'
275 print >>S,time.asctime(),'A Potência Máxima de Downlink em dBm
foi:',
PotMaxd,' dBm'
276 print >>S,time.asctime(),'A Potência Mínima de Downlink em dBm
foi:',
PotMind,' dBm'
277 print >>S,time.asctime(),'O Desvio Padrão do sinal de Downlink
foi:', DPd
278
279
280 potmediau = potacumulau /contador_pot
281 potmeddbu = 10*math.log10(potmediau)
282 #print ' A Potência média de Uplink foi:', potmediau , ' mW'
283 print 'A Potência média de Uplink em dBm foi:', potmeddbu,' dBm'
284 print 'A Potência Máxima de Uplink em dBm foi:', PotMaxu,' dBm'
285 print 'A Potência Mínima de Uplink em dBm foi:', PotMinu,' dBm'
286 print 'O Desvio Padrão do sinal de Uplink foi:', DPu
287
288
289 print >>S,time.asctime(),' A Potência média de Uplink em dBm
foi:',
potmeddbu,' dBm'
290 print >>S,time.asctime(),'A Potência Máxima de Uplink em dBm
foi:',
PotMaxu,' dBm'
291 print >>S,time.asctime(),'A Potência Mínima de Uplink em dBm
foi:',
PotMinu,' dBm'
292 print >>S,time.asctime(),'O Desvio Padrão do sinal de Uplink
foi:', DPu
293
294
295
296
297 PER = (float(contador_err)/float(contador_tot))* 100
298 print 'Total de Erros:', (float(contador_err))
299 print 'Contador Total:', (float(contador_tot))
300 PER = float(PER)
301 print 'A PER foi de:', PER,'%'
302 print >>S,time.asctime(),'A PER foi de:', PER,'%'
303
304
305 #Envia Estatistica Finais ao Zabbix
306 # Define as chaves dos itens no Zabbix
307 L3=["potmeddbd", "PotMaxd", "PotMind", "DPd", "potmeddbu",
"PotMaxu",
"PotMinu", "DPu", "PER"]
308
309 # Define as variáveis do Script que alimentarão os itens
```

```
310
L4=[potmeddbd,PotMaxd,PotMind,DPd,potmeddbu,PotMaxu,PotMinu,DPu,PER]
311
312 # Faz um loop para enviar cada valor de métrica para cada item no
Zabbix
313 x = 0
314 while x<len(L3):
315 # Envia Temperatura para o Zabbix
316 metrics = []
317 m = ZabbixMetric('DK 101', L3[x], L4[x])
318 metrics.append(m)
319 zbx = ZabbixSender(zabbix_server='192.168.0.198',
zabbix_port=10051,
use_config=None)
320 zbx.send(metrics)
321 x+=1
322
323
324
325 S.close()
326 elif Opcao == "s" or Opcao == "S":# caso o caracter digitado for s
327 ser.close() # fecha a porta COM
328 print 'Fim da Execução' # escreve na tela
329 break
330 ser.flushInput()
331
332 else:
333 contador_err = contador_err + 1
334 print ' erro'
335 print >>S,time.asctime(),' erro'
336 ser.flushInput()
337 time.sleep(0.5)
338
339 #contador_tot = contador_tot + 1
340 # Tempo que vai ficar esperando para busca novamente do valor do
ADC
341 #time.sleep(5)
342
343
344
345 # Caso seja dado Ctrl C também sai do loop
346 except KeyboardInterrupt:
347 S.close()
348 ser.close()
349 break
350
```

Apêndice B – Linhas inseridas no firmware do Nó Sensor para fazer a decifragem com o XOR

========##########=======================##########==================

```
inline int PHY::receive(packet * pkt){

  byte rxBytes, rxBytesVerify, rssi, lqi, current_state;
  char senha[]="pontificauniversidade";
  int i,j;
  /*Guardar o tempo de resposta inicial*/
  tempo = millis();
  /* Liga o LED Verde */
  digitalWrite (IO2_PIN, HIGH);

  /* Garante que o pacote terminou de ser recebido - deve ler duas vezes a mesma quantidade de bytes */
  cc1101.Read(CC1101_RXBYTES, &rxBytesVerify);
  do
  {
    rxBytes = rxBytesVerify;
    cc1101.Read(CC1101_RXBYTES, &rxBytesVerify);
  } while (rxBytes != rxBytesVerify);

  /* Checagem de sanidade para ver se o buffer nao esta vazio */
  if (rxBytes == 0)
  {
    cc1101.Strobe(CC1101_SRX);
    /* Desliga o LED Verde */
    digitalWrite (IO2_PIN, LOW);
    return ERR;
  }

  /* Le uma rajada de dados do FIFO de RX */
  cc1101.ReadBurst(CC1101_RXFIFO, (byte *)pkt, sizeof(packet));

  /* Le o byte de RSSI */
  cc1101.Read(CC1101_RXFIFO, &rssi);

  /* Le o byte de LQI */
  cc1101.Read(CC1101_RXFIFO, &lqi);

   Serial.println("Pacote recebido");
  showArray((byte*)pkt,52);
  /*Faz um XOR em todos os bytes do pacote recebido*/
```

==========############==================##########===========

Apêndice C - Linhas inseridas no firmware do Nó Sensor para fazer a cifragem com o XOR

==========#####==========#######==========

```
inline void PHY::send(packet * pkt)
{
  byte current_state;
  byte *txData = (byte *)pkt;
  char senha[]="pontificauniversidade";
  int i,j;

  /* Liga o LED Vermelho */
  if(TxLedBlink == 1)
  {
    digitalWrite (IO0_PIN, HIGH);
  }

  if(set_radio == 1){

    // pino PC1 ligado ao LNA - rx
    digitalWrite (15, LOW);  // pino PC1 ligado ao LNA - rx

    // pino PC0 ligado ao PA - tx
    digitalWrite (14, HIGH); // pino PC0 ligado ao PA - tx
    delay(1);

  }


    /*Faz um XOR em todos os bytes do pacote antes de enviar*/
 Serial.println("Enviando pacote - Antes XOR");
 showArray(txData,52);
 //Serial.println(sizeof(senha));
 j= 0;
 for ( i = 0 ; i < sizeof(packet) ; i++){
   if (j == (sizeof(senha)-1)) j=0;
   txData[i] ^=  senha[j];
   j++;
 }
 Serial.println("Enviando pacote - Depois XOR");
 showArray(txData,52);
```

==========#####==========#######==========

Apêndice D - Linhas inseridas no firmware da Estação Rádio Base para fazer a decifragem com o XOR

========##########=======================##########==================

```
inline int PHY::receive(packet * pkt){

  byte rxBytes, rxBytesVerify, rssi, lqi, current_state;
  char senha[]="pontificauniversidade";
  int i,j;
  /*Guardar o tempo de resposta inicial*/
  tempo = millis();
  /* Liga o LED Verde */
  digitalWrite (IO2_PIN, HIGH);

  /* Garante que o pacote terminou de ser recebido - deve ler duas vezes a mesma quantidade de bytes */
  cc1101.Read(CC1101_RXBYTES, &rxBytesVerify);
  do
  {
    rxBytes = rxBytesVerify;
    cc1101.Read(CC1101_RXBYTES, &rxBytesVerify);
  } while (rxBytes != rxBytesVerify);

  /* Checagem de sanidade para ver se o buffer nao esta vazio */
  if (rxBytes == 0)
  {
    cc1101.Strobe(CC1101_SRX);
    /* Desliga o LED Verde */
    digitalWrite (IO2_PIN, LOW);
    return ERR;
  }

  /* Le uma rajada de dados do FIFO de RX */
  cc1101.ReadBurst(CC1101_RXFIFO, (byte *)pkt, sizeof(packet));

  /* Le o byte de RSSI */
  cc1101.Read(CC1101_RXFIFO, &rssi);

  /* Le o byte de LQI */
  cc1101.Read(CC1101_RXFIFO, &lqi);

   Serial.println("Pacote recebido");
  showArray((byte*)pkt,52);
  /*Faz um XOR em todos os bytes do pacote recebido*/
```

==========############=====================##########===========

Apêndice E – Linhas inseridas no firmware do sniffer com o XOR

==========

```
void PHY::sendSerial(packet * pkt)
{
  /* Escreve o pacote inteiro na porta serial */
  Serial.print("Temperatura: ");
  Serial.write((byte *)pkt, sizeof(packet));
  Serial.println("");
  Serial.print("Posicoes: ");
  Serial.write(pkt->AD0[0]);
  Serial.print(pkt->AD0[1]);
  Serial.println(pkt->AD0[2]);


  Serial.print("[");
  Serial.print(pkt->AD0[0], DEC);
  Serial.print("|");
  Serial.print(pkt->AD0[1], DEC);
  Serial.print("|");
  Serial.print(pkt->AD0[2], DEC);
  Serial.println("");
  Serial.print("]");

  UIB uib;
  uib.bytes[0] = pkt->AD0[2];
  uib.bytes[1] = pkt->AD0[1];
  Serial.print(" ==================== ");
  Serial.println(uib.number);*/
}
```

==========

Apêndice F – Software principal do Python pybaseotp.py no IoT Proxy Manager com a cifragem OTP

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  # PROGRAMA PARA APLICAÇÃO DO AP1
5  import serial
6  import math
7  import time
8  import struct
9  import array
10 import pyotp # funções de geração de chave e criptografia
11 from time import localtime, strftime
12 from pyzabbix import ZabbixSender,ZabbixMetric
13
14 # Configura a serial
15 ############
16 # Linux #
17 ############
18 #n_serial = "/dev/ttyUSB0"
```

```
19 #ser = serial.Serial(n_serial, 9600,
timeout=0.5,parity=serial.PARITY_NONE) # seta
valores da serial
20 ############
21 # Windows #
22 ############
23 n_serial = raw_input("Digite o número da serial = ") #seta a serial
24 ser = serial.Serial("com"+n_serial, 9600,
timeout=1.5,parity=serial.PARITY_NONE) #
seta valores da serial
25
26 # Identificação da base
27 ID_base = raw_input('ID_base = ')
28
29 #Identificação do sensor
30 ID_sensor = raw_input('ID_sensor = ')
31
32 # Cria o vetor Pacote
33 Pacote = {}
34
35 # Cria o vetor Pacote
36 PacoteTX = {}
37 PacoteRX = {}
38
39 # Intervalo entre as medições
40 TEMPO1 = 5
41
42 #Cria o vetor para salvar os valores das potências
43 listaPotDesviod ={}
44 listaPotDesviou ={}
45
46
47 # Cria Pacote de 52 bytes com valor zero em todas as posições
48 for i in range(0,52): # faz um array com 52 bytes
49 Pacote[i] = 0
50 PacoteTX[i] = 0
51 PacoteRX[i] = 0
52 opcao='n'
53
54 ####LOOP INFINITO
55
56 while True:
57 try:
58 #Inicialização das variaveis
59
60 contador_tot = 0
61 contador_pot = 0
62 potmediad = 0.0
63 potacumulad = 0.0
64 potmeddbd = 0.0
65 contador_err = 0
66 potmediau = 0.0
67 potacumulau = 0.0
68 potmeddbu = 0.0
69 PER = 0
70 count = 0
71
72 AcumDPd = 0
73 AcumDPu = 0
74 AcumVad = 0
```

```
75 AcumVau = 0
76 MedDPd = 0
77 MedDPu = 0
78 DPd = 0
79 DPu = 0
80 rssid = 0
81 rssiu = 0
82
83
84 nb0 = 0
85 nb1 = 0
86 nta0 = 0
87 nta1 = 0
88
89 PotMaxd = -200
90 PotMind = 10
91
92 PotMaxu = -200
93 PotMinu = 10
94
95
96 # Imprime na tela o menu de opções
97 print 'Escolha um comandos abaixos e depois enter'
98 print '1 - Realiza medidas:'
99 print 's - Para sair:'
100
101 Opcao = raw_input('Entre com a Opção = ')
102
103 # Limpa o buffer da serial
104 ser.flushInput()
105
106 # Coloca no pacote o ID_sensor e ID_base
107 for i in range(52):
108 PacoteTX[i]=0
109
110 PacoteTX[8] = int(ID_sensor)
111 PacoteTX[10] = int(ID_base)
112 #PacoteTX[15] = 255
113 # Leitura de temperatura e luminosidade
114 if Opcao == "1":
115 #ID_sensor = raw_input('ID_sensor = ') # Identificação do sensor a
ser acessado
116 #Pacote[8] = int(ID_sensor) # Coloca no pacote o ID_sensor
117 num_medidas = raw_input('Entre com o número de medidas = ')
118 w = int(num_medidas)
119
120 for j in range(0,w): #Inicializa uma lista para gravar as
potências e
calcular o desvio padrão
121 listaPotDesviod[j] = 0
122 listaPotDesviou[j] = 0
123 contador_tot = contador_tot + 1
124
125 filename1 = strftime("Sensor_%Y_%m_%d_%H-%M-%S.txt")
126 print "Arquivo de log: %s" % filename1
127 S = open(filename1, 'w')
128 senha='pontificauniversidade'
129 for j in range(0,w):
130 #pega os milisegundos iniciais para posterior comparação
131 temporespostai = int(round(time.time() * 1000))
```

```
132 TEMP=''
133 #criptografar
134 senhamd5 = pyotp.gerarChave64(senha)
135 tmppact = array.array('B',[0]*52)
136
137 #print "pacote aberto"
138 for k in range(0,52):
139 tmppact[k] = PacoteTX[k]
140
141
142 pacotescript = pyotp.otp_enc52bytes( tmppact ,senhamd5 )
143
144 for k in range(0,52): # transmite pacote
145 ser.write(pacotescript[k])
146 #C = PacoteTX[18]
147
148 # Aguarda a resposta do sensor
149 time.sleep(0.5)
150
151 line = ser.read(52) # faz a leitura de 52 bytes do buffer que
recebe
da serial pela COM
152 if len(line) == 52:
153
154 rssid = ord(line[0]) # RSSI_DownLink
155 rssiu = ord(line[2]) # RSSI_UpLink
156
157 #RSSI Downlink
158 if rssid > 128:
159 RSSId=((rssid-256)/2.0)-74
160
161 else:
162 RSSId=(rssid/2.0)-74
163
164 #RSSI Uplink
165 if rssiu > 128:
166 RSSIu=((rssiu-256)/2.0)-74
167
168 else:
169 RSSIu=(rssiu/2.0)-74
170
171 count = ord(line[12]) # contador de pacotes enviados pelo
sensor
172
173 # Leitura do AD0
174 ad0t = ord(line[16]) # tipo de sensor - no caso está medindo
temperatura
175 ad0h = ord(line[17]) # alto
176 ad0l = ord(line[18]) # baixo
177 AD0 = ad0h * 256 + ad0l
178 #Vout = 0.003223 * AD0
179 #TEMP = (250/10)#(Vout*100)-50
180
181 # Leitura do AD1
182 ad1t = ord(line[19]) # tipo de sensor - no caso está medindo LDR
183 ad1h = ord(line[20]) # alto
184 ad1l = ord(line[21]) # baixo
185 AD1 = ad1h * 256 + ad1l
186
187 #Leitura do Nível da Bateria
```

```
188 nb0 = ord(line[23]) # alto
189 nb1 = ord(line[24]) # baixo
190 NB = nb0 * 256 + nb1
191 NB = float ((NB * 3.3)) / 1023
192
193 #Leitura do Nível de Tensão Auxiliar
194 nta0 = ord(line[26]) # alto
195 nta1 = ord(line[27]) # baixo
196 NTA = nta0 * 256 + nta1
197 NTA = float ((NTA * 3.3)) / 1023
198
199 if RSSId > PotMaxd:
200 PotMaxd = RSSId
201
202 if RSSId < PotMind:
203 PotMind = RSSId
204
205 if RSSIu > PotMaxu:
206 PotMaxu = RSSIu
207
208 if RSSIu < PotMinu:
209 PotMinu = RSSIu
210
211
212 listaPotDesviod[contador_pot]= RSSId #Grava a potência de
downlink para cálculo do desvio padrão
213 listaPotDesviou[contador_pot]= RSSIu #Grava a potência de
uplink para cálculo do desvio padrão
214
215 contador_pot=contador_pot+1 #incrementa o contador utilizado
para a média de potência e para o desvio padrão
216
217 potmwd = pow(10,(RSSId/10)) #converte a potência de downlink
em dBm para mW.
218 potacumulad = potacumulad + potmwd #Soma a potência em mW em um
acumulador
219
220 potmwu = pow(10,(RSSIu/10)) #converte a potência de uplink em
dBm para mW
221 potacumulau= potacumulau + potmwu
222 temporespostaf = int(round(time.time() * 1000))
223 B = temporespostaf - temporespostai
224 print'Número do pacote = ',count, 'RSSI DownLink = ', RSSId, '
RSSI UpLink ', RSSIu, ' Temp = ', AD0, ' Luminosidade = ',AD1,
' Nivel da Bateria = ',NB, ' Nivel de Tensao Auxiliar = ',NTA,
' Tempo de Resposta da RSSF = ',B
225 print >>S,time.asctime(),' Número do pacote = ',count, 'RSSI
DownLink = ', RSSId, ' RSSI UpLink ', RSSIu, ' Temp = ', TEMP,
' Luminosidade = ',AD1, ' Nivel da Bateria = ',NB, ' Nivel de
Tensao Auxiliar = ',NTA, ' Tempo de Resposta da RSSF = ',B
226
227 # Milisegundos final para calcular
228 #temporespostaf = int(round(time.time() * 1000))
229 #print '!!!!!Pacote recebido!!!!!'
230 print 'Tempo de resposta',
231 print temporespostaf - temporespostai,
232 print 'milissegundos'
233
234 # Define as chaves dos itens no Zabbix
235 L1=["temperatura", "luminosidade", "RSSId", "RSSIu", "NB",
```

```
"NTA", "tempo_resp_otp", "dht_temp_otp"]
236
237 # Define as variáveis do Script que alimentarão os itens
238 A= (250/10)#TEMP
239 #B= temporespostaf - temporespostai
240 C = PacoteTX[18]
241 L2=[AD0,AD1,RSSId,RSSIu,NB,NTA,B,C]
242
243 # Faz um loop para enviar cada valor de métrica para cada item
no Zabbix
244 x = 0
245 while x<len(L1):
246 # Envia Temperatura para o Zabbix
247 metrics = []
248 m = ZabbixMetric('DK 101', L1[x], L2[x])
249 metrics.append(m)
250 zbx = ZabbixSender(zabbix_server='192.168.0.198',
zabbix_port=10051, use_config=None)
251 zbx.send(metrics)
252 x+=1
253
254 for l in range(0,contador_pot):
255 AcumVad =AcumVad+ listaPotDesviod[l] #acumula o valor da lista
para calcular a média
256 AcumVau =AcumVau+ listaPotDesviou[l] #acumula o valor da lista
para calcular a média
257
258 MedDPd = float (AcumVad)/float(contador_pot)
259 MedDPu = float (AcumVau)/float(contador_pot)
260
261 for m in range(0,contador_pot):
262 AcumDPd =AcumDPd+ pow((listaPotDesviod[m]- MedDPd),2) #acumula o
valor da variancia
263 AcumDPu =AcumDPu+ pow((listaPotDesviou[m]- MedDPu),2) #acumula o
valor da variancia
264
265 DPd = float (AcumDPd)/float(contador_pot) #termina o calculo da
variancia
266 DPu = float (AcumDPu)/float(contador_pot) #termina o calculo da
variancia
267
268 potmediad = potacumulad /contador_pot
269 potmeddbd = 10*math.log10(potmediad)
270 #print ' A Potência média de downlink foi:', potmediad , ' mW'
271 print 'A Potência média de Downlink em dBm foi:', potmeddbd,' dBm'
272 print 'A Potência Máxima de Downlink em dBm foi:', PotMaxd,' dBm'
273 print 'A Potência Mínima de Downlink em dBm foi:', PotMind,' dBm'
274 print 'O Desvio Padrão do sinal de Downlink foi:', DPd
275
276 print >>S,time.asctime(),' A Potência média de Downlink em dBm
foi:',
potmeddbd,' dBm'
277 print >>S,time.asctime(),'A Potência Máxima de Downlink em dBm
foi:',
PotMaxd,' dBm'
278 print >>S,time.asctime(),'A Potência Mínima de Downlink em dBm
foi:',
PotMind,' dBm'
279 print >>S,time.asctime(),'O Desvio Padrão do sinal de Downlink
foi:', DPd
```

```
280
281
282 potmediau = potacumulau /contador_pot
283 potmeddbu = 10*math.log10(potmediau)
284 #print ' A Potência média de Uplink foi:', potmediau , ' mW'
285 print 'A Potência média de Uplink em dBm foi:', potmeddbu,' dBm'
286 print 'A Potência Máxima de Uplink em dBm foi:', PotMaxu,' dBm'
287 print 'A Potência Mínima de Uplink em dBm foi:', PotMinu,' dBm'
288 print 'O Desvio Padrão do sinal de Uplink foi:', DPu
289
290
291 print >>S,time.asctime(),' A Potência média de Uplink em dBm
foi:',
potmeddbu,' dBm'
292 print >>S,time.asctime(),'A Potência Máxima de Uplink em dBm
foi:',
PotMaxu,' dBm'
293 print >>S,time.asctime(),'A Potência Mínima de Uplink em dBm
foi:',
PotMinu,' dBm'
294 print >>S,time.asctime(),'O Desvio Padrão do sinal de Uplink
foi:', DPu
295
296
297
298
299 PER = (float(contador_err)/float(contador_tot))* 100
300 print 'Total de Erros:', (float(contador_err))
301 print 'Contador Total:', (float(contador_tot))
302 PER = float(PER)
303 print 'A PER foi de:', PER,'%'
304 print >>S,time.asctime(),'A PER foi de:', PER,'%'
305
306
307 #Envia Estatistica Finais ao Zabbix
308 # Define as chaves dos itens no Zabbix
309 L3=["potmeddbd", "PotMaxd", "PotMind", "DPd", "potmeddbu",
"PotMaxu",
"PotMinu", "DPu", "PER"]
310
311 # Define as variáveis do Script que alimentarão os itens
312
L4=[potmeddbd,PotMaxd,PotMind,DPd,potmeddbu,PotMaxu,PotMinu,DPu,PER]
313
314 # Faz um loop para enviar cada valor de métrica para cada item no
Zabbix
315 x = 0
316 while x<len(L3):
317 # Envia Temperatura para o Zabbix
318 metrics = []
319 m = ZabbixMetric('DK 101', L3[x], L4[x])
320 metrics.append(m)
321 zbx = ZabbixSender(zabbix_server='192.168.0.198',
zabbix_port=10051,
use_config=None)
322 zbx.send(metrics)
323 x+=1
324
325
326
```

```
327 S.close()
328 elif Opcao == "s" or Opcao == "S":# caso o caracter digitado for s
329 ser.close() # fecha a porta COM
330 print 'Fim da Execução' # escreve na tela
331 break
332 ser.flushInput()
333
334 else:
335 contador_err = contador_err + 1
336 print ' erro'
337 print >>S,time.asctime(),' erro'
338 ser.flushInput()
339 time.sleep(0.5)
340
341 #contador_tot = contador_tot + 1
342 # Tempo que vai ficar esperando para busca novamente do valor do
ADC
343 #time.sleep(5)
344
345
346
347 # Caso seja dado Ctrl C também sai do loop
348 except KeyboardInterrupt:
349 S.close()
350 ser.close()
351 break
352
```

Apêndice G - Software de funções de cifragem do Python pyotp.py no IoT Proxy Manager com a cifragem OTP

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import pyDes
5 import array
6 import hashlib
7
8
9 def gerarChave64(chavestr):
10 m='' # limpa o valor de m
11 m=hashlib.md5() # instancia a classe
12 m.update( bytearray(chavestr) ) # atribui o valor md5 para m
13 return m.digest()[:8] # retorna a metade do hash (o md5 gera por
padrão 128 bits)
14
15 #inicia a definição da função do One Time Pad
16 def otp_enc52bytes(pacote52bytes,chave64): # define a função otp
17 pacotestr = pacote52bytes.tostring() # converte o pacote para
string
18 crypt = pyDes.des(chave64, pyDes.CBC, "\0\0\0\0\0\0\0\0", pad=None,
padmode=pyDes.PAD_NORMAL) # Cria uma instância do objeto do DES na
memória PAD =
preenche automaticamente
19 blocoi=12 # começa do byte 12
20 blocof=20 # vai de 8 em 8 bytes
```

```
21 bloco64enc='' # Limpa
22 pacote=array.array('B',[0]*12) #cria o array de 12 bytes que não
entrarão no
processo do One Time Pad
23 j= 0
24
25 # Faz o Xor nos primeiros 12 bytes
26 for i in range(12):
27 if (j == len(chave64)):
28 j=0
29 pacote[i] = pacote52bytes[i] ^ ord(chave64[j])
30 j+=1
31 pacote52enc = pacote.tostring() # Atribui no pacote resultante os
primeiros 12
bytes com o Xor
32
33 #inicia a criptografia com o One Time Pad
34
35 while blocof <= 53 :
36 bloco64enc = crypt.encrypt(pacotestr[blocoi:blocof]) # Faz a
criptografia
DES em blocos de 64 bits do pacote (bytes 12 aos 20), (21 ao 28) e
assim por
diante
37 pacote52enc += bloco64enc # Acresecente a variável os 12 bytes com
o Xor + o
primeiro bloco One Time Pad
38 novachave=gerarChave64(bloco64enc) # gera nova chave baseada no
bloco anterior
39 crypt = pyDes.des(novachave, pyDes.CBC, "\0\0\0\0\0\0\0\0",
pad=None,
padmode=pyDes.PAD_NORMAL) # instancia um novo objeto DES com a senha
do
bloco anterior
40 blocoi = blocof # byte inicial se torna o byte final do processo
anterior
41 blocof += 8 #bloco final é acrescenteado 8 bytes
42 return pacote52enc # retorno o pacote criptografado (12 com Xor) e
restante com
One Time PAd
43
44 def otp_des52bytes(pacote52bytes,chave64):
45 pacotestr = pacote52bytes
46 crypt = pyDes.des(chave64, pyDes.CBC, "\0\0\0\0\0\0\0\0", pad=None,
padmode=pyDes.PAD_NORMAL)
47 blocoi=12;
48 blocof=20;
49 bloco64enc='';
50 pacote=array.array('B',[0]*12);
51 j= 0;
52
53
54 # Faz o Xor nos primeiros 12 bytes
55
56 for i in range(12):
57 if (j == len(chave64)):
58 j=0
59 pacote[i] = pacote52bytes[i] ^ ord(chave64[j])
60 j+=1
61 pacote52des = pacote.tostring()
```

```
63 #inicia a descriptografia com o One Time Pad
64
65 while blocof <= 53 :
66 bloco64enc = crypt.decrypt(pacotestr[blocoi:blocof]) #
descriptografa o o
pacote
67 pacote52des += bloco64enc
68 novachave=gerarChave64(pacotestr[blocoi:blocof])
69 crypt = pyDes.des(novachave, pyDes.CBC, "\0\0\0\0\0\0\0\0",
pad=None,
padmode=pyDes.PAD_NORMAL)
70 blocoi = blocof
71 blocof += 8
72 return pacote52des
```

Apêndice H - Linhas inseridas no firmware do Nó Sensor para fazer a decifragem com o OTP

========##########======================##########==================

```
//declaração das funções nesse arquivo de cabeçalho
void showArray(byte output[], int isize);

byte* geraChave64(unsigned char* chave);

byte* otp_enc52bytes(byte* pacote52bytes,byte *chave64);

byte* otp_des52bytes(byte* pacote52bytes,byte *chave64);


#include <MD5.h> // importa a biblioteca MD5
#include <DES.h> // importa a biblioteca DES
#include "Cript.h" // importa as funções de criptografia e geração de chave
#include "./printf.h"

//função mais de depuração para mostrar os bytes. Apenas para fins de teste
void showArray(byte output[], int isize)
{
  for (int i = 0; i < isize; i++)
  {
    if (output[i] < 0x10)
    {
      Serial.print("0");
    }
    Serial.print(output[i], DEC);
    Serial.print(" ");
  }
  Serial.println();
}
```

```
  /*Gera a primeira chave fazendo um md5 na chave escolhida pelo usuário
    retorna apenas metade do md5 8bytes = 64bits que será usado pelo algoritmo DES
  */
  byte* geraChave64(unsigned char* chave){
      unsigned char* hash=MD5::make_hash((char*) chave);

      byte* ret = (byte*) malloc(sizeof(byte)*8);
      for(int i =0; i < 8;i++){
        ret[i] =  hash[i];
      }
      free(hash);

      return ret;
  }


/*
 Rotina que descriptografa o pacote criptografado. OS primeiros 12 bytes
 descriptografa com o XOR no restante do pacotes, descriptografa com o OTP
 Faz a mesma coisa usando o mesmo algoritmo. No entando, ela descriptografa ao invés de cifrar
 */

byte* otp_des52bytes(byte* pacote52bytes,byte *chave64){
    int blocoi=12;
    int blocof=20;
    byte bloco64enc[8];
    byte bloco64[8];
    byte* pacote52enc;
    byte* chave;
    int i,j;
    DES des;
    chave = (byte*) malloc(sizeof(byte)*8);
    pacote52enc = (byte*) malloc(sizeof(byte)*52);

    for(i=0;i<8;i++){
      chave[i]=chave64[i];
    }
```

```
j=0;
for(i=0; i < 12;i++){
  if( j == strlen((char*)chave)-1  ) j=0;
  pacote52enc[i] = pacote52bytes[i] ^ chave[j];
  j++;
}


while( blocof <= 52 ){
    j=0;
    for(i = blocoi ; i < blocof ; i++ ){

      bloco64[j] =  pacote52bytes[i];
      j++;
    }

    des.decrypt(bloco64enc, bloco64, chave);

    j=0;
    for(i = blocoi ; i < blocof ; i++ )
    {
      pacote52enc[i] = bloco64enc[j];
      j++;
    }


        free(chave);
        chave=geraChave64( (byte*) bloco64 );

        blocoi = blocof;
        blocof += 8;
    }
    free(chave);
    return pacote52enc;
}
```

========##########======================##########==================

Apêndice I - Linhas inseridas no firmware do Nó Sensor para fazer a cifragem com o OTP

========##########======================##########==================

```
//declaração das funções nesse arquivo de cabeçalho
void showArray(byte output[], int isize);

byte* geraChave64(unsigned char* chave);

byte* otp_enc52bytes(byte* pacote52bytes,byte *chave64);

byte* otp_des52bytes(byte* pacote52bytes,byte *chave64);


#include <MD5.h> // importa a biblioteca MD5
#include <DES.h> // importa a biblioteca DES
#include "Cript.h" // importa as funções de criptografia e geração de chave
#include "./printf.h"

//função mais de depuração para mostrar os bytes. Apenas para fins de teste
void showArray(byte output[], int isize)
{
  for (int i = 0; i < isize; i++)
  {
    if (output[i] < 0x10)
    {
      Serial.print("0");
    }
    Serial.print(output[i], DEC);
    Serial.print(" ");
  }
  Serial.println();
}
```

```
/*Gera a primeira chave fazendo um md5 na chave escolhida pelo usuário
  retorna apenas metade do md5 8bytes = 64bits que será usado pelo algoritmo DES
*/
byte* geraChave64(unsigned char* chave){
    unsigned char* hash=MD5::make_hash((char*) chave);

    byte* ret = (byte*) malloc(sizeof(byte)*8);
    for(int i =0; i < 8;i++){
      ret[i] =  hash[i];
    }
    free(hash);

    return ret;
}

/*utilizando o DES e um hash de uma chave incial como base
  faz a criptografia dos dados usando One Time Pad
*/
byte* otp_enc52bytes(byte* pacote52bytes,byte *chave64){  //define a função OTP
    int blocoi=12; // começa do byte 12, já que os 11 primeiros são cifrados com o Xor
    int blocof=20; // pula de 8 em 8 bytes
    byte bloco64enc[8];
    byte bloco64[8];
    byte* pacote52enc;
    byte* chave;
    int i,j;
    DES des;

    //aloca memoria
    chave = (byte*) malloc(sizeof(byte)*8);
    pacote52enc = (byte*) malloc(sizeof(byte)*52);

    //coloca a chave passada pelo usuário para uma chave temporária
    for(i=0;i<8;i++){
      chave[i]=chave64[i];
    }
```

```
//atribui os primeiros 12 bytes do pacote descriptografado
//para um outro vetor que irá conter os dados cifrados
//os 12 primeiros bytes serão cifrados com xor simples

j=0;
for(i=0; i < 12;i++){
  if( j == strlen((char*)chave)-1  ) j=0;
  pacote52enc[i] = pacote52bytes[i] ^ chave[j];
  j++;
}

//laço principal do One time PAD para criptografar

while( blocof <= 52 ){
   j=0;
   //pega 8 bytes (64bits) do pacote original por vez
   //pego de 64 em 64 pq o DES aceita 64 bits de chave e 64bits de tamanho do bloco a cifrar
   //por esse mesmo motivo não criptografo os 52 bytes pq não é multiplo de 64
   //e se eu passar os 52 bytes o DES fará um alinhamento dos dados fazendo o pacote finaol ter 56 bytes
   //esse pacote maior quebrará a arquitetura do raduino portanto, criptografamos apenas 40 bytes
   for(i = blocoi ; i < blocof ; i++ ){

     bloco64[j] =  pacote52bytes[i];
     j++;
   }
   //criptografa com DES
   des.encrypt(bloco64enc, bloco64, chave);

   j=0;
   //atribui o bloco de 64 bits criptografado para o vetor que conterá todos os bytes criptografados
   //a cada iteração desse while coloco mais 8 bytes no pacote final, até das os 52 bytes e sair do while
   for(i = blocoi ; i < blocof ; i++ )
   {
     pacote52enc[i] = bloco64enc[j];
     j++;
   }
```

```
        free(chave);
        chave=geraChave64( (byte*) bloco64enc );

        blocoi = blocof;
        blocof += 8;
    }
    free(chave);

    return pacote52enc;
}
```

========##########======================##########==========

Apêndice J - Linhas inseridas no firmware da Estação Rádio Base para fazer a decifragem com o OTP

========##########======================##########==========

```
//declaração das funções nesse arquivo de cabeçalho
void showArray(byte output[], int isize);

byte* geraChave64(unsigned char* chave);

byte* otp_enc52bytes(byte* pacote52bytes,byte *chave64);

byte* otp_des52bytes(byte* pacote52bytes,byte *chave64);

#include <MD5.h> // importa a biblioteca MD5
#include <DES.h> // importa a biblioteca DES
#include "Cript.h" // importa as funções de criptografia e geração de chave
#include "./printf.h"

//função mais de depuração para mostrar os bytes. Apenas para fins de teste
void showArray(byte output[], int isize)
{
  for (int i = 0; i < isize; i++)
  {
    if (output[i] < 0x10)
    {
      Serial.print("0");
    }
    Serial.print(output[i], DEC);
    Serial.print(" ");
  }
  Serial.println();
}
```

```
/*Gera a primeira chave fazendo um md5 na chave escolhida pelo usuário
  retorna apenas metade do md5 8bytes = 64bits que será usado pelo algoritmo DES
*/
byte* geraChave64(unsigned char* chave){
    unsigned char* hash=MD5::make_hash((char*) chave);

    byte* ret = (byte*) malloc(sizeof(byte)*8);
    for(int i =0; i < 8;i++){
      ret[i] =  hash[i];
    }
    free(hash);

    return ret;
}


/*
 Rotina que descriptografa o pacote criptografado. OS primeiros 12 bytes
 descriptografa com o XOR no restante do pacotes, descriptografa com o OTP
 Faz a mesma coisa usando o mesmo algoritmo. No entando, ela descriptografa ao invés de cifrar
 */

byte* otp_des52bytes(byte* pacote52bytes,byte *chave64){
    int blocoi=12;
    int blocof=20;
    byte bloco64enc[8];
    byte bloco64[8];
    byte* pacote52enc;
    byte* chave;
    int i,j;
    DES des;
    chave = (byte*) malloc(sizeof(byte)*8);
    pacote52enc = (byte*) malloc(sizeof(byte)*52);

    for(i=0;i<8;i++){
      chave[i]=chave64[i];
    }
```

```
j=0;
for(i=0; i < 12;i++){
  if( j == strlen((char*)chave)-1  ) j=0;
  pacote52enc[i] = pacote52bytes[i] ^ chave[j];
  j++;
}


while( blocof <= 52 ){
    j=0;
    for(i = blocoi ; i < blocof ; i++ ){

      bloco64[j] =  pacote52bytes[i];
      j++;
    }

    des.decrypt(bloco64enc, bloco64, chave);

    j=0;
    for(i = blocoi ; i < blocof ; i++ )
    {
      pacote52enc[i] = bloco64enc[j];
      j++;
    }


        free(chave);
        chave=geraChave64( (byte*) bloco64 );

        blocoi = blocof;
        blocof += 8;
    }
    free(chave);
    return pacote52enc;
}
```

========##########======================###########==================

Apêndice K – Linhas inseridas no firmware do sniffer com o XOR

==========

```
void PHY::sendSerial(packet * pkt)
{
  /* Escreve o pacote inteiro na porta serial */
  Serial.print("Temperatura: ");
  Serial.write((byte *)pkt, sizeof(packet));
  Serial.println("");
  Serial.print("Posicoes: ");
  Serial.write(pkt->AD0[0]);
  Serial.print(pkt->AD0[1]);
  Serial.println(pkt->AD0[2]);


  Serial.print("[");
  Serial.print(pkt->AD0[0], DEC);
  Serial.print("|");
  Serial.print(pkt->AD0[1], DEC);
  Serial.print("|");
  Serial.print(pkt->AD0[2], DEC);
  Serial.println("");
  Serial.print("]");

  UIB uib;
  uib.bytes[0] = pkt->AD0[2];
  uib.bytes[1] = pkt->AD0[1];
  Serial.print(" ================== ");
  Serial.println(uib.number);*/
}
```

==========