UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Jose Henrique Grotto

ANÁLISE E IMPLEMENTAÇÃO DE UMA CAMADA DE SEGURANÇA COMPLEMENTAR NA REDE LORAWAN

Santa Maria, RS
2021

Jose Henrique Grotto

ANÁLISE E IMPLEMENTAÇÃO DE UMA CAMADA DE SEGURANÇA COMPLEMENTAR NA REDE LORAWAN

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação, Área de Concentração em Área de concentração do CNPq, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. Vitalio Alfonso Reguera

Santa Maria, RS
2021

Endereço: Rua Doutor Liberato Salzano Vieira da Cunha, N. 475, Ap. 302
Fone (0xx) 55 99630 8014; End. Eletr.: jose.grotto@ecomp.ufsm.br

Jose Henrique Grotto

ANÁLISE E IMPLEMENTAÇÃO DE UMA CAMADA DE SEGURANÇA COMPLEMENTAR NA REDE LORAWAN

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação, Área de Concentração em Área de concentração do CNPq, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**.

Aprovado em 7 de junho de 2021:

Vitalio Alfonso Reguera, Dr. (UFSM)
(Presidente/Orientador)

Carlos Henrique Barriquello, Dr. (UFSM) (videoconferência)

Leonardo Londero De Oliveira, Dr. (UFSM) (videoconferência)

Santa Maria, RS
2021

DEDICATÓRIA

Dedico este trabalho a todo curso de Engenharia De Computação da UFSM, professores, funcionários e alunos, o qual fico muito orgulhoso de ter feito parte, e a todos aqueles a quem essa pesquisa possa ajudar de alguma forma.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à minha família, pelo apoio que sempre me deu durante toda a minha vida. Aos amigos e colegas, que sempre estiveram ao meu lado e me ajudaram a superar meus problemas, em especial ao Felipe, Eduardo, Nelson, Yuri e Fabio, que fizeram grupos de trabalho comigo e me aguentaram durante os mesmos. À universidade, pelas pessoas com quem convivi a longo desses anos de curso, que de alguma forma tiveram impacto na minha vida. Deixo um agradecimento especial ao meu professor orientador Vitalio, que sempre me auxiliou e teve paciência comigo durante este tempo, e que não desistiu de mim e do nosso trabalho nos momentos que eu estava desmotivado. E também, em especial, aos amigos do grupo do Discord, que mesmo estando longe geograficamente, sempre estão presentes.

RESUMO

ANÁLISE E IMPLEMENTAÇÃO DE UMA CAMADA DE SEGURANÇA COMPLEMENTAR NA REDE LORAWAN

AUTOR: Jose Henrique Grotto
ORIENTADOR: Vitalio Alfonso Reguera

A internet das coisas (IoT) vem ganhando popularidade nos últimos anos, e a tecnologia LoRa, com seu protocolo de comunicação LoRaWAN, têm sido muito usadas por pessoas e empresas, por atenderem as demandas do ramo da IoT. A Universidade Federal de Santa Maria (UFSM) não é uma exceção, pois também possui uma rede LoRa implementada. Porém, como o seu protocolo é recente, ele está sujeito a falhas de segurança, que podem comprometer os dados que trafegam pela rede. Por isso, foi proposto um estudo sobre instalação da rede LoRa. Este estudo revelou um potencial risco na utilização de um servidor de rede e aplicação de uso público, também usado pela UFSM. Então, propôs-se uma solução, através da implementação de uma camada adicional de segurança para os dados, utilizando um algoritmo de criptografia, criptografando os mesmos antes de serem transmitidos e os descriptografando após sua remoção do servidor. Escolheu-se um algoritmo de criptografia leve, chamado LEA-128, por ele ter características bastante compatíveis com tecnologias IoT, como utilização reduzida de recursos de memória e menor tempo de execução. Para validar o algoritmo, ele foi comparado com o padrão de segurança, o AES-128, e foi submetido a um ataque de força bruta e ao efeito avalanche. O algoritmo foi efetivo na encriptação e desencriptação de dados, e seus testes de validação mostraram que seu tempo de execução foi aproximadamente 30% mais rápido que o AES-128.

Palavras-chave: IoT. Segurança. LoRa. LoRaWAN. Criptografia. Algoritmo de Criptografia. LEA-128.

ABSTRACT

ANALYSIS AND IMPLEMENTATION OF A COMPLEMENTARY SECURITY LAYER IN LORAWAN NETWORK

AUTHOR: Jose Henrique Grotto
ADVISOR: Vitalio Alfonso Reguera

The Internet of Things (IoT) has been gaining popularity in recent years, and LoRa technology, with its communication protocol LoRaWAN, has been widely used by people and companies, for meeting the demands of the IoT industry. The Federal University of Santa Maria (UFSM) is no exception, as it also has implemented LoRa network. However, its protocol is recent, and it is subject to security flaws, which can compromise the data that travel over the network. Because of this, a study of the LoRa network installation at the university was proposed. This study revealed a potential threat with the use of a public network and application server by UFSM. Then, a solution was proposed, through the implementation of an additional layer of security for the data, using an encryption algorithm, encrypting the data before being transmitted and decrypting them after removing from the server. A lightweight encryption algorithm, called LEA-128, was chosen because it has features that are quite compatible with IoT technologies, such as reduced use of memory resources and less execution time. To validate the algorithm, it was compared with the security standard, AES-128, and was subjected to a brute force attack and the avalanche effect. The algorithm was effective in encrypting and decrypting data, and its validation tests showed that its execution time was, approximately , 30% faster than AES-128

Keywords: IoT. Security. LoRa. LoRaWAN. Encryption. Encription Algorithm. LEA-128.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
$LPWAN$	Low Power Wide Area Network
$LoRa$	Long Range
$LoRaWAN$	Long Range Wide Area Network
$UFSM$	Universidade Federal de Santa Maria
TTN	The Things Network
$NIST$	National Institue of Standards and Technology
CBC	Cipher Block Chain
CFB	Cipher Feedback
OFB	Output Feedback
CTR	Counter
XOR	Exclusive Or
AES	Advanced Encryption Standard
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorithm
TCP	Transmission Control Protocol
MAC	Media Access Control
IP	Internet Protocol
$CHIRP$	Compressed High Intensity Radar Pulse
SF	Spreading Factor
ADR	Adaptive Data Rate
$CMAC$	Cipher-based Message Authentication Code
MIC	Message Integrity Code
$OTAA$	Over-the-Air Activation
ABP	Activation by Personalization
SSL	Secure Sockets Layer
CPD	Centro de Processamento de Dados

API	Application Programming Interface
CPU	Central Processing Unit
LEA	Lightweight Encryption Algorithm
$LMIC$	LoRaWAN MAC in C
WSL	Windows Subsystem for Linux

LISTA DE SÍMBOLOS

T_{Exec}	Tempo de Execução
K	Chave do algoritmo de Criptografia
P	Texto simples
C	Texto Criptografado

SUMÁRIO

1 INTRODUÇÃO

O vasto uso da computação exige, há muito tempo, soluções para problemas de segurança de aplicações, a fim de proteger a informação. O avanço da tecnologia serviu como catalisador para a concentração no estudo dessas soluções, pois o aumento do uso de sistemas que compartilham dados, em que seus componentes (usuários que tem acesso) estando ou não separados geograficamente, introduziu problemas e complexidades (Willis H. Ware, 1979). Esses sistemas devem ser projetados para proteger cada usuário contra interferências de outros usuários ou do próprio sistema.

Muitos eventos contribuíram para a evolução da segurança de informações. Temos como exemplo, em 1915, a criação da máquina *Enigma* por criptógrafos poloneses, o qual convertia mensagens simples em textos criptografados. Em 1930, a quebra do código do *Enigma* pelo matemático Alan Turing. Nos anos 80, o aparecimento de hackers e cibercrimes. Nos anos 90, o começo da internet pública, etc (DAYA, 2013).

Nos anos recentes, a segurança da informação se desenvolveu em duas direções: Segurança de dados e segurança de rede (KEHE et al., 2009). A teoria da segurança de dados, baseada na criptografia, pesquisa a confidencialidade, a integridade e a disponibilidade dos dados, estratégias de defesa e assim por diante. Já a teoria da segurança de rede pesquisa a partir de protocolos, mecanismos e serviços de segurança, a fim de fornecer uma base teórica para o estabelecimento de um sistema de rede seguro.

Uma prática que sempre andou lado a lado com a segurança da informação é a criptografia. A ação de tornar registros normais em uma forma ilegível e, após, reverter o processo, retorna a milhares de anos. Primeiramente, desde técnicas simples de embaralhamentos e substituições de caracteres, passando para cifras polialfabéticas e, então, às técnicas mais conhecidas hoje em dia. Temos como principais o princípio do uso de chaves pseudo-aleatórias em processos simétricos, desenvolvido por Claude Shannon, e o principio de chaves públicas e privadas para processos assimétricos, anunciado por Diffie & Hellman (HILL, 2008). Ambos fazem parte da criptografia clássica, e são baseadas em técnicas matemáticas.

Atualmente, com o aperfeiçoamento da criptografia clássica, outros campos de estudo de técnicas estão surgindo, como a criptografia quântica, que é baseada na física e nas leis da mecânica quântica, e a criptografia neural, baseada em *Deep Learning*.

Agora, também, na era da conectividade, fala-se muito no conceito da Internet das Coisas (*Internet of Things* - IoT), onde objetos, como sensores, atuadores, eletrônicos, estão conectados a outros de alguma forma, permitindo a troca de dados e a permanência da conexão. Com isso, é possível fazer o controle de atividades e registros de lugares remotos, sem a necessidade da presença física de uma pessoa. Yang et al. (2018) supôs que, agora em 2021, teríamos mais de 13,5 bilhões de dispositivos conectados.

No ramo da IoT, é fundamental que um projeto eficiente cumpra com o compromisso da segurança e organização da rede de conexão e conte com componentes que gastem pouca energia. Por isso, várias empresas estão apostando em uma classe de tecnologia IoT, as Redes de ampla área e baixa potência (*Low Power Wide Area Networks* - LPWANs). Entre elas, está a tecnologia apresentada pela *Semtech*, a LoRa (*Long Range*), que significa longo alcance.

Essa tecnologia, juntamente com seu protocolo de comunicação LoRaWAN, têm chamado a atenção de empresas devido a suas características serem compatíveis com dispositivos IoT, facilitando aplicações e tráfego de dados, sejam elas nas indústrias, na agricultura, nos lares, etc. Da mesma forma, como o LoRaWAN é um protocolo razoavelmente novo, ele está sujeito a chamar, também, a atenção de cibercriminosos e usuários que possam, de alguma forma, se beneficiar com os dados que trafegam. Consequentemente, sua segurança esta constantemente sendo estudada e aprimorada, pois, como o objetivo da IoT é fazer nossa vida cotidiana melhor e mais fácil, seria ruim se informações fossem comprometidas a outros. E esse aprimoramento acaba não sendo uma tarefa simples, pois a tecnologia possui recursos limitados e, portanto, não pode executar funções de segurança tradicionais (ARAS et al., 2017).

Em vista disso, o objetivo deste trabalho tem um foco no aprimoramento da segurança dos dados que trafegam por essa rede. Para isso, são levantadas questões das características da rede LoRa e de como implementar segurança, através dos tipos de criptografia e dos algoritmos de criptografia. Então, são dados detalhes de como a rede é implementada na UFSM, da vulnerabilidade que ela possui e de que modo é possível solucionar isso. Após, essa solução é implementada e testada na própria rede.

1.1 JUSTIFICATIVA

Devido a esta tecnologia ser recente, as redes LoRa estão propensas a falhas de segurança que podem ser exploradas por atacantes. Através da exploração dessas falhas, é possível a infiltração na comunicação e a obtenção e modificação dos dados confidenciais que trafegam a rede.

Por ela ser, também, uma rede implementada e utilizada na UFSM, os dados que passam por ela podem estar sujeitos a estas mesmas falhas. Assim, esses dados, que são usados pela universidade por diversos motivos, podem ser roubados, fazendo com que terceiros tenham conhecimento, ou até alterados, gerando resultados inesperados em pesquisas ou medições, com um risco muito baixo dos atacantes serem descobertos.

Devido a isso, motiva-se o estudo sobre o nível de segurança das redes LoRa, considerando o que já existe atualmente, e o constante aprimoramento, pois devido a recursos limitados, isto acaba sendo uma tarefa complicada. Estes estudos podem levar a solu-

ções para vulnerabilidades de exposição dos dados, permitindo garantir maior privacidade e segurança para quem as usa, entre elas, a UFSM.

1.2 OBJETIVOS

A partir da justificativa, o trabalho tem como objetivo geral aumentar a segurança das informações que trafegam na rede LoRa, a qual está instalada na UFSM. Com isso, são propostos os seguintes objetivos específicos:

- Fazer uma descrição de uma rede LoRa e de seu protocolo LoRaWAN, e introduzir informações sobre segurança e criptografia.
- Fazer uma análise sobre a instalação da rede LoRa e do protocolo LoRaWAN na UFSM, observando módulos de transmissão instalados, *gateways* que estão conectados, servidores de rede e aplicação usados, etc;
- Identificar e implementar uma camada adicional de segurança nos dados que trafegam a rede, através da escolha de um algoritmo de criptografia, que não interfira significativamente nas características da mesma;
- Justificar a seleção do algoritmo, apresentando comparações com outros, como ocupação de memória, tempo computacional, e testes para avaliar a robustez, através de um ataque de força bruta e análise do efeito avalanche.

1.3 METODOLOGIA

Primeiramente, pesquisou-se em livros, vídeos, artigos, especificações, *webnars*, informações sobre a rede LoRa, o protocolo LoRaWAN e tipos e algoritmos de criptografia. Então, foi obtido informações sobre a rede na UFSM, com alunos, professores e empresas que participaram de sua instalação.

Com isso, analisou-se os serviços que a UFSM usa para a rede. A partir deles, foram estudados falhas na segurança desses serviços, que poderiam levar à exposição dos dados, e maneiras de cobrir as mesmas, através de ideias utilizando algoritmos criptográficos.

Após isso, foi estudado e escolhido um algoritmo para a criptografia, a partir de algumas características que fazem parte, também, da rede. Após a escolha, o fez-se a comparação do algoritmo selecionado com um dos algoritmos criptográficos mais utilizados atualmente.

Realizada a comparação, foi simulada uma transmissão de dados pela rede LoRa da universidade. Para ela, utilizou-se um Raspberry pi, com um *script* em linguagem Python para uma configuração rápida de frequência, compatível a do *gateway* instalado na UFSM. Também foi criado uma aplicação nos serviços utilizados pela UFSM, para receber os dados puros e criptografados. Então, foram observados, extraídos e, para os dados cifrados, foi aplicado novamente o algoritmo, para a desencriptação.

Por fim, foram feitos alguns testes para observar a robustez do algoritmo escolhido. Criou-se um código de um ataque de força bruta, para tentar decifrar os dados, o qual foi observado o tempo de execução e feito um cálculo do tempo estimado que o algoritmo iria levar, e foi analisado o efeito avalanche do algoritmo, observando o quanto a saída do mesmo mudava quando alterava-se minimamente a entrada.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte forma. Na seção 2 é feita a fundamentação teórica do trabalho, trazendo informações sobre os elementos usados, como a rede LoRa e a segurança de dados através da criptografia. A seção 3 apresenta o caminho para o problema encontrado, juntamente com uma solução. Para a solução, ela descreve o algoritmo usado, as comparações feitas entre ele e o algoritmo padrão de segurança, os testes de robustez e os *hardwares* e *softwares* usados. A seção 4 mostra os resultados da solução do problema, das comparações feitas e do testes. A seção 5 conclui o trabalho e apresenta ideias futuras para seu aprimoramento.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo propõe um embasamento das informações que foram utilizadas como base na proposta e desenvolvimento do trabalho. Por isso, serão abordados dados de características e funcionamento de uma rede LoRa, com seu protocolo, tendo um foco adicional na organização de sua segurança, e dados sobre a técnica de segurança através da criptografia, definindo tipos e citando algoritmos.

2.1 SEGURANÇA E CRIPTOGRAFIA

O manual de segurança de computadores do Instituto Nacional de Padrões e Tecnologia (*National Institute of Standards and Technology* - NIST) define o termo Segurança de Computadores da seguinte forma: "A proteção oferecida para um sistema de informação automatizado, a fim de alcançar os objetivos de preservar a integridade, a disponibilidade e a confidencialidade dos recursos do sistema de informação (incluindo hardware, software, firmware, informações/dados e telecomunicações)"(NIST95, 1995). Essa definição nos introduz três objetivos principais da segurança, tanto de computadores, quanto de redes: Confidencialidade, integridade e disponibilidade.

Confidencialidade assegura que indivíduos não autorizados não tenham acesso às informações privadas e que os indivíduos autorizados controlem seus dados e destinatários. Integridade assegura a prevenção contra a modificação sem autorização de informações e programas e que um sistema execute suas funcionalidades ileso de manipulações deliberadas ou inadvertidas. Disponibilidade assegura um acesso e uso rápido das informações, e sem que seus serviços fiquem indisponíveis para usuários autorizados. Também há outras definições no campo da segurança, como a Autenticidade, a qual assegura a verificação de que os usuários são quem dizem ser e que cada entrada no sistema vem de uma fonte confiável.

Em uma rede, uma mensagem deve ser transferida de uma parte para outra. Um canal de informação é estabelecido pela definição de uma rota da origem ao destino e, pelo uso cooperativo de algum protocolo de comunicação. Quando é necessário proteger as informações da transmissão de um atacante, que pode ser uma ameaça à confidencialidade, autenticidade e assim por diante, é necessário o uso dos requisitos de segurança.

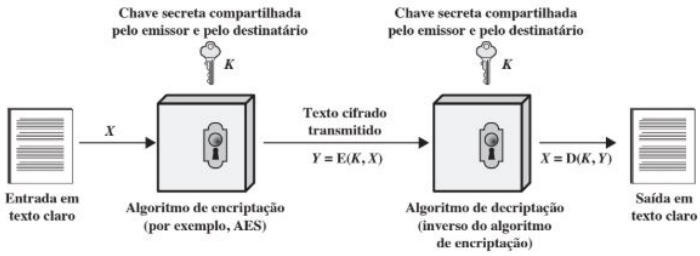
Os requisitos de segurança, à primeira vista, parecem ser claros e simples. Porém, os mecanismos usados para satisfazê-los são bastante complexos e sempre devem estar sendo atualizados, pois segurança é uma batalha infinita entre criminosos que tentam encontrar brechas e projetistas que tentam cobri-las. É dentro desses mecanismos que encontramos a criptografia, que é o uso de algoritmos matemáticos para transformar os

dados em um formato inteligível. Temos a criptografia reversível, o qual os dados podem ser cifrados e decifrados através de um algoritmo, utilizando chaves, usados em aplicações de confidencialidade. Temos, também, a criptografia irreversível, o qual os dados são apenas codificados, para serem usados em aplicações de integridade e autenticação. Os algoritmos não precisam ser mantidos em segredo, porém as chaves, que são entradas para estes algoritmos, sim, pois as transformações que ocorrem nos dados ao utilizar algum algoritmo dependem delas.

2.1.1 Criptografia Simétrica

A criptografia simétrica é um tipo de criptografia em que os lados de uma comunicação possuem conhecimento da chave secreta, e conseguem cifrar e decifrar os dados fazendo a utilização da mesma. Ela possui cinco itens: O texto original, que será enviado e deve ser cifrado, o algoritmo de encriptação, que realiza as transformações no texto original, a chave secreta, que modifica as transformações que o algoritmo faz, o texto cifrado, o qual é a mensagem embaralhada após a criptografia, e o algoritmo de decriptação, que é semelhante ao de encriptação, porém executado de modo inverso com a mesma chave, gerando o texto original. Esses itens são mostrados na Figura 2.1 (STALLINGS, 2014).

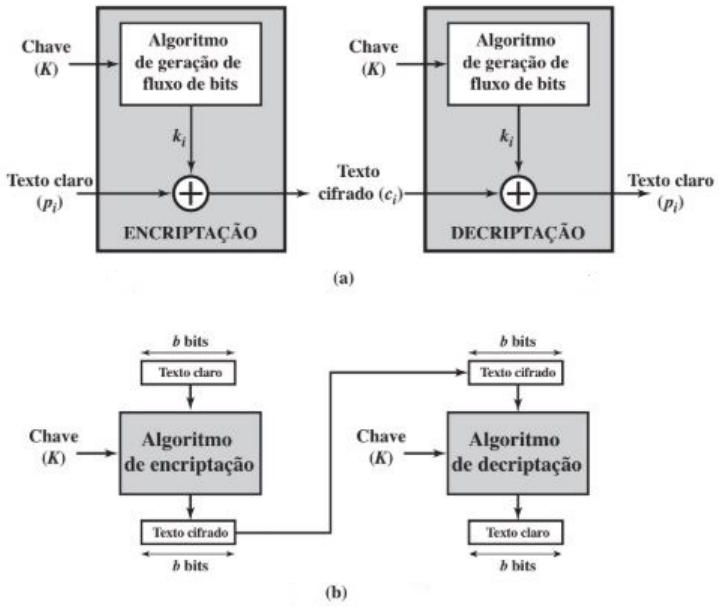
Figura 2.1 – Modelo de criptografia simétrica



Fonte: Adaptado de Stallings (2014).

Para este tipo de criptografia funcionar, ele deve ter um algoritmo forte, em que, mesmo o atacante tendo conhecimento, ele deve ser incapaz de decifrar o texto ou encontrar a chave, assim como o remetente e o destinatário da comunicação devem obter, com segurança, a chave secreta e mantê-la em segredo. Além disso, o modo como os dados originais são processados em diferentes algoritmos também influenciam. Esses modos são chamados de cifra de bloco e cifra de fluxo.

Figura 2.2 – Modos de Cifra para Criptografia. (a) Cifra de Fluxo. (b) Cifra de Bloco



Fonte: Adaptado de Stallings (2014).

2.1.1.1 Cifra de Fluxo

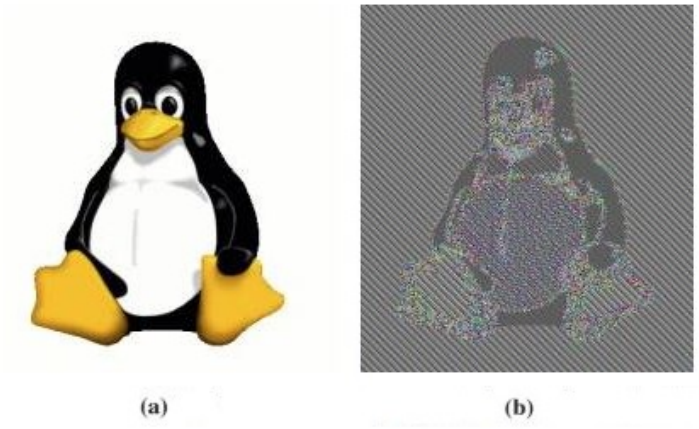
A cifra de fluxo é aquela que utiliza de um algoritmo para encriptar um fluxo de dados digital, sendo um bit ou um byte por vez, o que as tornam mais rápidas. Esse tipo de cifra gera uma sequência de bits a partir da chave inicial e de um gerador de dígitos pseudo-aleatório, chamada de fluxo de chave, que é usado como chave no algoritmo. Assim, a encriptação ocorre na combinação dos bits ou bytes do texto com os do fluxo de chave. Para esse modo de cifra funcionar, os dígitos pseudo-aleatórios devem ser livres de padrões que permitam atacantes detectarem relações entre os fluxos e chaves. O problema da cifra de fluxo é que a maior parte dos algoritmos garante apenas privacidade, e não integridade.

2.1.1.2 Cifra de Bloco

A cifra de bloco é aquela que utiliza de um algoritmo para encriptar blocos de dados de tamanho fixo para produzir blocos cifrados do mesmo tamanho, utilizando da chave secreta. Esse modo é melhor para criptografar dados estáticos, onde sabe-se o tamanho do texto e a divisão dele em blocos de um numero de bits. Normalmente, algoritmos de cifra de blocos produzem a mesma criptografia para cada bloco se usado a mesma chave,

como mostra a Figura 2.3.

Figura 2.3 – Cifra de Bloco Simples. (a) Imagem para ser cifrada. (b) Imagem cifrada.



Fonte: Adaptado de Stallings (2014).

Devido a esse problema, são usados alguns modos, chamados de modos com realimentação (*feedback modes*). São eles o modo *Cipher Block Chain* (CBC), o modo *Cipher Feedback* (CFB), o modo *Output Feedback* (OFB) e o modo *Counter* (CTR).

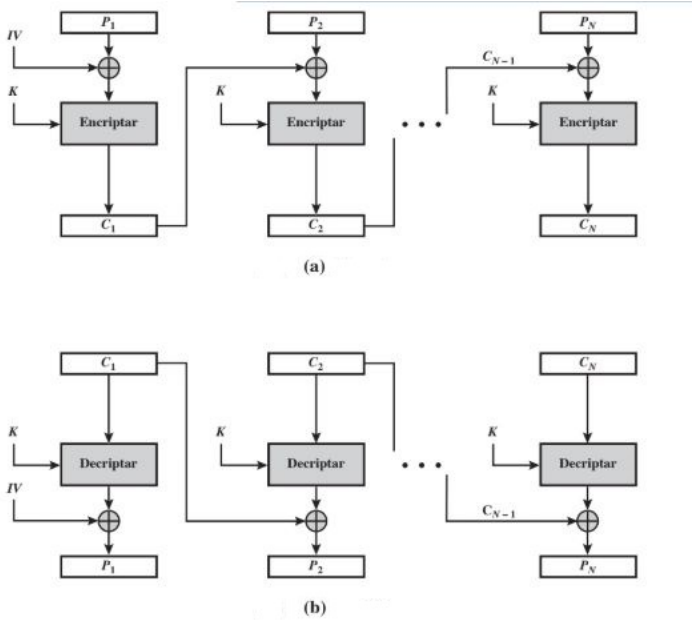
No modo CBC, a entrada do algoritmo de criptografia é uma operação XOR entre o bloco de texto original e o bloco de texto cifrado anteriormente, todos utilizando a mesma chave. Isso provoca um encadeamento do processo de criptografia. Para o primeiro bloco, é usado um vetor de inicialização

No modo CFB, a entrada do algoritmo de criptografia é o texto cifrado anteriormente, porém deslocado por uma unidade de n bits. Após a encriptação, os n bits mais significativos do resultado fazem uma operação XOR com um segmento do texto original, para produzir outra unidade de texto cifrada. Para a primeira unidade, é usado um vetor de inicialização.

No modo OFB, a estrutura é semelhante ao do modo CFB. Porém, ele trabalha com blocos de texto cheios, e a saída da função de encriptação é alimentada de volta para a entrada da função de encriptação do bloco de texto seguinte. Nesse modo é usado como inicialização um *Nonce*, ou seja, um numero que não se repete.

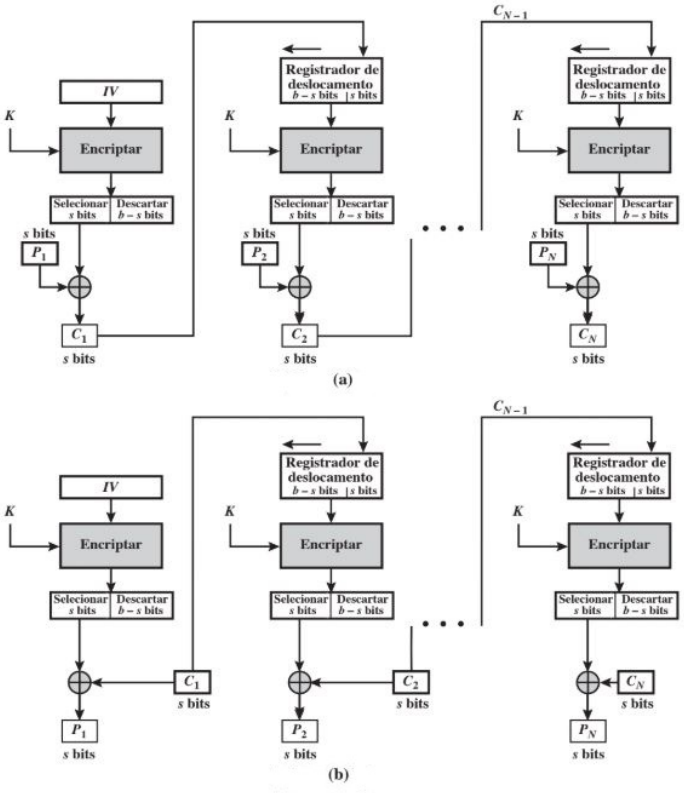
No modo CTR, usa-se um contador para a entrada do algoritmo de criptografia, o qual é iniciado com um valor, e incrementado a cada bloco subsequente. O contador é cifrado e passa por uma operação XOR com o bloco de texto original, a fim de produzir o bloco de texto cifrado. Não existe encadeamento. Esse modo possui algumas vantagens, como a simplicidade, pois não é necessário um algoritmo de decriptação diferente, eficiência, pois os blocos podem ser processados paralelamente, etc...

Figura 2.4 – Cifra de Bloco com o modo CBC. (a) Processo de encriptação. (b) Processo de decriptação.



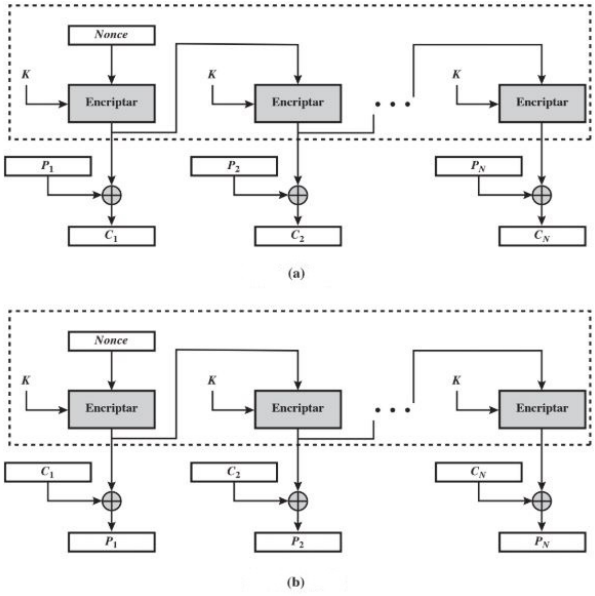
Fonte: Adaptado de Stallings (2014).

Figura 2.5 – Cifra de Bloco com o modo CFB. (a) Processo de encriptação. (b) Processo de decriptação.



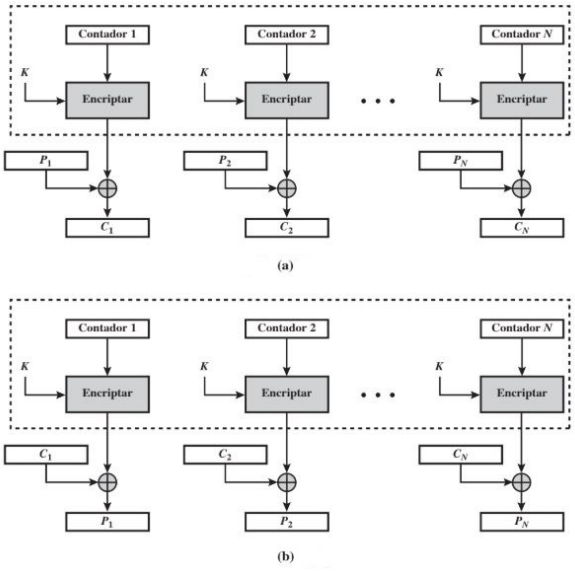
Fonte: Adaptado de Stallings (2014).

Figura 2.6 – Cifra de Bloco com o modo OFB. (a) Processo de encriptação. (b) Processo de decriptação.



Fonte: Adaptado de Stallings (2014).

Figura 2.7 – Cifra de Bloco com o modo CTR. (a) Processo de encriptação. (b) Processo de decriptação.



Fonte: Adaptado de Stallings (2014).

Há um número maior de pesquisas para a análise de cifras de bloco, em geral, por elas serem adequadas a uma gama maior de aplicações. A maioria das aplicações de criptografia simétrica, baseadas em redes, utilizam cifras de bloco.

Tabela 2.1 – Resumo da estrutura do Algoritmo AES

Algoritmo	Tamanho da Chave	Tamanho do Bloco	Número de Rodadas
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Fonte: Adaptado de Stallings (2014).

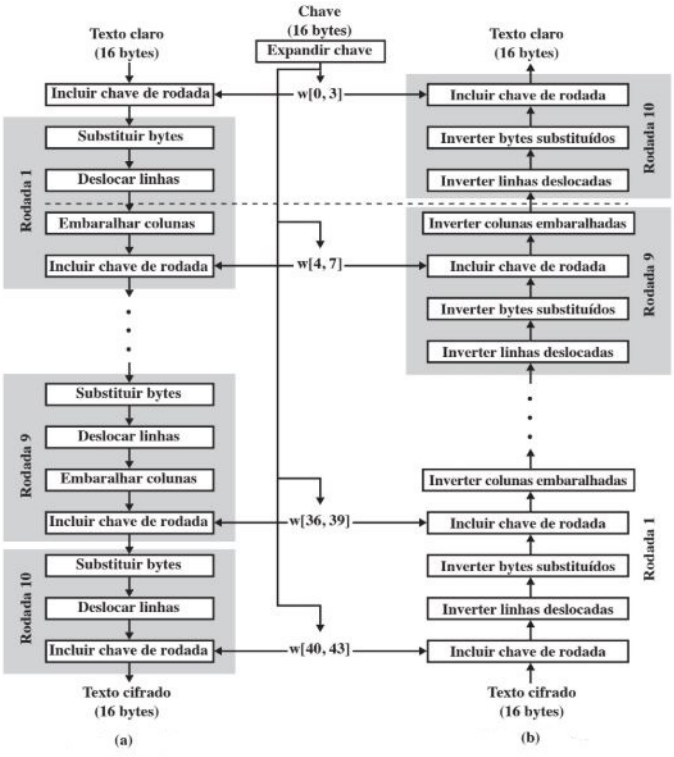
2.1.1.3 AES - Advanced Encryption Standart

O AES é um algoritmo de cifra de bloco, recomendado pelo NIST, que se tornou um dos mais populares entre os algoritmos de criptografia simétrica. Ele tem vantagens comparado a outros algoritmos, como velocidade, facilidade na implementação e dificuldade para atacar, porém sua estrutura é bastante complexa e não é explicada tão facilmente (YU et al., 2018).

O algoritmo cifra blocos de 128 bits, com diferentes tamanhos de chaves, sendo 128 bits (AES-128), 192 bits (AES-192) e 256 bits (AES-256). O número de iterações do algoritmos, ou rodadas, depende do tamanho da chave, sendo 10, 12 ou 14 rodadas. Ele consiste em um módulo de encriptação, decriptação e expansão de chave. Para simplificação, o algoritmo será explicado utilizando a chave de 128 bits.

A expansão da chave é um processo de preparação da chave para cada rodada. O algoritmo utiliza como entrada uma chave de 16 bytes (128 bits) e o expande para uma chave de 176 bytes (1408 bits). Então, essa chave é dividida pela número de rodadas + 1, gerando um vetor de chaves de 16 bytes, uma para cada rodada, chamado de *RoundKey* (chave de rodada). A encriptação consiste em quatro passos por rodada. *AddRoundKey* (incluir chave de rodada), *SubBytes* (substituir bytes), *ShiftRows* (deslocar linhas) e *MixColumns* (embaralhar colunas). Porém, a rodada final contém apenas três passos (não há o *MixColumns*), e há um inicial único (*AddRoundKey*) antes da primeira rodada. Vale observar que o passo *AddRoundKey* é chamado onze vezes, sendo o número de rodadas + 1, valor em que o vetor de chaves expandido é dividido. O algoritmo utiliza um bloco de 128 bits rearranjado em uma matriz de bytes 4x4 como entrada. No passo *AddRoundKey*, cada byte da matriz faz a operação XOR com a *RoundKey*. Seguindo, no passo *SubBytes*, cada byte é substituído por outro correspondente, de acordo com uma tabela de referência (chamada de *S-box*). No passo *ShiftRows*, cada linha é deslocada de um determinado número de posições. E no passo *MixColums*, os 4 bytes de cada coluna são combinados, utilizando uma transformação linear. A saída do algoritmo é uma matriz de bytes, também 4x4. A decriptação é semelhante a encriptação, porém invertendo o processo.

Figura 2.8 – Algoritmo AES. (a) Processo de encriptação. (b) Processo de decriptação



Fonte: Adaptado de Stallings (2014).

2.1.2 Criptografia Assimétrica

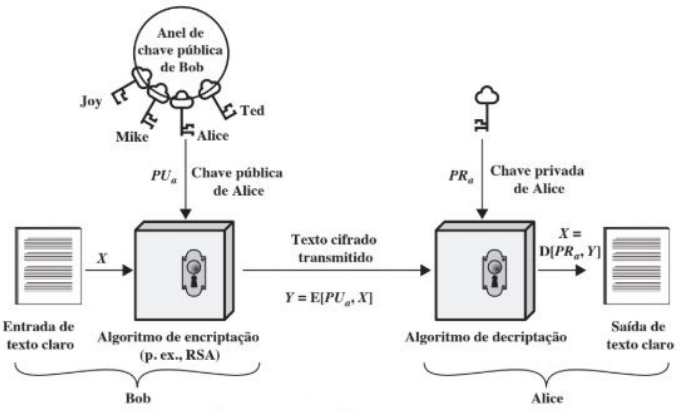
A criptografia assimétrica, mais conhecida como criptografia de chave pública, é um tipo de criptografia em que os lados da comunicação envolvem o uso de duas chaves secretas separadas. Além disso, enquanto a criptografia simétrica é baseada em substituições e permutações, a assimétrica faz a utilização de funções matemáticas. Isso não faz dela superior ou inferior, pois o que depende nos esquemas de criptografia é o tamanho das chaves e o trabalho computacional para quebrar a cifra (STALLINGS, 2014).

Esse conceito evoluiu devido a alguns problemas difíceis, associados à criptografia simétrica, sendo eles a distribuição de chaves e o conceito de assinatura digital. No primeiro caso, como dois sistemas conseguiriam compartilhar uma chave secreta, sem antes nunca terem se comunicado? Dessa forma, seria necessário uma entidade terciária, o qual é confiável para os dois lados, e que conseguisse fazer esse compartilhamento. Porém, essa entidade poderia ser comprometida por algum motivo, e a chave acabar sendo revelada. No segundo caso, com o uso da criptografia se tornado comum, a comunicação precisaria de assinaturas, equivalentes das usadas nos documentos em papel, para ter o conhecimento de que uma mensagem digital foi enviada pelo determinado sistema com que é desejado se comunicar.

Um esquema de criptografia de chave assimétrica possui, também, cinco elementos: O texto original, que serve de entrada para o algoritmo, o algoritmo de encriptação,

que realiza as transformações no texto original, as chaves publicas e privadas, um par de chaves selecionado de modo que, se uma for usada para encriptação, a outra é para desencriptação, o texto cifrado, o qual é a mensagem embaralhada, e o algoritmo de desencriptação, que realiza transformações no texto cifrado, utilizando a chave correspondente, e produz o texto original. A Figura 2.9 apresenta os elementos.

Figura 2.9 – Modelo de criptografia assimétrica



Fonte: Adaptado de Stallings (2014).

Para esta criptografia funcionar, ela deve seguir uma característica importante. Cada sistema deve colocar uma das duas chaves em um arquivo disponível, onde outro sistema possa acessa-la. Essa é a chave pública. A chave acompanhante permanece privada. Qualquer uma das duas chaves relacionadas pode ser usada para encriptação, com a outra para a decriptação. Porém, deve ser computacionalmente inviável determinar a chave privada, conhecendo o algoritmo e a chave publica.

2.1.2.1 RSA - Rivest-Shamir-Adleman

O algoritmo RSA tem sido usado como a técnica de uso geral mais aceita e implementada para a encriptação de chave publica. Ele é uma cifra que utiliza uma expressão exponencial. O texto original é encriptado em blocos, onde cada um possui um valor entre 0 e um número n, que depende do número de bits usado. A encriptação e desencriptação tem a seguinte forma, para um texto original P e um texto cifrado C:

$$C = P^e \bmod n \tag{2.1}$$

$$P = C^d \bmod n \tag{2.2}$$

As variáveis e e n são conhecidas pelo emissor e pelo receptor, e d apenas pelo

receptor. Assim, a chave pública é {e,n} e a chave privada é {d,n}. Para o algoritmo ser satisfatório, é necessário atender alguns requisitos:

- É possível encontrar valores de e, d e n, tais que $P = P^{ed} \bmod n$ para todo $M < n$;
- É relativamente fácil calcular C e P para todos os valores de $M < n$;
- Como dito anteriormente, deve ser inviável determinar d conhecendo e e n.

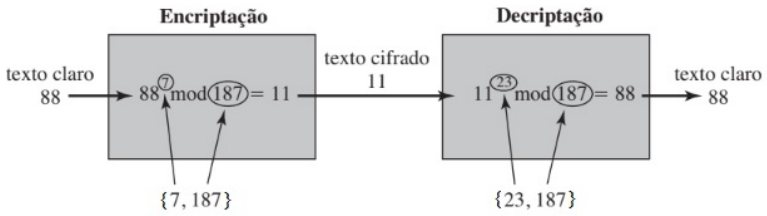
Dessa forma, é possível formular o esquema RSA, como é mostrado a seguir

- Selecionar p e q, sendo eles dois números primos diferentes;
- Calcular $n = p * q$;
- Calcular $\Phi(n) = (p - 1) * (q - 1)$;
- Selecionar e, sendo que o mdc($\Phi(n), e) = 1$ e $1 < e < \Phi(n)$;
- Calcular $d = e^{-}1 * (1 \bmod \Phi(n))$

Dessa forma, a chave privada consiste em {d,n} e a chave publica em {e,n}. Suponha que um usuário A tenha publicado sua chave publica, e um usuário B queira enviar uma mensagem para A. Então, B consegue encriptar o texto original utilizando a chave publica {e,n}. Após o envio, o usuário A recebe o texto cifrado, e com a chave privada {d,n}, ele consegue decriptar a mensagem. Na Figura 2.10 é possível ver um exemplo utilizando números.

1. Selecione dois números primos, $p = 17$ e $q = 11$;
2. Calcule $n = p * q = 17 * 11 = 187$;
3. Calcule $\Phi(n) = (p - 1) * (q - 1) = 16 * 10 = 160$;
4. Selecione e, tal que e seja relativamente primo de $\Phi(n) = 160$ e menor que $\Phi(n)$; e = 7;
5. Determine d, tal que $d * e = 1(mod160)$ e $d < 160$. O valor é $d = 23$, pois $23 * 7 = 161 = (1 * 160) + 1$.

Figura 2.10 – Exemplo do uso do Algoritmo RSA



Fonte: Adaptado de Stallings (2014).

2.1.3 Criptografia para Integridade de Dados

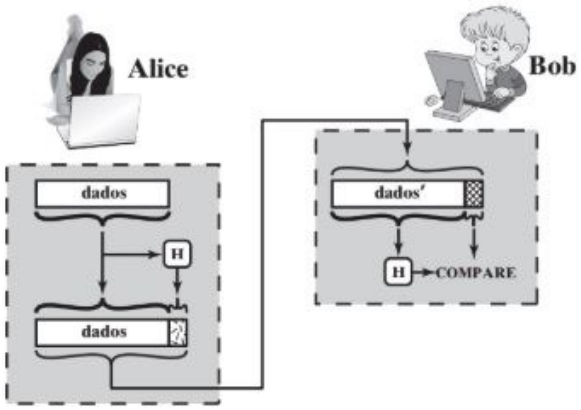
A criptografia para integridade de dados, conhecida por função de *hash* criptográfica, é um tipo de criptografia irreversível, o qual aceita uma mensagem de tamanho variável como entrada e produz um valor, chamado de *hash*, de tamanho fixo. Esse valor serve para ser analisado e garantir a integridade da mensagem, pois uma mudança em qualquer bit da mensagem original resulta, com alta probabilidade, em uma mudança no código *hash* (STALLINGS, 2014).

Para uma função *hash* funcionar, ela deve transformar mensagens, a ponto de ser computacionalmente inviável descobrir o mapeamento de um resultado de *hash* pré-especificado, ou seja, inviável descobrir o resultado de uma *hash* antes de aplicar o algoritmo, e descobrir duas mensagens que sejam mapeadas para o mesmo resultado. Por conta disso, essas funções são usadas com frequência para determinar se os dados mudaram.

Devido a sua versatilidade, as funções hash possuem varias aplicações, como para integridade de mensagens, assinaturas digitais, criação de arquivos de senha de mão única, detecção de intrusão ou vírus, etc. Quando usada para autenticação de mensagens, o seu resultado é chamado de resumo da mensagem. O emissor calcula um valor de *hash* como uma função dos bits da mensagem, e transmite a mensagem original juntamente com o valor de hash. O receptor realiza o mesmo calculo sobre os bits da mensagem e compara o resultado com o valor de *hash* recebido. Caso for diferente, a mensagem foi alterada. Quando usada para assinatura digital, o valor de *hash* da mensagem é encriptado com a chave privada do usuário. Assim, qualquer um que conhecer a chave publica do mesmo, pode verificar a integridade da mensagem. Isso permite também a autenticação, pois somente o emissor poderia ter produzido o código hash encriptado. Outro caso, focado mais para segurança do sistema, é para a criação de um arquivo de senhas. Em vez das próprias senhas do usuário serem armazenadas por um sistema operacional, o valor *hash* é armazenado. Desse modo, a senha não pode ser recuperada por quem conseguir acessar o arquivo. Quando o usuário informa uma senha, o sistema compara o *hash* dela com o armazenado no arquivo. Também para a detecção de vírus, pois eles alteram as

mensagens e programas, o que acabam mudando o valor *hash*.

Figura 2.11 – Modelo de criptografia para Integridade de Dados



Fonte: Adaptado de Stallings (2014).

2.2 LORA E O PROTOCOLO LORAWAN

Em uma rede de computadores, é necessário haver um modelo padrão para os protocolos de comunicação. Atualmente, o modelo utilizado é o TCP/IP, uma arquitetura que divide as redes de computadores em quatro camadas: A de acesso a rede, a de internet, a de transporte e a de aplicação. Porém, diferentemente de uma rede de computadores, uma rede LoRa possui apenas duas camadas: Uma camada física e uma camada de acesso ao meio (*Media Access Control* - MAC)(para o modelo TCP/IP, ambas seriam subcamadas da de acesso a rede). Ela está cada vez mais sendo utilizada, no contexto das LPWANs, devido a sua rede possuir um link de comunicação de longo alcance, baixo gasto de energia, alta capacidade de nós na rede, serviço de alta qualidade, bi-direcional e alta segurança. Porém, segundo Devalal e Karthikeyan (2018), para garantir essas características de longo alcance e baixo gasto de energia, ela acaba sacrificando sua taxa de dados na hora da comunicação, ou seja, não é possível transmitir arquivos que requerem grandes taxas, como vídeos.

2.2.1 LoRa (Long Range)

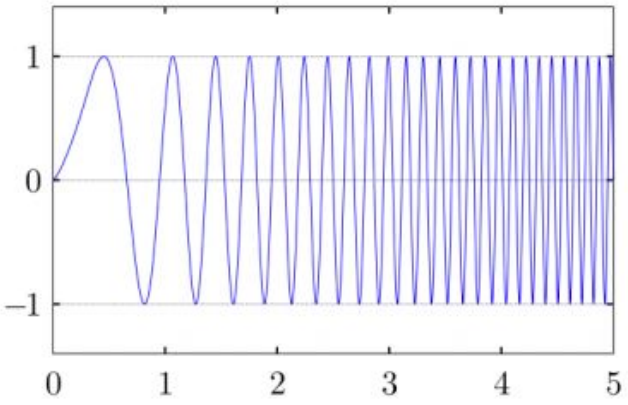
LoRa é uma camada física, criada pela *Semtech*, que proporciona um *link* de comunicação de longo alcance com baixo custo de energia. Ela define, principalmente, a técnica de modulação, chamada de *LoRa Spread Spectrum*. Ela opera nas bandas de frequência

sem licença, reservadas para o desenvolvimento industrial, científico e médico (*Industry Scientific and Medical* - ISM) (NEIS, 2017).

2.2.1.1 LoRa Spread Spectrum

A modulação de rede sem fio utilizada é uma variação da *Chirp Spread Spectrum*, uma técnica que vem sendo usada para fins militares, e agora comercialmente, devido a sua natureza robusta e longo alcance. Na modulação, o sinal de dados é multiplicado pelo sinal *Chirp* (*Compressed High Intensity Radar Pulse* - Pulso de Radar Comprimido de Alta Intensidade), que é um sinal em que a frequência aumenta e diminui com o tempo. Isso espalha a largura de banda além da banda do sinal original. No receptor, o sinal é comprimido de volta. Isso proporciona uma robustez contra ruídos e interferências, e um baixo requerimento de energia para a transmissão (DEVALAL; KARTHIKEYAN, 2018). Exemplo de um sinal *Chirp* no tempo na Figura 2.12.

Figura 2.12 – Exemplo de sinal *Chirp*



Fonte: Adaptado de Sakshama Ghoslya (2017).

2.2.1.2 Fator de Espalhamento

O fator de espalhamento (*Spreading Factor* - SF) é um parâmetro que define o número de *Chirps* em que o sinal de dados é multiplicado. Esse fator troca distância de comunicação pela taxa de transmissão. O SF pode ser configurado de 7 a 12. Quanto maior o SF, maior a sensibilidade do receptor, o que permite comunicações mais distantes, mas, também, maior tempo dos dados no ar (ZORBAS et al., 2018).

2.2.2 LoRaWAN (*Long Range Wide Area Network*)

Além da camada física, uma rede LoRa é composta, também, por um protocolo da camada de acesso ao meio (MAC), chamado de LoRaWAN. Esse protocolo controla o fluxo de dados, estabelece uma norma de comunicação entre os sistemas conectados e define a arquitetura da rede. Essa norma garante as características citadas anteriormente, como a alta capacidade de nós na rede, serviço de alta qualidade, a bidirecionalidade e alta segurança.

2.2.2.1 Arquitetura

A arquitetura do LoRaWAN foi proposta no contexto da IoT para fornecer uma comunicação com padrões transparentes, sem a necessidade de instalações complexas. Ela é composta por quatro elementos básicos: Os Nós LoRa (*End-Nodes*), o *Gateway*, o servidor de Rede (*Network Server*) e o servidor de Aplicação (*Application Server*) (LoRa Alliance Technical Marketing Workgroup, 2015).

2.2.2.1.1 Nós LoRa (*End Nodes*)

Os Nós LoRa compreendem sensores e aplicações (chamados também de *end devices*) onde ocorrem detecção e controle, como monitoramento de pressão, rastreamento de posição, medição de energia, alarmes, etc. Eles coletam dados e os enviam para um ou vários *gateways*, utilizando a tecnologia de modulação.

2.2.2.1.2 *Gateway*

O *Gateway* constitui o elemento de entrelaçamento de uma rede LoRa, pois ele é a conexão entre todos nós LoRa e o servidor de Rede. Eles examinam o sinal transmitido e recebem os pacotes LoRa dos nós, encaminhando os mesmos para um servidor, via Wi-Fi, 4G, *Ethernet* ou Satélite, os quais são manipulados caso sejam válidos.

2.2.2.1.3 Servidor de Rede

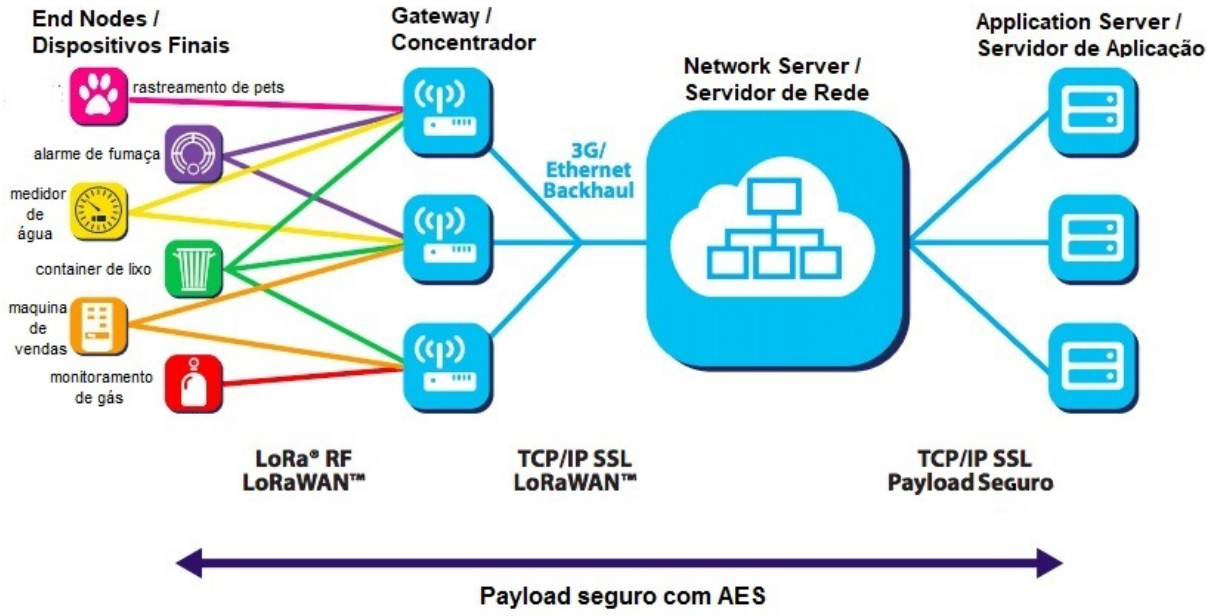
O Servidor de Rede é uma entidade inteligente que administra a rede. Os pacotes recebidos dos *Gateways* são filtrados e executados alguns recursos, como verificações de

segurança, ajuste das taxas de dados adaptáveis, etc. Após, ele envia para os *Gateways* uma mensagem de reconhecimento dos pacotes e identifica para quais aplicações os mesmos são pretendidos. Então, ele os envia para seus respectivos servidores de aplicação.

2.2.2.1.4 Servidor de Aplicação

Recebem os dados intencionados a eles, para serem analisados e manipulados. Nele, também, são executados recursos de segurança.

Figura 2.13 – Arquitetura de uma rede LoRa



Fonte: Adaptado de LoRa Alliance Technical Marketing Workgroup (2015).

2.2.2.2 Dispositivos finais (End Devices)

Na arquitetura, os dispositivos finais podem servir para diferentes aplicações e ter diferentes requerimentos, de acordo com a vida útil de sua bateria e a latência de comunicação do *downlink*. Para otimizar a variedade de perfis de aplicações, os dispositivos finais são divididos em três categorias (DEVALAL; KARTHIKEYAN, 2018).

2.2.2.2.1 Dispositivos finais classe A

Dispositivos finais classe A são os dispositivos mais econômicos. Eles suportam uma comunicação bidirecional assíncrona, no qual cada dispositivo possui uma janela de transmissão de *uplink*, seguida de duas janelas de recepção de *downlink*. A comunicação do dispositivo para o servidor de rede é chamada de *uplink*, e a contraria é chamada de *downlink*. Os dispositivos finais programam a janela de transmissão de acordo com suas necessidades de comunicação, com uma pequena variação, baseada no protocolo ALOHA, ou seja, ocupar o canal de transmissão sempre que tiverem dados a serem enviados.

2.2.2.2.2 Dispositivos finais classe B

Dispositivos finais classe B também suportam uma comunicação bidirecional, porém possuem janelas de recepção adicionais em horários programados. Esses horários são sincronizados com o *gateway*, permitindo que o servidor saiba quando o dispositivo final está pronto para receber informações, porém essa sincronização requer um gasto maior de energia.

2.2.2.2.3 Dispositivos finais classe C

Dispositivos finais classe C são outros dispositivos que suportam comunicação bidirecional, porém suas janelas de recepção estão abertas todo o tempo, fechando apenas quando ocorre uma transmissão. Devido a isso, eles consomem mais energia que os dispositivos das outras classes, mas providenciam a menor latência para a comunicação. Podem ser usados para dispositivos que não possuem restrições de energia.

2.2.2.3 Capacidade da Rede

Uma rede LoRa é organizada a partir da topologia de rede em estrela, onde todos os dispositivos finais estão conectados diretamente a um ou mais *gateways*, sem ligações entre si. Essa topologia é adequada para sistemas embarcados de baixa potência, pois ela elimina a transmissão de dados para dispositivos intermediários, gastando menos energia, e continua executando funções caso algum dispositivo final pare de funcionar. Também, um *gateway* recebe dados de muitos dispositivos finais. Devido a isso, ele deve ter uma alta capacidade. Isso é adquirido através de uma técnica chamada Taxa de Dados Adaptável (*Adaptive Data Rate* - ADR) (LoRa Alliance Technical Marketing Workgroup, 2015).

2.2.2.3.1 *Adaptive Data Rate*

É um mecanismo para otimizar a taxa de dados da transmissão. A rede coleta alguns dados, como a taxa de dados, a relação sinal ruído, o número de *gateways* que recebem os dados, a intensidade do sinal da transmissão, através de *uplinks*. Baseado neles, a rede decide o quanto pode aumentar a taxa de dados para ter uma eficiência energética e diminuição do tempo do sinal no ar. Isso tudo é feito através do uso de um algoritmo de ADR (Mathieu Monneret, 2019).

2.2.2.4 Recursos de Segurança

O principal desafio de garantir a segurança é que a tecnologia possui recursos limitados e, portanto, não pode executar funções de segurança tradicionais (ARAS et al., 2017). Também, como o canal de comunicação é sem fio, o trafego e a autenticidade se tornam preocupações, pois ele é acessível para a injeção ou modificação dos dados, e os dispositivos podem estar conectados a aparelhos hostis. A segurança de uma rede LoRa é projetada para atender aos seus critérios e, devido aos dispositivos ficarem em campos por tempos longos, deve ser à prova de futuro (Gemalto and Actility and Semtech, 2017). Seu *design* segue os princípios da integridade, autenticidade e confidencialidade.

2.2.2.4.1 Camadas

Enquanto muitas tecnologias incorporam apenas uma camada de segurança, o protocolo LoRaWAN envolve duas, a camada de segurança da rede e a de segurança da aplicação. Ambas usam o algoritmo de criptografia AES-128 (*Advanced Encryption Standart*), porém enquanto a primeira usa em modo CMAC (*Cipher-based Message Authentication Code*), responsável pela autenticação do dispositivo final e da integridade da mensagem, a segunda usa em modo CTR (*Counter*), responsável pela confidencialidade do dado, protegendo-o, também, do operador do servidor de rede.

Todo pacote de dados é encriptado, primeiramente, por uma chave AES-128 (128 bits) em modo CTR, compartilhada com o servidor de aplicação. Após, ele é encriptado novamente por outra chave AES-128 em modo CMAC, compartilhada com o servidor de rede, gerando um Código de Integridade da Mensagem (*Message Integrity Code* - MIC) e o enviando junto ao pacote. Essas chaves serão explicadas a seguir. Ao chegar no servidor de rede, ambos pacote e MIC são analisados, garantindo a autenticidade e evitando que o pacote fosse alterado. Assim, o pacote é desencriptado e enviado ao servidor de aplicação. La, ele é novamente desencriptado para poder ser analisado e manipulado (Gemalto and Actility and Semtech, 2017).

2.2.2.4.2 Identificadores e Chaves

O LoRaWAN especifica um número de identificadores para os dispositivos finais. Todos possuem um identificador único, chamado *Device Identifier* (DevEUI), o qual identifica o dispositivo na rede. Ele é definido pelos fabricantes. Outro identificador é o da aplicação, chamado de *Application Identifier* (AppEUI), o qual identifica o servidor de aplicação que está habilitado a fazer a conexão. Pode ser diferente ou o mesmo para os dispositivos (ARAS et al., 2017). Também há o DevNonce, um valor pseudoaleatório gerado pelo dispositivo final para a conexão, o qual só pode ser usado uma vez. Esse número é monitorado pelo servidor de rede, e usado para prevenir repetições (TOMASIN; ZULIAN; VANGELISTA, 2017).

Além dos identificadores, os dispositivos finais também possuem chaves para o algoritmo AES-128. Uma chave AES de 128 bits, conhecida como *Application Key* (Appkey) é usada para gerar duas chaves de sessões, os quais são chamadas de *Network Session Key* (NwkSKey) e *Application Session Key* (AppSKey). A NwkSKey é compartilhada entre o dispositivo final e o servidor de rede, para gerar e verificar o MIC , garantindo integridade e uma assinatura especifica para o dispositivo. A AppSKey é similar a NwkSKey, mas é usada para cifrar e decifrar os dados do pacote entre o dispositivo final e o servidor de aplicação (ARAS et al., 2017).

2.2.2.4.3 Modos de União

Para um dispositivo fazer parte em uma rede LoRa, ele deve ser conectado, ativado e autenticado, através de modos de união, utilizando as chaves e identificadores. O LoRaWAN especifica dois mecanismos para essa conexão, *Over-the-Air Activation* (OTAA) e *Activation by Personalization* (ABP).

O modo OTAA é o método mais seguro para autenticar um dispositivo final (ARAS et al., 2017), pois diferentes chaves de sessão são geradas especificamente para ele cada vez que entra na rede. O modo inicia com o dispositivo enviando uma mensagem de *Join Request*. Ela inclui os identificadores AppEUI, DevEUI, o número DevNonce e a chave AppKey. Ao recebê-la, o servidor de rede verifica se o dispositivo final pode ser aceito. Caso for, ele envia uma mensagem de *Join Accept*, contendo um número AppNonce, gerado pelo mesmo. Com esses dados, ambos os lados conseguem calcular as chaves NwkSKey e AppSKey. Porém, enquanto o dispositivo final calcula suas próprias chaves, uma entidade do servidor de rede, chamada de *Join Server*, faz o mesmo cálculo, e envia a NwkSKey para o servidor de rede e a AppSKey para o servidor de aplicação (Alper Yegin, 2019). Após isso, ocorre a comunicação. Um dispositivo final deve sempre seguir esse procedimento cada vez que se junta a uma nova rede ou perde a informação das suas chaves de sessão.

O modo ABP ignora a troca de mensagem inicial para a conexão. Nesse modo, os identificadores e chaves já estão salvos no dispositivo e nos servidores. Quando ocorre a tentativa de comunicação, ele manda mensagens diretamente, já criptografadas e autenticadas, de modo que apenas os servidores correspondentes podem decifra-las (YANG et al., 2018). Esse modo é menos seguro, pois se as chaves estiverem comprometidas, toda comunicação poderá ser descriptografada por terceiros, até que elas sejam atualizadas.

Vale ressaltar também que, os dados transmitidos do *gateway* para o servidor de rede também podem estar protegidos por protocolos, como o SSL. Como o *gateway*, geralmente, transmite os dados através da internet, e o SSL é o protocolo que estabelece comunicações seguras na Internet, os dados recebem essa proteção adicional durante esse processo. O protocolo SSL utiliza os princípios da criptografia de chave pública, descritos anteriormente.

3 ESTUDO DE CASO

3.1 ESCOLHA E CONTEXTUALIZAÇÃO DO PROBLEMA

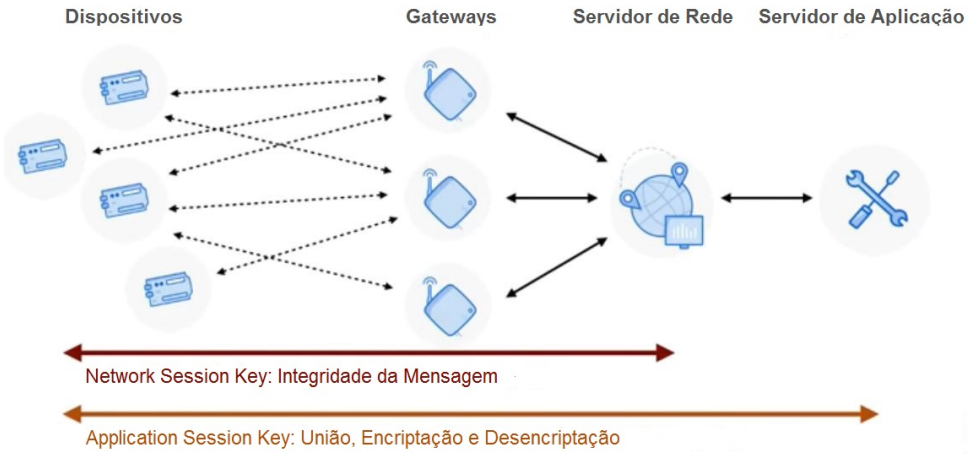
Através de informações coletadas com conversas com professores e alunos, soube-se que a rede LoRa foi instalada na UFSM pelo CPD (Centro de Processamento de Dados). Os dispositivos finais são, na maioria, medidores e sensores, ligados a um controlador e módulos LoRa, modelo RFM95 da empresa *RF Solutions*, instalados na universidade pela empresa *FoxIoT*, uma *StartUp* que trabalha com IoT e tecnologias LPWAN. Eles fazem a medição de energia elétrica (*Smart Meters*) nas construções, a fim de analisar variações e controlar gastos. Eles estão ligados a um *gateway*, modelo RHF2S008 da empresa *RisingHF*, instalado sobre o prédio da reitoria e conectado ao *The Things Network* (TTN). O TTN é um serviço de servidor de rede e aplicação gratuito e público, desenvolvido pela comunidade e baseada em contribuições voluntárias (*Open Source*). Nele, é permitido a criação de aplicações, com dispositivos finais vinculados à elas. Também são oferecidos APIs para que os dados sejam acessados e tratados. Ao passar pelo *gateway*, os dados são levados até o TTN, onde há aplicações em que eles são armazenados. Lá, os dados são retirados, organizados e disponibilizados aos professores da área no *website* da *FoxIoT* ou diretamente à aplicação instalada no servidor da UFSM.

Na teoria, tratando-se da segurança do protocolo LoRaWAN, o servidor de rede conhece a *Network Session Key*, responsável pela identificação, e o servidor de aplicação conhece a *Application Session Key*, responsável pela confidencialidade da carga (Figura 3.1). Porém o que acontece na prática com o TTN e com alguns outros serviços disponibilizados por terceiros, como a *ResIOT*, a *ChirpStack*, a *Proximus*, por exemplo, é que, além de servidores de rede são, também, servidores de aplicação, e eles lidam com os processos de adesão, encriptação e desencriptação. Isso ocorre porque esse sistema torna o uso da aplicação muito mais fácil para o desenvolvedor, pois não é necessário implementar nenhum mecanismo adicional (Figura 3.2).

Dentro desse contexto, teve-se a ideia de implementar uma camada adicional de segurança, pois como o TTN é, também, um servidor de aplicação, os dados armazenados nele já podem ter sidos descriptografados com a *Application Session Key*, e estão em texto puro. Assim, caso ocorra uma falha no sistema do TTN, ou exista alguma vulnerabilidade, em que uma ameaça possa se aproveitar, pode ocorrer vazamento e exposição dos dados. Isso se intensifica por dois motivos.

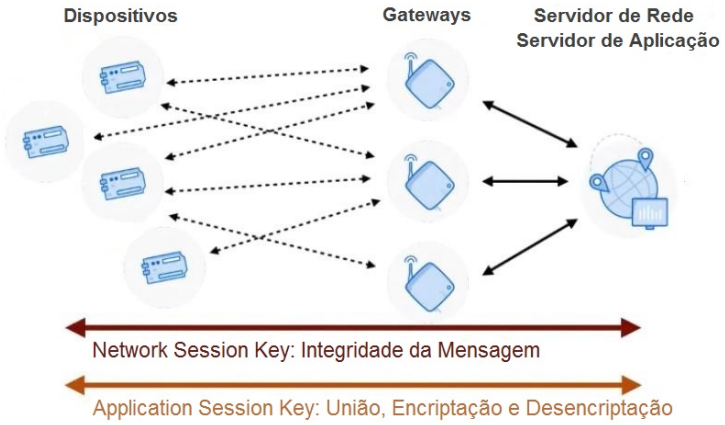
Um é devido a uma propriedade do manifesto do TTN, que diz “Qualquer pessoa que fizer uso da rede está autorizada a fazê-lo por qualquer motivo ou causa, possivelmente limitada pela legislação local, totalmente por seu próprio risco e sabendo que os

Figura 3.1 – Modelo Teórico da Segurança do protocolo LoRaWAN.



Fonte: Adaptado de The Things Network (2017).

Figura 3.2 – Segurança do Protocolo LoRaWAN na prática.



Fonte: Adaptado de The Things Network (2017).

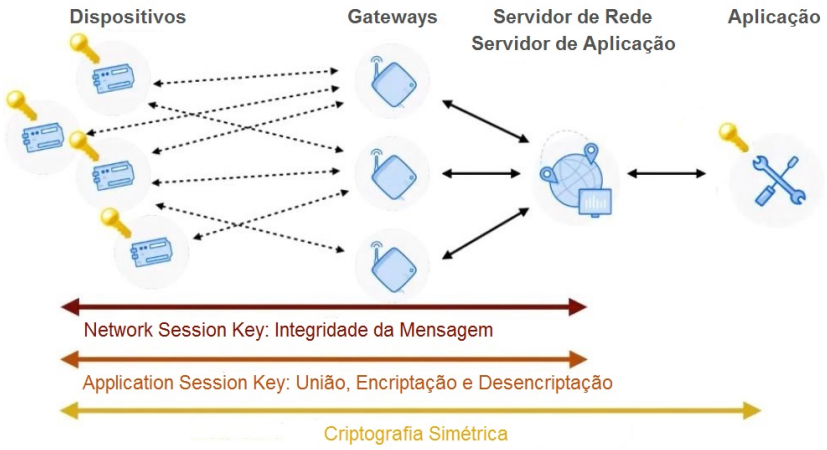
serviços são fornecidos “como estão” e podem ser encerrados por qualquer motivo a qualquer momento.”(Wienke Giezeman, 2017). Isso implica que, ao utilizar a rede do TTN, os riscos dela são aceitos, independente de quais forem. Outra é pelo fato de que o TTN não possui um grupo responsável pela análise de risco, ou seja, um grupo responsável pela governança, que observaria potenciais ameaças e falhas dentro de um escopo de sua estrutura, e trabalharia para fazer regras e mudanças, a fim de evitá-las. Um dos motivos de não existir esse grupo é que a comunidade inteira, que faz parte da rede do TTN, seria obrigada a cumprir essas decisões, o que vai contra uma propriedade do manifesto TTN, que diz “Os provedores de “*The Things Network*” não irão impor restrições aos seus usuários.”(Wienke Giezeman, 2017).

Por isso, a implementação de uma camada adicional de segurança, sem o envolvimento da rede do TTN, seria uma solução para esse problema. Para a construção dessa camada, foi cogitado o uso de um algoritmo de criptografia para cifrar os dados. Esse algo-

ritmo seria aplicado diretamente após a aquisição dos mesmos, antecedendo a transmissão, nos dispositivos finais. Isso manteria os dados, armazenados no servidor de aplicação, cifrados, e, ao retirá-los de la, apenas seria necessário aplicar o algoritmo novamente (Figura 3.3). Porém, por ser aplicado direto nos dispositivos finais, alguns fatores devem ser levados em consideração, como, a compatibilidade da linguagem de programação, utilizada no algoritmo, com a do dispositivo, a semelhança do tipo dos dados de entrada do algoritmo com os do tipo obtido na aquisição, etc.

Então, para desenvolver essa proposta, inicialmente, foi criada uma aplicação no TTN e a transmissão de uma mensagem simples pela rede LoRa da UFSM até ele, para confirmação da teoria de que os dados armazenados já foram descriptografados. Então, houve a escolha do algoritmo a ser usado para a encriptação da mensagem e, posteriormente, a transmissão da mesma mensagem, porém, dessa vez, cifrada.

Figura 3.3 – Idéia para a segurança do Protocolo LoRaWAN.



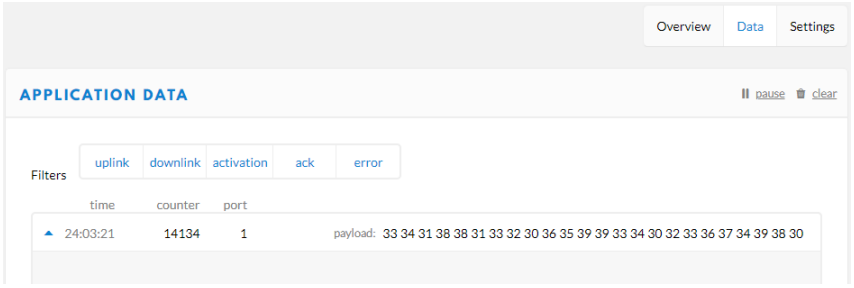
Fonte: Adaptado de The Things Network (2017).

3.2 APLICAÇÃO LORAWAN

Para mostrar como os dados estão armazenados no TTN, foi criado uma aplicação no mesmo, através do *console* que cada conta criada no site possui. Dentro da aplicação, foi registrado um dispositivo. Então, foi enviado uma mensagem, "*HelloWorld*", convertida em base Hexadecimal, para a aplicação, ou seja, cada caractere da mensagem foi transformado em sua representação hexadecimal, formando um número hexadecimal. Com isso foi possível ver que, no TTN, na aba *Data*, havia um *payload*, ou seja, uma mensagem. Nela, havia um número em base hexadecimal armazenado, onde cada byte estava separado por um espaço, como mostra a Figura 3.4. Utilizando desse número e de um conversor de números hexadecimais para caracteres, obteve-se, como resultado, um nú-

mero na base decimal. Então, convertendo este número para a base hexadecimal, foi possível perceber que era o mesmo que enviamos anteriormente, o qual era a mensagem "*HelloWorld*" convertida em base hexadecimal. Esse teste confirmou a teoria de que os dados armazenados estão descriptografados.

Figura 3.4 – Numero hexadecimal armazenado no TTN.



Fonte: Autor (2021).

Essa série de conversões é necessária, pois os bits da mensagem hexadecimal são enviados na transmissão. No TTN, os dados mostrados são representações hexadecimais de um número decimal, o qual é a representação decimal desses mesmos bits enviados. Então, para adquirir a mensagem original, é necessário apenas obter a representação hexadecimal desses bits, convertendo o numero decimal para hexadecimal.

3.3 ESCOLHA DO ALGORITMO

A fim de aplicar a criptografia da forma dita anteriormente, um algoritmo de criptografia simétrica seria uma escolha melhor, pois o CPD e a *FoxIoT*, que trabalham nisso para a universidade, possuem acesso a ambos os lados da técnica (encriptação e desencriptação) e esse tipo de criptografia é menos complexo do que, por exemplo, um algoritmo de criptografia assimétrica. Para um algoritmo de criptografia assimétrica funcionar, o dispositivo final precisaria ter conexão com a internet ou com o sistema de ambas entidades, conseguindo assim obter as chaves. Porém, pode haver situações em que o dispositivo final esteja em um lugar remoto, sem conexões, impossibilitando a troca de chaves e a desencriptação dos dados.

Presumivelmente, o algoritmo mais amplamente usado é o AES. Como o AES-128 é usado no protocolo LoRaWAN, e ele é um padrão para criptografia, pensou-se em utilizá-lo para essa camada de segurança. Porém, pelo fato de que esses dispositivos finais serem cada vez mais limitados em recursos, como uso da CPU, memória, energia, foi feita uma pesquisa sobre a eficiência energética de algoritmos de criptografia simétrica, o qual levou ao conhecimento da *Lightweight Cryptography* (Criptografia Leve). A principal característica dos algoritmos leves de criptografia é sua implementação, o qual utiliza poucos

recursos. Dentre eles, está o algoritmo utilizado para o propósito da camada adicional de segurança, o Algoritmo de Criptografia Leve (*Lightweight Encryption Algorithm* - LEA).

3.3.1 LEA - *Lightweight Encryption Algorithm*

O Algoritmo de Criptografia Leve (*Lightweight Encryption Algorithm* - LEA) foi desenvolvido em 2013. Ele é muito eficiente para estes dispositivos com poucos recursos, pois ele possui um código pequeno, consome pouca energia e consegue cifrar rapidamente com operações simples, como adição, rotação e XOR.

O algoritmo cifra blocos de 128 bits com diferentes tamanhos de chaves, sendo elas 128 bits (LEA-128), 192 bits (LEA-192) e 256 bits (LEA-256). O número de rodadas do algoritmo depende do tamanho da chave, sendo, respectivamente, 24, 28 e 32 rodadas. Nessas rodadas, ele consiste apenas em operações ARX (adição modular, rotação bit a bit e XOR bit a bit), utilizando palavras de 32 bits. O processo do algoritmo consiste no escalonamento da chave, encriptação e desencriptação. Para simplificação, o algoritmo será explicado utilizando a chave de 128 bits.

No escalonamento da chave, é gerado uma vetor de 6 chaves de rodadas (*RoundKeys*) de 32 bits cada, totalizando 192 bits, fazendo uso de uma tabela de constantes. Na figura 3.5 é mostrado como é gerada a tabela. A chave de 128 bits é divida em um vetor de palavras de 32 bits, e em cada uma, durante o numero de rodadas, são feito cálculos, até a geração do vetor de *RoundKeys*.

Figura 3.5 – Escalonamento da chave do algoritmo LEA

Escalonamento da Chave:

Constantes:

$$\begin{aligned} &\delta[0] = 0xc3efe9db, && \delta[1] = 0x44626b02,\\ &\delta[2] = 0x79e27c8a, && \delta[3] = 0x78df30ec,\\ &\delta[4] = 0x715ea49e, && \delta[5] = 0xc785da0a,\\ &\delta[6] = 0xe04ef22a, && \delta[7] = 0xe5c40957. \end{aligned}$$

Operações:

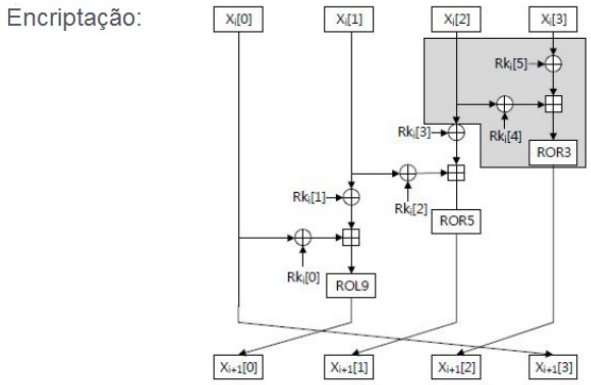
$$\begin{aligned} &T[0] \leftarrow \mathrm{ROL}_1(T[0] \boxplus \mathrm{ROL}_i(\delta[i \bmod 4])),\\ &T[1] \leftarrow \mathrm{ROL}_3(T[1] \boxplus \mathrm{ROL}_{i+1}(\delta[i \bmod 4])),\\ &T[2] \leftarrow \mathrm{ROL}_6(T[2] \boxplus \mathrm{ROL}_{i+2}(\delta[i \bmod 4])),\\ &T[3] \leftarrow \mathrm{ROL}_{11}(T[3] \boxplus \mathrm{ROL}_{i+3}(\delta[i \bmod 4])),\\ &RK_i \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1]). \end{aligned}$$

Fonte: Adaptado de Hong et al. (2013).

Na encriptação, o texto original de 128 bits é dividido em um vetor de 4 palavras de 32 bits. O resultado é outro vetor de 4 palavras de 32 bits. Durante a primeira rodada, para a primeira palavra alterada, são feitas duas operações XOR, uma da primeira palavra original com a primeira *RoundKey*, e outra da segunda palavra original com a segunda *RoundKey*. Os resultado são somados, e após, seus bits são deslocados para a esquerda.

Para a segunda palavra alterada, também são feitas duas operações XOR, porém uma da segunda palavra original com a terceira *RoundKey*, e outra da terceira palavra original com a quarta *RoundKey*. Os resultados são novamente somados, porém seus bits são deslocados para a direita. Semelhantemente acontece com a terceira palavra alterada. Já a quarta palavra alterada recebe apenas a primeira palavra original. Esse processo repete 24 vezes, ou seja, o número de rodadas, porém, durante as rodadas seguintes, as palavras alteradas são usadas como entradas, até as rodadas acabarem, resultando no texto cifrado.

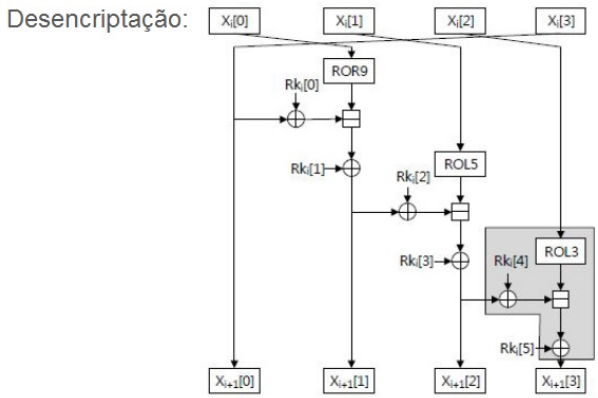
Figura 3.6 – Processo de encriptação do algoritmo LEA



Fonte: Adaptado de Seo et al. (2015).

Na desencriptação, o processo é basicamente o inverso do processo de encriptação. Nas imagens a seguir, é mostrado o processo visualmente.

Figura 3.7 – Processo de desencriptação do algoritmo LEA



Fonte: Adaptado de Seo et al. (2015).

Como o algoritmo foi projetado baseado nas recentes técnicas de segurança e no AES, ele possui uma margem de segurança suficiente contra muitos ataques existentes.

O LEA-128 foi escolhido entre outros algoritmos parecidos através das vantagens e diferenças descritas em Alshamsi e Barka (2017).

3.4 IMPLEMENTAÇÃO DO ALGORITMO

O algoritmo possui dois parâmetros de entrada, um texto de 128 bits, o qual é o bloco a ser criptografado ou descriptografado (*plaintText*) e outro texto de 128 bits, sendo a chave (*secretKey*). No algoritmo, o bloco a ser cifrado ou decifrado e a chave são transferidos para dois vetores de 16 bytes, chamados, respectivamente, de *data* e *key*. Ele possui também três funções, uma de escalonamento da chave (*keySchedule*), uma para encriptação (*leaEncryption*) e outra para desencriptação (*leaDecryption*).

Figura 3.8 – Pseudocódigo da função principal do algoritmo LEA-128

```
void leaEncryption(void *roundKey, void *data)
void leaDecryption(void *roundKey, void *data)
void keySchedule(void *key, void *roundKey)

#define R(v,n)(((v)>>(n))|((v)<<(32-(n))))
//Deslocamentos

main([plaintText, secretKey]) {
    data[16] <= plainText
    key[16] <= secretKey
// Vetor de 16 posições de 8 bits(byte), totalizando 128 bits

    roundKey[4]
// Vetor de 4 posições de 8 bytes, totalizando 128 bits

    keySchedule(key, roundKey)
    leaEncryption(roundKey, data)
    leaDecryption(roundKey, data)
}
```

Fonte: Autor (2021).

Então, é chamada a função de escalonamento da chave, *keySchedule*, enviando como parâmetro o endereço do vetor *key* e um vetor de 4 números inteiros, sem sinal, de 32 bits, chamado de *roundKey* (Ou seja, esse vetor contém 4 posições, no qual, cada uma dessas posições contém 4 bytes). Na função, são recebidos, como parâmetros, o ponteiro desses vetores, os quais são atribuídos para duas variáveis ponteiro de inteiros, sem sinal, de 32 bits, respectivamente, *funcKey* e *funcRoundKey*. Esse processo faz a conversão de vetores de 16 bytes, para vetores de 4 inteiros, sem sinal, de 32 bits. Também, dentro da função, é inicializado outro vetor de 4 inteiros, sem sinal, de 32 bits, contendo as constantes características do algoritmo, chamado de *constants*. Então, após as inicializações, a função executa o seu laço em 24 rodadas. Durante o laço, são executados várias atribuições e deslocamentos para *funcKey*, utilizando seus valores anteriores. O valor final do

processo é gravado no vetor *funcRoundKey* a partir dos resultados obtidos em *funcKey*. Após a execução da função *keySchedule*, o resultado dela está no vetor *roundKey*.

Figura 3.9 – Pseudocódigo da função de escalonamento da chave do algoritmo LEA-128

```
void keySchedule(void *key, void *roundKey) {
    *funcKey[4] <= key
    *funkRoundKey[4] <= roundKey
    constantes[4] <= {0xc3efe9db, 0x44626b02, 0x79e27c8a, 0x78df30ec};
// Constantes características do algoritmo LEA

    Para r = 0 até 23:
        temp <= constantes[r mod 3];
        constantes[r mod 3] <= temp << 4;
// Deslocamento de 4 bits para esquerda

        funcKey[0] <= (funcKey[0] + temp) << 1
// Deslocamento de 1 bit para esquerda

        funcKey[1] <= (funcKey[1] + (temp << 1)) << 3
// Primeiro, deslocamento de 1 bit para esquerda
// Após, deslocamento de 3 bits para esquerda

        funcKey[2] <= (funcKey[2] + (temp << 2)) << 6
// Primeiro, deslocamento de 2 bits para esquerda
// Após, deslocamento de 6 bits para esquerda

        funcKey[3] <= (funcKey[3] + (temp << 3)) << 11
// Primeiro, deslocamento de 3 bits para esquerda
// Após, deslocamento de 11 bits para esquerda

        funkRoundKey[0] <= funcKey[0];
        funkRoundKey[1] <= funcKey[1];
        funkRoundKey[2] <= funcKey[2];
        funkRoundKey[3] <= funcKey[3];

}
```

Fonte: Autor (2021).

Então, as funções de encriptação e desencriptação são chamadas. Para encriptar, é chamada a função (*leaEncryption*), enviando como parâmetro o endereço do vetor *roundKey* e o endereço do vetor *data*. Nessa função ocorre o mesmo processo que na função *keySchedule*, na qual são recebidos ponteiros dos vetores, e eles são atribuídos para duas variáveis ponteiro de inteiros, sem sinal, de 32 bits, respectivamente, *encKey* e *encData*. Posteriormente, a função executa um laço 24 vezes. Nesse laço, são executados operações XOR, adições, deslocamentos e atribuições, utilizando valores de rodadas anteriores, e gravando os atuais no vetor *encData*, exceto a última posição do vetor, que recebe diretamente o valor da primeira posição, do mesmo vetor, da rodada passada. No final da execução, o vetor *encData* contém o texto encriptado.

Figura 3.10 – Pseudocódigo da função de encriptação do algoritmo LEA-128

```
void leaEncryption(void *roundKey, void *data) {
    *encData <= data
    *encKey <= roundKey

    dataTemp0, dataTemp1, dataTemp2

    Para r = 0 até 23:
        dataTemp0 <= ((encData[0] xor encKey[0]) + (encData[1] xor
encKey[1])) << 9
// Deslocamento de 9 bits para esquerda

        dataTemp1 <= ((encData[1] xor encKey[2]) + (encData[2] xor
encKey[1])) >> 5
// Deslocamento de 5 bits para direita

        dataTemp2 <= ((encData[2] xor encKey[3]) + (encData[3] xor
encKey[1])) >> 3
// Deslocamento de 3 bits para direita

        encData[3] <= encData[0]
        encData[2] <= dataTemp2
        encData[1] <= dataTemp1
        encData[0] <= dataTemp0

}
```

Fonte: Autor (2021).

Para desencriptar, é chamada a função (*leaDecryption*), enviando, também, como parâmetro o endereço do vetor *roundKey* e o endereço do vetor *data*. Nessa função ocorre o mesmo processo que em ambas funções *keySchedule* e *leaEncryption*, no qual são recebidos ponteiros dos vetores, e eles são atribuídos para duas variáveis ponteiro de inteiros, sem sinal, de 32 bits, respectivamente, *decKey* e *decData*. Assim, a função avança para a execução do laço, repetindo-a 24 vezes. Nesse laço, são executadas operações XOR, subtrações, deslocamentos e atribuições, utilizando valores de rodadas anteriores e valores calculados na rodada atual. Os valores calculados são gravados no vetor *decData*, exceto na primeira posição, onde é gravado direto o valor da última posição, do mesmo vetor, da rodada passada. Então, finalmente, o vetor *decData* contém o texto desencriptado.

Figura 3.11 – Pseudocódigo da função de desencriptação do algoritmo LEA-128

```
void leaDecryption(void *roundKey, void *data) {
    *decData[4] <= data;
    *decKey[4] <= roundKey;

    dataTemp1, dataTemp2, dataTemp3;

    Para r = 0 até 23:
        dataTemp1 <= ((decData[0] >> 9) - (decData[3] xor decKey[0])) xor
decKey[1];
// Deslocamento de 9 bits para direita
        dataTemp2 <= ((decData[1] << 5) - (dataTemp1 xor decKey[2])) xor
decKey[1];
// Deslocamento de 5 bits para esquerda
        dataTemp3 <= ((decData[2] << 3) - (dataTemp2 xor decKey[3])) xor
decKey[1];
// Deslocamento de 3 bits para esquerda

        decData[0] <= decData[3];
        decData[1] <= dataTemp1;
        decData[2] <= dataTemp2;
        decData[3] <= dataTemp3;

}
```

Fonte: Autor (2021).

Como é passado para as funções os endereços dos vetores, e gravados nas variáveis ponteiro, as alterações ocorrem direto nos endereços de memória onde estão guardados os valores. Por isso, todas as 3 funções não retornam valores, pois elas os alteram por referência.

Após essa implementação, decidiu-se testar o mesmo, encriptando a mensagem transmitida anteriormente para a aplicação do TTN, o "*HelloWorld*". Dessa forma, a mensagem cifrada foi transformada em um número na base hexadecimal e a enviada novamente. Em seguida, foi obtido seu valor armazenado e feita a mesma sequência de conversões, como anteriormente.

3.5 ANÁLISE DO ALGORITMO

Para obter uma visão real de que esse algoritmo foi uma escolha correta, foram feitas algumas análises em cima dele. A primeira foi a partir da comparação com o algoritmo AES-128, devido a ele ser um padrão para criptografia e por ser cogitado o seu uso, como visto anteriormente. Depois, foi analisada sua força de encriptação, ou seja, robustez. Isso foi feito através da execução de um algoritmo de força bruta e da observação do efeito avalanche.

3.5.1 Comparação do Algoritmo

Utilizando os dois algoritmos, o LEA-128 e o AES-128, foram comparados dois fatores, a complexidade temporal (pelo tempo de execução), pois o processo de encriptação deve ser executado rapidamente, sem atrasar a comunicação, e a complexidade espacial (pela ocupação da memória), pois o dispositivo final possui recursos limitados e o algoritmo deve utilizar o quanto menos, deixando para outras tarefas. Pelo fato de que a encriptação ocorre entre a aquisição dos dados pelos sensores e transmissão pelo módulo LoRa, e a desencriptação ocorre apenas no momento de retirar os dados do servidor de aplicação, essas comparações foram feitas levando em consideração apenas o processo de encriptação de ambos algoritmos.

3.5.2 Robustez do Algoritmo

Uma maneira de observar a robustez do algoritmo é a partir do tempo médio que, teoricamente, levaria para testar todas chaves possíveis em uma mensagem cifrada. Dessa forma, a chave correta seria a usada no momento em que a mensagem fosse decifrada. Esse processo de testar todas combinações de chaves possíveis é chamado de ataque de força bruta.

Em vista disso, foi criado um algoritmo de força bruta para obter o tempo estimado que o mesmo iria demorar para descobrir uma chave do algoritmo LEA-128. Nele, foi encriptado um texto de 128 bits, utilizando uma chave de 128 bits, juntamente com as funções do LEA-128, implementadas anteriormente. O ponto principal do algoritmo é a sequência de 16 laços que ele possui. Como cada caractere é um byte (ou seja, uma sequência de 8 bits), quando ele executa 256 vezes, significa que, naquele caractere, são testados todas combinações de bits. Então, o próximo caractere adiciona uma unidade, o que muda sua sequência de bits, e os caracteres anteriores são resetados, para todas suas associações serem testadas com essa nova sequência. E isso se dá para todos os 16 bytes que compõem a chave. Para cada sequência de bits diferente, é chamada a função de escalonamento da chave, para formar uma *RoundKey*, e a função de desencriptação, enviando o texto cifrado e a *RoundKey*. Então, é feita a comparação do texto decifrado, resultado da chave gerada, com o texto antes de ser cifrado.

Outra maneira de observar a robustez do algoritmo é analisar se uma pequena modificação na entrada do mesmo, faz com que sua saída seja significantemente diferente. Esse é o efeito avalanche, em que uma modificação de um bit na chave ou na mensagem do algoritmo deve alterar drásticamente os bits de saída do mesmo. Ela pode ser calculada a partir da equação 3.1. Um bom algoritmo possui um alto efeito avalanche, fazendo com que tenha uma randomização forte o suficiente para dificultar previsões sobre a entrada,

apenas com a saída (RAMANUJAM; KARUPPIAH, 2011).

$$EfeitoAvalanche = \frac{N_{BitsVariados}}{N_{Bits}} * 100 \quad (3.1)$$

Para isso, primeiramente, foram pegos vários pares de mensagens de entrada, o qual se diferiam em apenas um bit. Então, foi feito com que o algoritmo executasse duas vezes a criptografia, uma com cada entrada, e, no final, comparasse os Bits de ambas saídas, gravando quantos eram diferentes. Isso foi efetuado com todos os pares e obtido a média de quantos bits variaram em cada. Da mesma forma, foram pegos vários pares de chaves, também se diferindo em apenas um bit, e executou-se o mesmo processo.

3.6 SELEÇÃO DE *HARDWARES* E *SOFTWARES*

Toda implementação e execuções dos algoritmos foram feitos em um processador Intel Core i7 7500u, com 2.7GHz de frequência, utilizando o Subsistema Linux para Windows (*Windows Subsystem for Linux*- WSL), no sistema operacional Windows 10 Home. O WSL é um módulo do Windows 10, que disponibiliza um ambiente do sistema operacional Linux, incluindo ferramentas de linha de comando, utilitários e aplicativos, sem a necessidade de máquinas virtuais.

Para obter os tempos de execução dos algoritmos, foi utilizada a função *clockgettime*, da biblioteca *time.h*, da linguagem C. Com ela, é possível obter o tempo antes da execução da função e após a execução, e fazer os cálculos. A execução dessa função foi realizada no mesmo hardware e software da implementação, citados anteriormente

Para obter a ocupação da memória dos algoritmos, foram utilizados dois programas, o AWE (*Arduino Web Editor*) e o *TinkerCad*. O primeiro é uma IDE (*Integrated Development Environment*), o qual permite compilar códigos em diferentes microcontroladores, retornando arquivos com as informações do código. O segundo é um simulador, onde é possível construir e simular um circuito utilizando microcontroladores, sem a necessidade de um físico. Para ambos, foi escolhido o microcontrolador *Arduíno Uno*, com frequência de 16MHz, memória *Flash* de 32KB e memória SRAM (*Static Random-Access Memory*) de 2KB. O segundo foi utilizado pela fato do primeiro precisar de um hardware físico para utilizar a porta serial, que serve para mostrar algumas informações desejadas em tela.

Então, para obter a quantidade de memória que os algoritmos ocupavam, era necessário encontrar os valores da memória Flash, referente as instruções, e da memória SRAM, referente aos dados. A execução dos mesmos no AWE já retornava a ocupação da memória *Flash*. Para a outra parte da memória, foi necessário executar os algoritmos no *TinkerCad* e utilizar a função *freeMemory*, da biblioteca *MemoryFree.h*. Essa função retorna a quantidade de memória livre na SRAM, sendo necessário apenas pegar o máximo

dela (2KB) e subtrair pelo resultado.

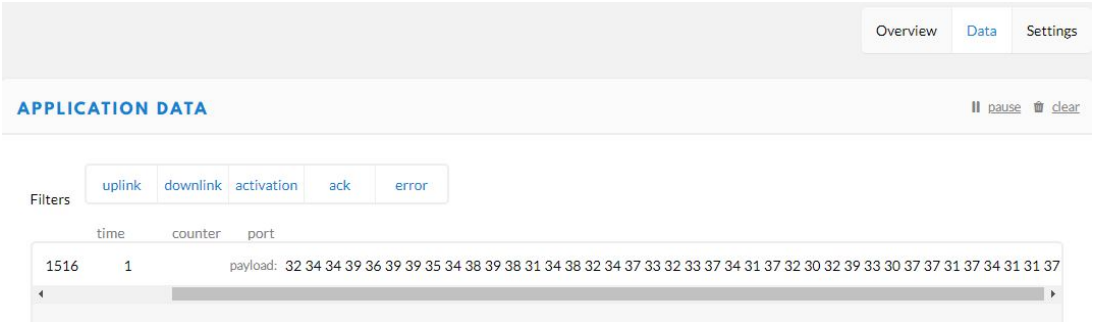
Semelhante aos algoritmos anteriores, para estimar o tempo de execução do algoritmo de força bruta, foram feitos os mesmos passos, utilizado a função *clockgettime*, da biblioteca *time.h*, da linguagem C.

Para a comunicação com a rede LoRa da UFSM, foi utilizado um Raspberry Pi com um modulo *Dragino LoRa/GPS HAT*. Ele possui um *script* em linguagem Python, o qual configura o mesmo a uma frequência compatível com a do *gateway* instalado na UFSM, e se une a ele pelo modo ABP. Foram necessários executar alguns passos para a instalação e mudar algumas variáveis, como as chaves e identificadores.

4 RESULTADOS

A partir da confirmação da teoria, é possível afirmar que, caso houver alguma falha no sistema do TTN, e os dados armazenados sejam vazados, eles podem ser lidos por qualquer um que conseguir acesso a eles, pois estão desprotegidos. A Figura 4.1 mostra o resultado de como a mensagem, *"HelloWorld"*, é armazenada no TTN, caso ela for criptografada com o algoritmo LEA-128 e com a chave *"chavesecreta"*, antes de ser enviada. Pegando a mensagem e fazendo a mesma sequência de conversões feitas anteriormente, o resultado é um texto ininterpretável, o qual inclui caracteres que, dependendo dos programas que processam e mostram as palavras, não são decodificados, e que não contém nenhuma informação sobre a mensagem real.

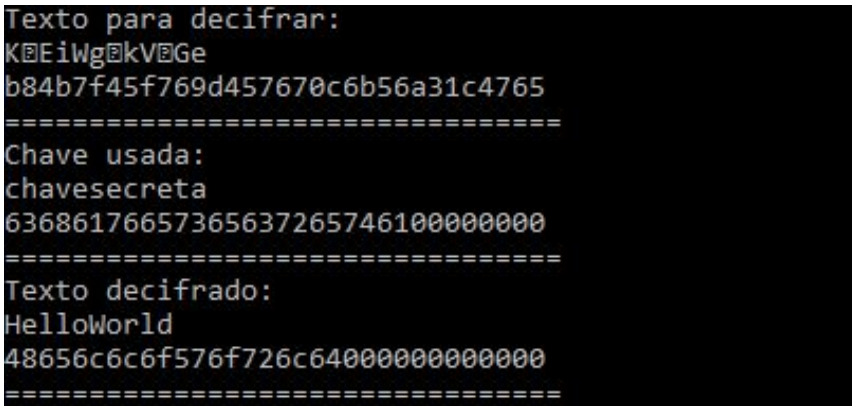
Figura 4.1 – Número hexadecimal cifrado armazenado no TTN



Fonte: Autor (2021).

Copiando o número hexadecimal do texto cifrado, e aplicando a descriptografia do algoritmo LEA-128 (juntamente com a mesma chave usada na criptografia, a *"chavesecreta"*), o resultado é a mensagem original, como mostra a Figura 4.2.

Figura 4.2 – Processo de desencriptação do texto



Fonte: Autor (2021).

Em relação às comparações feitas entre os algoritmos, os testes feitos, mesmo

com algumas interferências, os quais diminuíram sua precisão, mostraram resultados, para o processo de encriptação, melhores para o algoritmo LEA-128 do que para o algoritmo AES-128.

A Tabela 4.1 mostra os resultados da obtenção dos tempo de execução dos algoritmos. Devido ao hardware utilizado para obter os tempos de execução estar sempre executando outros processos, em vários momentos ele pode ter interrompido a execução do processo de criptografia, executado algum outro de maior prioridade e, em seguida, retornado, causando variações nos tempos obtidos. Sem o controle desses processos, não é possível criar um ambiente de execução igual para os dois algoritmos, ou seja, um ambiente em que a execução de ambos algoritmos tivessem o mesmo tempo de interrupção e processos prioritários.

Por isso, os algoritmos foram executados mil vezes, obtendo-se o menor, o maior e a média dos valores, mostrados na tabela 4.1. Nela, é possível observar que os tempos de execução do algoritmo LEA-128 são menores que o AES-128. Porém, o menor e a média do T_{Exec} do AES-128 são menores que o maior T_{Exec} do LEA-128. Isso demonstra as interferências que ocorre devido a utilização desse *hardware*, porém, em um contexto geral, o algoritmo LEA-128 possui um T_{Exec} menor, executando em torno de 30% mais rápido.

Tabela 4.1 – Tempo de execução do processo de encriptação dos algoritmos

Algoritmo	Menor $T_{Exec}(\mu s)$	Maior $T_{Exec}(\mu s)$	Média $T_{Exec}(\mu s)$
LEA-128	1,5	2,6	1,6
AES-128	2,1	3,3	2,2

Fonte: Autor (2021).

Como obter os valores exatos do tempo de execução dos algoritmos através desse método está sujeito a erros, os dados adquiridos nessa aquisição foram estimativas dos tempos de execução no hardware utilizado, e a observação feita pode servir como um reflexo, quando utilizados outros hardwares, ou seja, embora os algoritmos iriam levar mais tempo para serem executados em hardwares menos potentes, como em controladores, a observação de que o LEA-128 possui um T_{Exec} menor pode ser levada em consideração.

Também, a Tabela 4.2 mostra os resultados do espaço que os algoritmos ocupam na memória. É possível perceber que em ambas memórias, seja a que contém as instruções e a que contem os dados, o algoritmo LEA-128 ocupa um número menor de bytes. Uma observação a ser feita é que esse número de bytes da memória flash contém, também, os bytes que o *bootloader* do controlador ocupa, porém esse valor é igual para as execuções dos dois algoritmos.

Quanto à análise da robustez, obteve-se, primeiramente, com a execução do algoritmo de força bruta, através de varias repetições, a informação de que o teste de cada chave, sendo ele escalonamento, desencriptação da mensagem cifrada e comparação da

Tabela 4.2 – Espaço de memória ocupado pelo processo de encriptação dos algoritmos

Algoritmo	Flash (B)	SRAM (B)
LEA-128	1565	517
AES-128	3158	731

Fonte: Autor (2021).

mensagem decifrada com a mensagem original, resultou em média um tempo de $2,09$ microssegundos, com desvio padrão de $0,15$. Para o melhor caso, seria necessário executar apenas uma vez. Já para o pior caso, seria necessário a execução de todas as tentativas, ou seja, como cada bit pode ser dois valores diferentes (0 ou 1), 2^{128} vezes. Pegando a média de tentativas, temos 2^{127} vezes. Multiplicando esses valores, obteve-se o tempo estimado de $3,55595073432380694319226464766 2 * 10^{38}$ microssegundos. Convertendo esse tempo para anos, utilizando o hardware citado, demoraria um tempo estimado de $1,13*10^{25}$ anos, em média, para encontrar a chave e quebrar o algoritmo.

Após, com a execução do algoritmo, utilizando duzentos pares de mensagens de entrada que se diferiam de apenas um bit e mantendo a chave sem alterações, obteve-se que a média da variação dos bits da mensagem criptografada de saída era de 64 bits, com desvio padrão de 6 bits. Também, com a execução utilizando duzentos pares de chaves que se diferiam de apenas um bit, e mantendo a mensagem de entrada sem alterações, foi obtida uma média de 65 bits variados, com desvio padrão de 5 bits. Isso significa que, em média metade dos bits da mensagem criptografada de saída alteram quando ocorre uma pequena mudança, seja na mensagem de entrada ou na chave do algoritmo. Utilizando a equação 3.1, o algoritmo possui uma variação na sua saída em torno de 50% dos bits, o qual é um grau significativo do efeito de avalanche, dificultando o descobrimento da chave ou da mensagem.

5 CONCLUSÃO

Com as análises feitas e obtidas, é perceptível que, não só os dados da UFSM, mas de qualquer pessoa ou instituição, que utilizam apenas um servidor de rede e aplicação de uso publico, como o TTN, e que não possuem um mecanismo de segurança adicional, podem estar sujeitos a atacantes, caso haja alguma falha no sistema e os dados sejam vazados. Dessa forma, esses atacantes podem se beneficiar com os dados roubados, causando danos prejudiciais aos proprietários. Devido a isso, a solução apresentada neste trabalho, de adicionar uma camada de segurança para os dados através do uso de um algoritmo de criptografia, encriptando-os antes da transmissão, e desencriptando-os após a retirada dos mesmo do servidor, atende ao objetivo de proteger as informações da universidade que trafegam na rede.

Também, como os dispositivos que transmitem os dados através da rede estão cada vez mais em lugares remotos e de difícil acesso, seria benéfico que sua bateria durasse o maior tempo possível. Por isso, um algoritmo que gaste a menor quantidade de recursos possível para encriptar uma mensagem, e que tenha o mesmo nível de segurança dos algoritmos padrões utilizados hoje em dia, é ideal.

Embora essa aquisição dos dados dos testes de tempo de execução estarem sujeitos a erros, principalmente devido ao hardware utilizado, a estimativa adquirida mostrou que o algoritmo LEA-128 executa mais rápido que o algoritmo AES-128. Também, os testes de memória utilizando o compilador e o simulador, mostraram que o AES-128 ocupa menos recursos. De acordo com isso, o algoritmo selecionado nesse trabalho se torna uma escolha melhor para essa ocasião. Também, ele possui resistência contra ataques de força bruta, demorando anos para ser quebrado, e um bom efeito avalanche, variando bastante sua saída ao alterar minimamente a entrada, dificultando previsões. Por isso, este algoritmo também atende ao objetivo do trabalho, possuindo características que são necessárias à rede.

Com a implementação dessa solução, os serviços desenvolvidos e prestados por entidades, que utilizam uma rede publica LoRa, iriam agregar um diferencial em sua comercialização, por eles terem um ponto de segurança adicional.

Com o futuro, a rede LoRa e o protocolo LoRaWAN tendem muito a evoluir, podendo ser implementado uma solução, ou que o próprio TTN tenha novas regras que possam evitar que esse tipo de risco aconteça. Também, é possível que apareçam cada vez mais algoritmos leves, que teriam muito mais vantagens e deixariam os atuais obsoletos.

Para trabalhos futuros, poderia ocorrer a aplicação do algoritmo nos dispositivos finais, levando a ideia da implementação da camada adicional de segurança para a prática. Também, utilizar hardwares específicos para obter dados mais precisos de tempos de execução e ocupação de memória. Além disso, analisando o dispositivo final, pode-se

descobrir o processo que limita o desempenho do mesmo, e melhorar alguma função do algoritmo de criptografia, fazendo com que esse tempo de espera não seja desperdiçado.

REFERÊNCIAS BIBLIOGRÁFICAS

Alper Yegin. **LoRaWAN Security**: A technical overview of lorawan security from standard, implementation, and deployment perspective. Actility, 2019. Acesso em 29 nov. 2019. Disponível em: <https://pt.slideshare.net/Actility/lorawan-security-webinar>.

ALSHAMSI, A. Z.; BARKA, E. S. Implementation of energy efficient/lightweight encryption algorithm for wireless body area networks. In: **2017 International Conference on Informatics, Health & Technology (ICIHT)**. [S.l.]: IEEE, 2017. p. 1–7.

ARAS, E. et al. Exploring the security vulnerabilities of lora. In: **2017 3rd IEEE International Conference on Cybernetics (CYBCONF)**. [S.l.]: IEEE, 2017. p. 1–6.

DAYA, B. Network security: History, importance, and future. **University of Florida Department of Electrical and Computer Engineering**, v. 4, p. 13, 2013.

DEVALAL, S.; KARTHIKEYAN, A. Lora technology - an overview. In: **2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)**. [S.l.]: IEEE, 2018. p. 284–290.

Gemalto and Actility and Semtech. **LoRaWAN Security**: Full end to end encryption for iot application providers. LoRa Alliance, 2017. Acesso em 29 nov. 2019. Disponível em: <https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_security_whitepaper.pdf>.

HILL, P. C. J. Vigenère through shannon to planck a short history of electronic cryptographic systems. In: **2008 IEEE History of Telecommunications Conference**. [S.l.]: IEEE, 2008. p. 41–46.

HONG, D. et al. Lea: A 128-bit block cipher for fast encryption on common processors. In: **International Workshop on Information Security Applications**. [S.l.]: Springer, 2013. p. 3–27.

KEHE, W. et al. Security model based on network business security. In: **2009 International Conference on Computer Technology and Development**. [S.l.]: IEEE, 2009. p. 577–580.

LoRa Alliance Technical Marketing Workgroup. **LoRaWAN What is it?**: A technical overview of lora and lorawan. LoRa Alliance, 2015. Acesso em 29 nov. 2019. Disponível em: <https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf>.

Mathieu Monneret. **LoRaWAN Adaptive Data Rate**. The Things Network, 2019. Acesso em 29 nov. 2019. Disponível em: <https://www.thethingsnetwork.org/docs/lorawan/adaptive-data-rate.html>.

NAOUI, S.; ELHDHILI, M. E.; SAIDANE, L. A. Enhancing the security of the iot lorawan architecture. In: **2016 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)**. [S.l.]: IEEE, 2016. p. 1–7.

NEIS, M. P. **Implementação de Rede LoRa no sistema de transporte intracampus da UFSM.** 2017. 63 p. Monografia (Trabalho de Conclusão de Curso) — Curso de Graduação em Engenharia de Computação, Universidade Federal de Santa Maria, Santa Maria, 2017.

RAMANUJAM, S.; KARUPPIAH, M. Designing an algorithm with high avalanche effect. In: **IJCSNS International Journal of Computer Science and Network Security**. [S.l.: s.n.], 2011. p. 106–111.

Sakshama Ghoslya. **All About LoRa and LoRaWAN**: Lora: Symbol generation. 2017. Acesso em 25 fev. 2020. Disponível em: <https://www.sghoslya.com/p/lora-is-chirp-spread-spectrum.html>.

SEO, H. et al. Compact implementations of lea block cipher for low-end microprocessors. In: **International Workshop on Information Security Applications**. [S.l.]: Springer, 2015. p. 28–40.

STALLINGS, W. **Criptografia e Segurança de Redes: Princípios e Práticas Sexta Edição**. São Paulo: Pearson Education, 2014. 541 p.

The Things Network. **LoRaWAN Security - The Things Network Webinar by Johan Stokking**. YouTube, 2017. Acesso em 06 set. 2019. Disponível em: <https://www.youtube.com/watch?v=Nu_yZelDMZI>.

TOMASIN, S.; ZULIAN, S.; VANGELISTA, L. Security analysis of lorawan join procedure for internet of things networks. In: **2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)**. [S.l.]: IEEE, 2017. p. 1–6.

Wienke Giezeman. **The Things Network Manifesto**. The Things Network, 2017. Acesso em 18 jan. 2021. Disponível em: <https://github.com/TheThingsNetwork/Manifesto>.

Willis H. Ware. **Security Controls for Computer Systems**: Report of defense science board task force on computer security. RAND CORP SANTA MONICA CA, 1979. Acesso em 20 out. 2019. Disponível em: <https://www.rand.org/pubs/reports/R609-1/index2.html>.

YANG, X. et al. Security vulnerabilities in lorawan. In: **2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)**. [S.l.]: IEEE, 2018. p. 129–140.

YU, L. et al. Aes design improvements towards information security considering scan attack. In: **2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)**. [S.l.]: IEEE, 2018. p. 322–326.

ZORBAS, D. et al. Improving lora network capacity using multiple spreading factor configurations. In: **2018 25th International Conference on Telecommunications (ICT)**. [S.l.]: IEEE, 2018. p. 516–520.