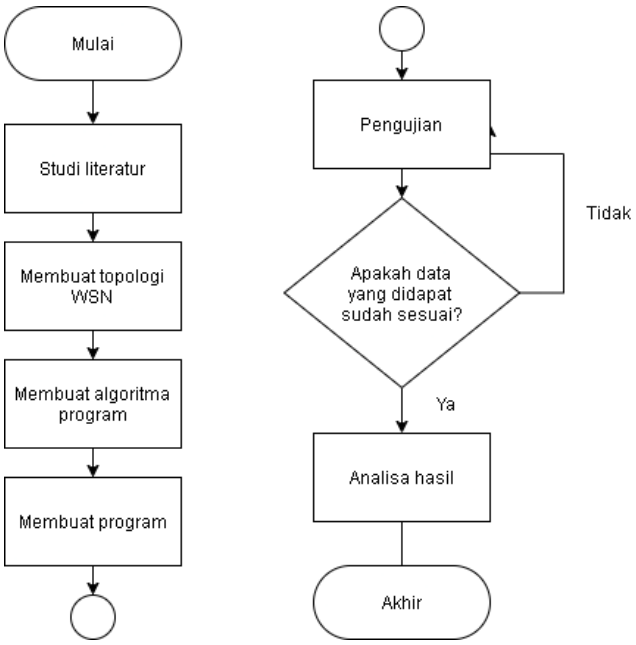
BAB 3
METODE PENELITIAN

3.1 ALAT YANG DIGUNAKAN

Penelitian ini menggunakan Raspberry Pi, dan modul Dragino LoRa/HAT untuk alat bantu koneksi ke jaringan WSN. Penelitian ini menggunakan program *sniffer* untuk menganalisa keamanan data, serta menggunakan protokol SSH untuk mengendalikan perangkat Raspberry Pi.

3.2 ALUR PENELITIAN

Penelitian ini memiliki beberapa tahap yang harus dilalui, tahap-tahap tersebut membentuk sebuah diagram alur yang ditunjukkan pada gambar 3.1.



Gambar 3.1 Alur proses penelitian

Pada penelitian ini dilakukan studi literatur terkait topik yang diambil. Membandingkan metode yang pernah digunakan untuk penelitian dengan metode yang akan digunakan pada penelitian ini. Kemudian dilanjutkan dengan membuat topologi jaringan WSN yang akan digunakan dan membuat algoritma program

berdasarkan topologi yang telah dibuat. Setelah itu program dibuat sesuai algoritma yang telah ditentukan dan melakukan pengujian terhadap program yang telah dibuat apakah bisa digunakan dan mendapatkan data sesuai dengan parameter pengujian. Jika belum maka akan dilakukan pengecekan algoritma program dan pengujian ulang, namun jika sudah sesuai maka dilanjutkan ke proses analisa kemudian selesai.

Topologi WSN yang digunakan pada penelitian ini dapat dilihat pada gambar 3.2. Penelitian hanya dilakukan pada pengiriman data *sensing* dari *sensor node* ke *server* secara langsung.

Gambar 3.2 Topologi WSN yang digunakan

Kedua perangkat pada jaringan WSN akan terhubung secara *point-to-point* menggunakan modul Dragino LoRa/HAT. Dimana frekuensi yang digunakan 923 MHz sesuai regulasi kominfo. Kemudian data yang dikirimkan merupakan data *sensing* dari *sensor node* menggunakan sensor DHT11 secara *real-time*. Penelitian ini akan berfokus pada pengujian dan analisa keamanan data *sensing* saat dikirimkan dari *sensor node* ke *server node*, paramater QoS *delay* (*latency*) dan *packet loss*. Sehingga dari penelitian ini dapat menghasilkan informasi dari sisi keamanan dan kehandalan program yang dibuat.

3.3 SKENARIO PENELITIAN

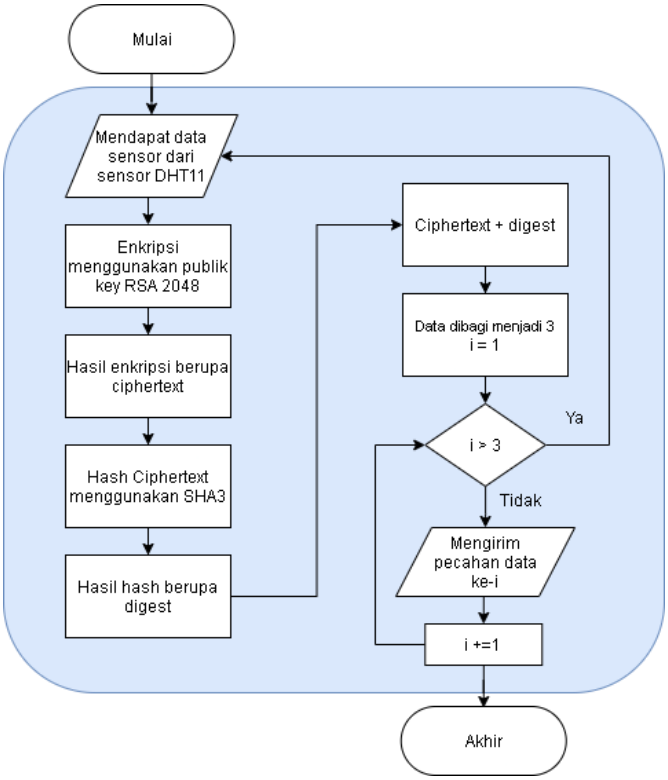
Pada skenario penelitian bagian *sensor node* dan *server node* menggunakan Raspberry Pi 3B. Topologi yang digunakan pada skenario penelitian ini ditunjukkan oleh gambar 3.3.

Gambar 3.3 Skenario pengujian

Modul Dragino LoRa/HAT pada Raspberry Pi yang berperan sebagai *sensor node* menggunakan mode *transmitter* mengirimkan data dari sensor DHT11 ke Raspberry Pi yang berperan sebagai *server node* dengan modul Dragino LoRa/HAT menggunakan mode *receiver*. Pada proses inilah dilakukan analisis keamanan dan QoS terhadap data yang dikirimkan. Skenario pengujian yang dilakukan 3 kali berdasarkan jarak antara *sensor node* dan *server node* yaitu jarak 100, 200, dan 300 meter dengan kondisi *line of sight*.

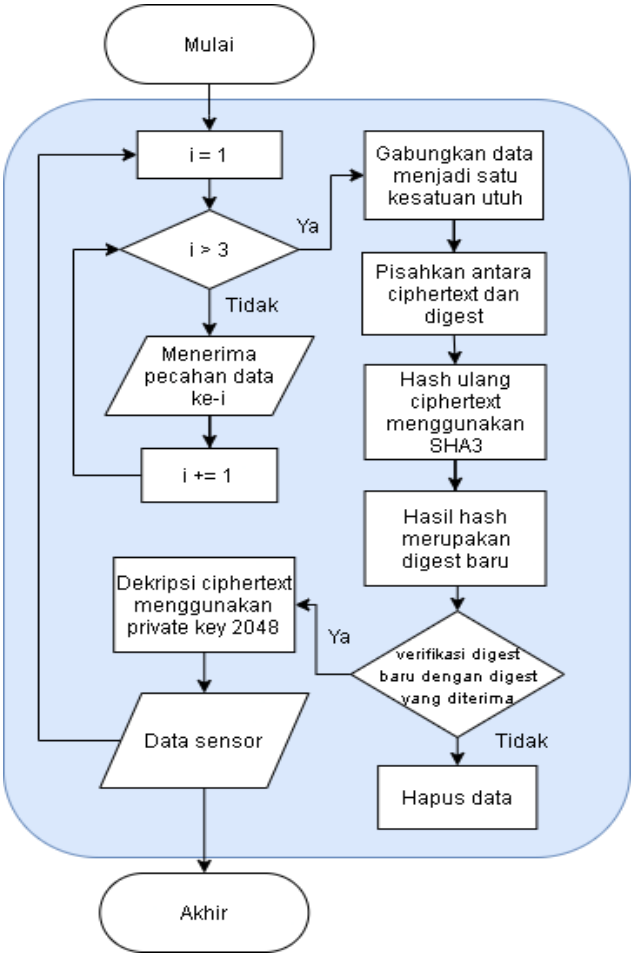
3.4 ALUR PEMROGRAMAN

Program yang dibuat pada penelitian ini memiliki 2 jenis, yaitu program yang berjalan pada sisi pengirim (*transmitter*) dan sisi penerima (*receiver*). Alur algoritma progam yang berjalan pada sisi pengirim dapat dilihat pada gambar 3.4, sedangkan untuk program yang berjalan pada sisi penerima dapat dilihat pada gambar 3.5.



Gambar 3.4 Alur algoritma program pengirim

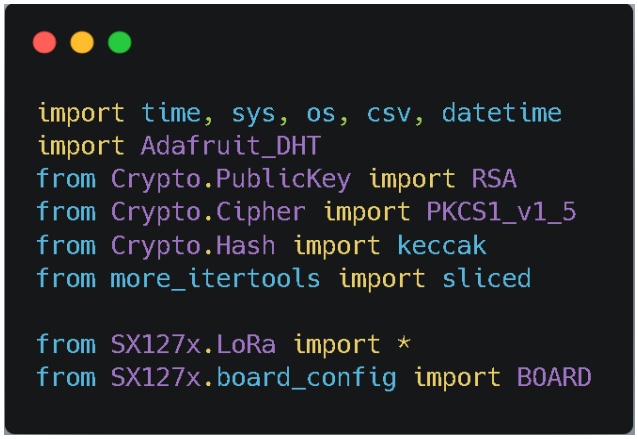
Pada gambar 3.4 ditunjukkan alur proses kinerja program pengirim bahwa data yang dikirimkan merupakan hasil sensor DHT11 yang kemudian dienkripsi menggunakan *public key* algoritma RSA 2048 dan menghasilkan data yang terenkripsi berupa *ciphertext*. Kemudian *ciphertext* tersebut akan dilakukan *hashing* menggunakan algoritma SHA-3 dan menghasilkan sebuah *digest*. Kemudian *ciphertext* dan *digest* digabungkan menjadi satu data yang utuh baru kemudian dipecah menjadi 3 bagian. Selanjutnya pada proses transmisi data yang sudah dipecah tadi akan dikirim satu per satu ke *receiver* baru kemudian diulang lagi ke langkah awal untuk mengambil data lagi dari sensor DHT11.



Gambar 3.5 Alur algoritma program penerima

Pada gambar 3.5 menunjukkan alur proses kerja program penerima dimana program akan menerima pecahan program sebanyak 3 kali baru kemudian menggabungkannya lagi menjadi satu kesatuan. Lalu setelah itu pisahkan *ciphertext* dengan *digest*, setelah dipisah kemudian *chipertext* di-*hash* ulang untuk mendapatkan *digest* baru. Lalu *digest* baru tersebut akan diverifikasi terlebih dahulu dengan cara membandingkannya dengan *digest* yang diterima, jika sama maka akan dilanjutkan proses dekripsi menggunakan *private key* algoritma RSA 2048. Namun jika tidak sama, maka data *ciphertext* tersebut akan dihapus. Proses verifikasi ini sangat penting untuk menjamin data yang diterima merupakan data asli. Karena ada kemungkinan ketika data dikirimkan diubah oleh pihak yang tidak berwenang yang disebabkan oleh celah yang terdapat pada proses transmisi, kerusakan pada proses enkripsi, dan lainnya.

Pada program yang dibuat menggunakan python versi 3.7.3 dengan menggunakan *library pycryptodome* untuk melakukan pengamanan data. Fungsi-fungsi yang diimpor dari *library* tersebut untuk program pengirim diperlihatkan pada gambar 3.6 sedangkan untuk program penerima diperlihatkan pada gambar 3.7.



```
import time, sys, os, csv, datetime
import Adafruit_DHT
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from Crypto.Hash import keccak
from more_itertools import sliced

from SX127x.LoRa import *
from SX127x.board_config import BOARD
```

Gambar 3.6 *Import* fungsi dari *library* pada program pengirim

Gambar 3.6 menunjukkan perintah mengimpor *library* pada program pengirim yang berfungsi untuk mengimpor beberapa fungsi dari *library pycryptodome* untuk penggunaan algoritma kriptografi RSA 2048, penggunaan kunci publik, dan penggunaan algoritma *hash* SHA-3 (*Keccak*). Selain itu juga ada *library* LoRa, sensor DHT dan *library* pendukung lainnya.

```
import time, sys, os, csv, datetime
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from Crypto.Hash import keccak
from Crypto import Random

from SX127x.LoRa import *
from SX127x.board_config import BOARD
```

Gambar 3.7 *Import* fungsi dari *library* pada program penerima

Gambar 3.7 menunjukkan perintah mengimpor *library* pada program penerima, *library* yang digunakan hampir sama kecuali *library* sensor DHT yang tidak diimpor.

3.5 PROSES ENKRIPSI

Proses enkripsi yang dilakukan sesuai dengan alur algoritma pemograman pada program pengirim yang ditunjukkan pada gambar 3.4. Pertama, program akan memanggil fungsi get_sensor() untuk mengambil data dari sensor DHT11 (*plaintext*). Kemudian program memanggil fungsi enc_rsa() untuk melakukan enkripsi *plaintext* menggunakan *public* key untuk mendapatakan *ciphertext*. Lalu, *ciphertext* di*hash* menggunakan SHA-3 untuk mendapatkan *digest*. Kemudian *ciphertext* dan *digest* digabungkan lalu dibagi menjadi 3 bagian. Lalu kemudian data yang dipecah menjadi 3 tersebut dikirim 1 per 1 ke penerima. Proses enkripsi dilakukan program pengirim bisa dilihat pada gambar di bawah ini.

```
if (self.tx_counter % 3 == 0):
        self.sens = get_sensor()
        print(self.sens.decode())
        self.s_time = time.time()
        self.data = list(enc_rsa(self.sens,pub))

self.write_payload(list(self.data[self.tx_counter%3]))
time.sleep(1)
self.tx_counter += 1
```

(a)

```
def get_sensor():
    sensor = Adafruit_DHT.DHT11
    pin = 27
    humidity, temperature = Adafruit_DHT.read_retry(sensor,pin)
    if humidity is not None and temperature is not None:
        return "Temp={0:0.1f}* Humidity={1:0.1f}%".format(temperature,humidity).encode()
    else:
        print('Failed to get reading. Try again!')
        sys.exit(1)
```

(b)

```
def enc_rsa(message,pub):
    rsa = PKCS1_v1_5.new(pub)
    ciphertext = rsa.encrypt(message)

    sha3 = keccak.new(digest_bits=256)
    sha3.update(ciphertext)
    return list(sliced(ciphertext+sha3.digest(),100))
```

(c)

Gambar 3.8 Proses enkripsi pada program pengirim

a) Proses enkripsi dan pengiriman data

b) Fungsi get_sensor()

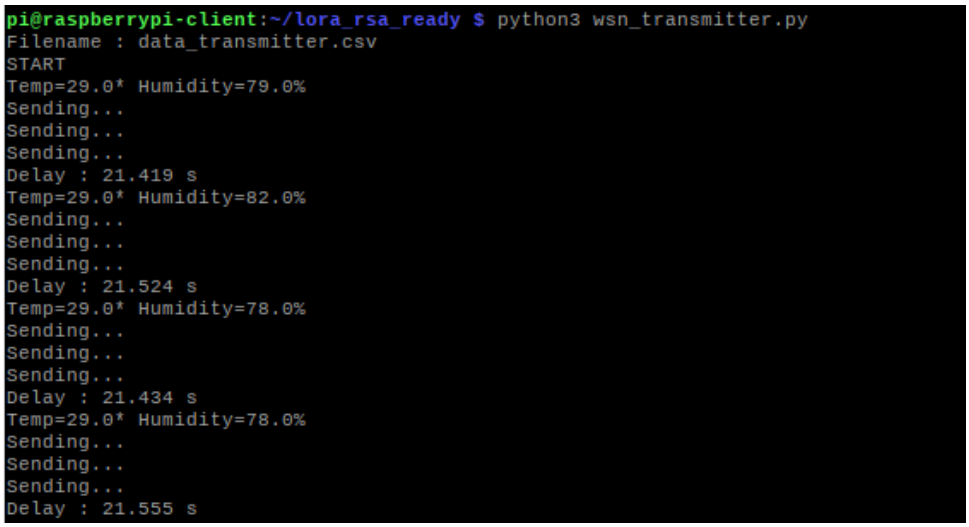
c) Fungsi enc_rsa()

Sebelum proses diatas dilakukan program pada sisi pengirim melakukan proses lain terlebih dahulu, yaitu proses konfigurasi frekuensi dan mode yang digunakan dan juga melakukan impor *public key* yang sudah dibuat sebelumnya. Proses yang dimaksud bisa dilihat pada potongan program di gambar 3.9.

```
BOARD.setup()
BOARD.reset()
pub = RSA.importKey(open("key/public.pem","r").read())

class mylora(LoRa):
    def __init__(self, verbose=False):
        super(mylora, self).__init__(verbose)
        self.set_mode(MODE.SLEEP)
        self.set_dio_mapping([1,0,0,0,0,0])

    def start(self):
        self.tx_counter = 0
        self.data = None
        self.sens = None
        BOARD.led_on()
        self.write_payload([])
        self.set_mode(MODE.TX)
        while True:
            pass

lora = mylora(verbose=False)
lora.set_freq(923)

try:
  lora.start()
```

Gambar 3.9 Proses konfigurasi program pengirim

Selanjutnya jika digabungkan maka alur program yang dilakukan pada program pengirim yaitu dimulai dari proses *setup* LoRa dan impor *public key* terlebih dahulu, kemudian mengambil data dari sensor DHT11, lalu mengenkripsi dan hash data tersebut yang akan dikirim, membagi 3 bagian data yang akan dikirim baru kemudia mengirim data yang telah dienkripsi dan dipecah ke program penerima. Program pada sisi pengirim disimpan menggunakan nama *wsn_transmitter.py* dan untuk menjalankannya menggunakan perintah python3 wsn_transmitter.py pada *terminal* Raspberry Pi. Tampilan program *wsn_transmitter.py* bisa dilihat pada gambar 3.10.



Gambar 3.10 Tampilan program *wsn_transmitter.py*

3.6 PROSES DEKRIPSI

Proses dekripsi dilakukan ketika 3 pecahan pesan telah sampai di program penerima, kemudian *ciphertext* dipisahkan dengan *digest* dan *digest* telah berhasil diverifikasi valid. Proses dekripsi dilakukan dengan menggunakan *private key*. Namun sebelum melakukan dekripsi, dilakukan konfigurasi frekuensi dan mode yang digunakan pada modul Dragino LoRa/HAT. Proses konfigurasi bisa dilihat pada potongan program pada gambar 3.11.

```
BOARD.setup()
BOARD.reset()
pri = RSA.importKey(open("key/private.pem","r").read())

class mylora(LoRa):
    def __init__(self, verbose=False):
        super(mylora, self).__init__(verbose)
        self.set_mode(MODE.SLEEP)
        self.set_dio_mapping([0] * 6)

    def start(self):
        self.tx_counter = 0
        self.data = b""
        self.reset_ptr_rx()
        self.set_mode(MODE.RXCONT)
        while True:
            pass

lora = mylora(verbose=False)
lora.set_freq(923)

try:
  lora.start()
```

Gambar 3.11 Proses konfigurasi program penerima

Selanjutnya proses dekripsi yang dilakukan sesuai dengan alur algoritma pemograman pada program penerima yang ditunjukkan pada gambar 3.5. Pertama, program akan menerima 3 buah pecahan pesan terenkripsi yang dikirimkan oleh program pengirim, kemudian program memanggil fungsi dec_rsa() untuk melakukan verifikasi *digest* dan melakukan dekripsi menggunakan *private key* untuk mendapatkan *plaintext*. Proses dekripsi yang dilakukan program pengirim dan fungsi dec_rsa() bisa dilihat pada gambar di bawah ini..

```
payload = self.read_payload(nocheck=True)
self.data += bytes(payload)
self.tx_counter += 1

self.set_mode(MODE.RXCONT)
if (self.tx_counter % 3 == 0):
    data = dec_rsa(self.data,pri).decode()
    print("Receive:",data)
    self.data = b""
```

(a)

```python
def dec_rsa(ciphertext,priv):
    rsa = PKCS1_v1_5.new(priv)

    sha3 = keccak.new(digest_bits=256)
    dsize = sha3.digest_size
    sentinel = Random.new().read(15+dsize)

    digest = sha3.update(ciphertext[:-dsize]).digest()
    if digest == ciphertext[-dsize:]:
        message = rsa.decrypt(ciphertext[:-dsize],sentinel)
        return message
    else:
        return b"Data terpotong"
```

(b)

Gambar 3.12 Proses dekripsi pada program

(a) Proses menerima dan dekripsi data

(b) Fungsi dec_rsa()

Program penerima disimpan menggunakan nama *wsn_receiver.py*. Untuk menjalankan program itu caranya dengan menjalankan perintah python3 *wsn_receiver.py*. Tampilan dari *wsn_receiver.py* ketika dijalankan bisa dilihat pada gambar 3.13.

```
pi@raspberrypi-server:~/lora_rsa_ready $ python3 wsn_receiver.py
Filename : data_receiver.csv
START
Receive: Temp=29.0* Humidity=79.0%
Delay : 0.289 s
Receive: Temp=29.0* Humidity=82.0%
Delay : 0.284 s
Receive: Temp=29.0* Humidity=78.0%
Delay : 0.299 s
Receive: Temp=29.0* Humidity=78.0%
Delay : 0.044 s
Receive: Temp=29.0* Humidity=78.0%
Delay : 0.268 s
Receive: Temp=29.0* Humidity=78.0%
Delay : 0.299 s
Receive: Temp=29.0* Humidity=78.0%
Delay : 0.273 s
Receive: Temp=28.0* Humidity=77.0%
Delay : 0.299 s
Receive: Temp=28.0* Humidity=77.0%
Delay : 0.166 s
Receive: Temp=28.0* Humidity=77.0%
Delay : 0.264 s
```

Gambar 3.13 Tampilan pada program *wsn_server.py*