LAPORAN TUGAS BESAR KRIPTOGRAFI

IMPLEMENTASI ALGORITMA ASCON UNTUK KEAMANAN DATA SENSOR HC-SR04 PADA SISTEM IoT MENGGUNAKAN PROTOKOL MQTT

Disusun Oleh:

Kelompok [X]

- Nama 1 (NIM)
- Nama 2 (NIM)
- Nama 3 (NIM)

Program Studi: [Nama Program Studi]
Fakultas: [Nama Fakultas]
Universitas: [Nama Universitas]
Tahun: 2025

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga kami dapat menyelesaikan Tugas Besar Kriptografi dengan judul "Implementasi Algoritma ASCON untuk Keamanan Data Sensor HC-SR04 pada Sistem IoT Menggunakan Protokol MQTT".

Tugas besar ini disusun untuk memenuhi salah satu syarat dalam mata kuliah Kriptografi. Dalam laporan ini, kami menjelaskan secara detail mengenai implementasi algoritma enkripsi ASCON pada sistem Internet of Things (IoT) yang menggunakan sensor HC-SR04 dan protokol komunikasi MQTT.

Kami menyadari bahwa laporan ini masih jauh dari sempurna. Oleh karena itu, kami sangat mengharapkan kritik dan saran yang membangun dari semua pihak demi perbaikan di masa mendatang.

Akhir kata, kami berharap laporan ini dapat memberikan manfaat bagi pembaca dan dapat menjadi referensi untuk pengembangan sistem IoT yang aman di masa depan.

Malang, November 2025

Penyusun

DAFTAR ISI

BAB I PENDAHULUAN

1.1 Latar Belakang

Internet of Things (IoT) telah menjadi teknologi yang semakin populer dalam berbagai bidang, mulai dari smart home, smart city, hingga industri 4.0. Sistem IoT memungkinkan perangkat-perangkat untuk saling berkomunikasi dan bertukar data melalui jaringan internet. Namun, pertukaran data pada sistem IoT seringkali menghadapi tantangan keamanan yang serius.

Salah satu protokol komunikasi yang banyak digunakan dalam sistem IoT adalah MQTT (Message Queuing Telemetry Transport). MQTT adalah protokol komunikasi yang ringan dan efisien, dirancang khusus untuk perangkat dengan sumber daya terbatas. Meskipun efisien, komunikasi MQTT pada dasarnya tidak menyediakan enkripsi data secara default, sehingga data yang ditransmisikan rentan terhadap serangan seperti eavesdropping (penyadapan) dan man-in-the-middle attack.

Untuk mengatasi masalah keamanan tersebut, diperlukan implementasi algoritma enkripsi yang lightweight namun tetap memberikan tingkat keamanan yang tinggi. ASCON adalah algoritma authenticated encryption yang dipilih sebagai standar lightweight cryptography oleh NIST pada tahun 2023. ASCON dirancang khusus untuk perangkat dengan resource terbatas seperti IoT devices, dengan memberikan proteksi terhadap confidentiality (kerahasiaan) dan integrity (integritas) data.

Dalam tugas besar ini, kami mengimplementasikan algoritma ASCON untuk mengamankan data dari sensor HC-SR04 (sensor ultrasonik pengukur jarak) yang dikirimkan melalui protokol MQTT menggunakan

microcontroller ESP32. Sistem ini akan diuji dari segi performa enkripsi/dekripsi, konsumsi energi, serta ketahanannya terhadap serangan pasif dan aktif.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, rumusan masalah dalam tugas besar ini adalah:

1. Bagaimana cara mengimplementasikan algoritma ASCON pada sistem IoT dengan sensor HC-SR04 dan protokol MQTT?
2. Berapa waktu yang dibutuhkan untuk proses enkripsi dan dekripsi data menggunakan algoritma ASCON?
3. Berapa konsumsi energi yang dibutuhkan oleh sistem IoT dengan implementasi enkripsi ASCON?
4. Bagaimana efektivitas algoritma ASCON dalam melindungi data dari serangan pasif (eavesdropping)?
5. Bagaimana efektivitas algoritma ASCON dalam melindungi data dari serangan aktif (modification attack, replay attack)?

1.3 Tujuan

Tujuan dari tugas besar ini adalah:

1. Mengimplementasikan algoritma enkripsi ASCON pada sistem IoT berbasis ESP32 dengan sensor HC-SR04 dan protokol MQTT.
2. Mengukur dan menganalisis waktu enkripsi dan dekripsi data menggunakan algoritma ASCON.
3. Mengukur dan menganalisis konsumsi energi sistem IoT dengan implementasi enkripsi ASCON.
4. Menguji dan menganalisis ketahanan sistem terhadap serangan pasif (passive attack).
5. Menguji dan menganalisis ketahanan sistem terhadap serangan aktif (active attack).
6. Membandingkan keamanan sistem dengan dan tanpa implementasi enkripsi ASCON.

1.4 Batasan Masalah

Untuk membatasi ruang lingkup pembahasan, batasan masalah dalam tugas besar ini adalah:

1. Sensor yang digunakan adalah HC-SR04 (sensor ultrasonik pengukur jarak).
2. Protokol komunikasi yang digunakan adalah MQTT.
3. Algoritma enkripsi yang diimplementasikan adalah ASCON-128.
4. Microcontroller yang digunakan adalah ESP32.
5. MQTT broker yang digunakan adalah broker public (broker.hivemq.com).
6. Enkripsi dan dekripsi dilakukan pada sisi Python (subscriber/publisher), bukan pada ESP32.

7. Pengujian serangan dilakukan dalam lingkungan simulasi, bukan pada jaringan production.
8. Pengukuran konsumsi energi dilakukan secara software-based (time-based estimation).

1.5 Manfaat

Manfaat yang diharapkan dari tugas besar ini adalah:

1. **Bagi Akademisi:** Memberikan referensi implementasi lightweight cryptography pada sistem IoT untuk penelitian lebih lanjut.
2. **Bagi Industri:** Memberikan contoh implementasi praktis keamanan data pada sistem IoT yang dapat diadopsi pada aplikasi real-world.
3. **Bagi Pengembang:** Menyediakan tutorial lengkap implementasi ASCON pada sistem IoT dengan ESP32 dan MQTT.
4. **Bagi Pengguna IoT:** Meningkatkan kesadaran akan pentingnya keamanan data pada perangkat IoT.

BAB II LANDASAN TEORI

2.1 Internet of Things (IoT)

2.1.1 Definisi IoT

Internet of Things (IoT) adalah konsep jaringan perangkat fisik yang terhubung ke internet dan dapat saling berkomunikasi serta bertukar data. Perangkat IoT dapat berupa sensor, aktuator, atau perangkat komputasi lainnya yang dilengkapi dengan kemampuan networking.

2.1.2 Komponen IoT

Sistem IoT umumnya terdiri dari empat komponen utama:

1. **Sensors/Actuators:** Perangkat untuk mengumpulkan data dari lingkungan atau melakukan aksi.
2. **Connectivity:** Metode komunikasi untuk mengirim data (WiFi, Bluetooth, LoRa, dll).
3. **Data Processing:** Pemrosesan data yang dikumpulkan untuk menghasilkan informasi.
4. **User Interface:** Interface untuk pengguna berinteraksi dengan sistem.

2.1.3 Tantangan Keamanan IoT

Sistem IoT menghadapi berbagai tantangan keamanan:

- **Resource Constraints:** Keterbatasan CPU, memory, dan power pada IoT devices
- **Unencrypted Communication:** Banyak protokol IoT tidak mengenkripsi data secara default

- **Physical Access:** IoT devices seringkali mudah diakses secara fisik oleh attacker
- **Large Attack Surface:** Banyaknya perangkat IoT memperluas area serangan
- **Lack of Updates:** Banyak IoT devices tidak mendapat security updates

2.2 Protokol MQTT

2.2.1 Pengertian MQTT

MQTT (Message Queuing Telemetry Transport) adalah protokol komunikasi berbasis publish-subscribe yang dirancang untuk aplikasi Machine-to-Machine (M2M) dan IoT. MQTT dikembangkan oleh IBM pada tahun 1999 dan kemudian menjadi standar OASIS pada tahun 2013.

2.2.2 Arsitektur MQTT

MQTT menggunakan arsitektur publish-subscribe dengan komponen utama:

1. **Publisher:** Client yang mengirim data (publish) ke broker
2. **Broker:** Server pusat yang menerima dan mendistribusikan pesan
3. **Subscriber:** Client yang menerima data dari broker berdasarkan topic
4. **Topic:** Channel komunikasi yang digunakan untuk kategorisasi pesan

2.2.3 Karakteristik MQTT

- **Lightweight:** Header minimal (2 bytes) untuk efisiensi bandwidth
- **Quality of Service (QoS):** Tiga level QoS (0, 1, 2) untuk reliability
- **Persistent Session:** Mendukung persistent connection
- **Last Will and Testament (LWT):** Notifikasi jika client disconnect
- **Retained Messages:** Pesan yang disimpan broker untuk subscriber baru

2.2.4 Keamanan MQTT

MQTT mendukung beberapa mekanisme keamanan:

- **Username/Password Authentication:** Autentikasi dasar
- **TLS/SSL:** Enkripsi transport layer
- **Client Certificates:** Autentikasi menggunakan sertifikat

Namun, implementasi keamanan pada level aplikasi (payload encryption) tetap diperlukan untuk proteksi end-to-end.

2.3 Sensor HC-SR04

2.3.1 Spesifikasi HC-SR04

HC-SR04 adalah sensor ultrasonik yang digunakan untuk mengukur jarak dengan spesifikasi:

- **Tegangan Operasi:** 5V DC
- **Arus Operasi:** 15 mA
- **Frekuensi Ultrasonik:** 40 kHz
- **Jangkauan:** 2 cm - 400 cm
- **Akurasi:** ±3 mm
- **Sudut Pengukuran:** ~15 derajat

2.3.2 Prinsip Kerja

HC-SR04 bekerja dengan prinsip:

1. Trigger pin diberikan pulsa HIGH selama 10 µs
2. Sensor mengirimkan 8 pulsa ultrasonik 40 kHz
3. Gelombang ultrasonik dipantulkan oleh objek
4. Sensor menerima pantulan melalui Echo pin
5. Durasi Echo HIGH berbanding lurus dengan jarak
6. Jarak = (Durasi × Kecepatan Suara) / 2
7. Jarak (cm) = Durasi (µs) × 0.034 / 2

2.3.3 Aplikasi

HC-SR04 banyak digunakan untuk:

- Sistem pengukuran jarak otomatis
- Robot obstacle avoidance
- Parking sensor
- Level monitoring
- Security system

2.4 Algoritma ASCON

2.4.1 Pengenalan ASCON

ASCON adalah keluarga authenticated encryption with associated data (AEAD) yang dirancang untuk aplikasi

lightweight cryptography. ASCON dipilih sebagai standar lightweight cryptography oleh NIST pada Februari 2023.

2.4.2 Varian ASCON

ASCON memiliki beberapa varian:

1. **ASCON-128:** 128-bit key, 128-bit nonce, 128-bit tag
2. **ASCON-128a:** Versi lebih cepat dengan rate 128-bit
3. **ASCON-80pq:** 160-bit key untuk post-quantum security
4. **ASCON-Hash:** Hash function berbasis ASCON permutation
5. **ASCON-Hasha:** Versi lebih cepat dari ASCON-Hash

2.4.3 Struktur ASCON

ASCON menggunakan sponge construction dengan komponen:

- **State Size:** 320 bits (5 × 64-bit words)
- **Rate:** 64 bits (ASCON-128) atau 128 bits (ASCON-128a)
- **Capacity:** 256 bits (ASCON-128) atau 192 bits (ASCON-128a)
- **Rounds:** 12 rounds untuk initialization/finalization, 6 atau 8 rounds untuk processing

2.4.4 Operasi ASCON

Proses enkripsi ASCON meliputi:

1. **Initialization:**
   - State diinisialisasi dengan IV, key, dan nonce
   - Permutation dengan 12 rounds
   - XOR dengan key
2. **Associated Data Processing:**
   - Associated data di-XOR ke state
   - Permutation dengan 6/8 rounds per block
   - Domain separation dengan S[4] ^= 1
3. **Plaintext Processing:**
   - Plaintext di-XOR dengan state → ciphertext
   - State di-update dengan ciphertext

- Permutation dengan 6/8 rounds per block

4. **Finalization:**
    - XOR dengan key
    - Permutation dengan 12 rounds
    - Extract tag dari state

2.4.5 Keunggulan ASCON

- **Lightweight:** Cocok untuk hardware dan software terbatas
- **Efficient:** Performa tinggi pada berbagai platform
- **Secure:** Resisten terhadap berbagai jenis serangan kriptografi
- **Authenticated Encryption:** Menyediakan confidentiality dan integrity sekaligus
- **Standardized:** Dipilih oleh NIST sebagai standar lightweight crypto

2.4.6 Keamanan ASCON

ASCON memberikan proteksi terhadap:

- **Confidentiality:** Plaintext tidak dapat dibaca tanpa key
- **Integrity:** Modifikasi ciphertext terdeteksi melalui tag verification
- **Authenticity:** Data berasal dari pengirim yang sah
- **Side-channel Resistance:** Desain mempertimbangkan serangan side-channel

2.5 ESP32 Microcontroller

2.5.1 Spesifikasi ESP32

ESP32 adalah microcontroller 32-bit dengan spesifikasi:

- **Processor:** Dual-core Xtensa LX6, hingga 240 MHz
- **RAM:** 520 KB SRAM
- **Flash:** 4 MB (dapat diperluas)
- **WiFi:** 802.11 b/g/n (2.4 GHz)
- **Bluetooth:** Bluetooth Classic dan BLE
- **GPIO:** 34 programmable pins
- **ADC:** 18 channel, 12-bit resolution

- **Power:** 3.3V operating voltage

2.5.2 Konsumsi Daya ESP32

Mode operasi ESP32 dengan konsumsi daya:

Mode	Konsumsi Arus	Keterangan
Active (WiFi TX)	160-260 mA	Transmit data via WiFi
Active (WiFi RX)	95-100 mA	Receive data via WiFi
Active (CPU)	40-80 mA	CPU aktif tanpa WiFi
Modem Sleep	20-40 mA	WiFi off, CPU aktif
Light Sleep	0.8 mA	CPU suspended
Deep Sleep	10 μA	Minimal power

2.5.3 Kelebihan ESP32 untuk IoT

- **Powerful Processing:** Dual-core processor untuk multitasking
- **Built-in Connectivity:** WiFi dan Bluetooth terintegrasi
- **Low Power Modes:** Berbagai mode untuk efisiensi energi
- **Rich Peripherals:** ADC, DAC, PWM, I2C, SPI, UART
- **Cost-Effective:** Harga terjangkau untuk fitur yang ditawarkan
- **Large Community:** Dukungan komunitas dan library yang luas

2.6 Ancaman Keamanan IoT

2.6.1 Klasifikasi Serangan

Serangan pada sistem IoT dapat diklasifikasikan menjadi:

A. Serangan Pasif (Passive Attack)

Serangan yang hanya mengamati/mendengarkan komunikasi tanpa memodifikasi data:

1. **Eavesdropping (Penyadapan):**
   - Attacker menangkap dan membaca data yang ditransmisikan
   - Tidak meninggalkan jejak pada sistem
   - Mengancam confidentiality
2. **Traffic Analysis:**
   - Menganalisis pola komunikasi (frekuensi, ukuran, timing)

- Mendapatkan informasi dari metadata, bukan content
- Sulit dideteksi

B. Serangan Aktif (Active Attack)

Serangan yang memodifikasi atau mengganggu komunikasi:

1. **Data Modification Attack:**
   - Attacker mengubah isi data yang ditransmisikan
   - Mengancam integrity data
   - Dapat menyebabkan sistem salah membuat keputusan
2. **Replay Attack:**
   - Attacker menangkap data valid dan mengirim ulang
   - Sistem menerima data lama sebagai data baru
   - Dapat menyebabkan aksi yang tidak diinginkan
3. **Man-in-the-Middle (MITM) Attack:**
   - Attacker berada di tengah komunikasi
   - Dapat membaca dan memodifikasi data
   - Mengancam confidentiality dan integrity
4. **Denial of Service (DoS):**
   - Attacker membanjiri sistem dengan request
   - Membuat sistem tidak dapat melayani request legitimate
   - Mengancam availability

2.6.2 Countermeasures

Proteksi terhadap ancaman keamanan:

1. **Encryption:** Melindungi confidentiality
2. **Authentication:** Memastikan identitas pengirim
3. **Integrity Check:** Mendeteksi modifikasi data
4. **Timestamp/Nonce:** Mencegah replay attack
5. **Rate Limiting:** Mencegah DoS attack

BAB III PROSES INSTALASI

Pada bab ini dijelaskan tahapan instalasi software, library, dan konfigurasi yang diperlukan untuk mengimplementasikan sistem.

3.1 Instalasi Software

3.1.1 Arduino IDE atau VSCode dengan PlatformIO

Option A: Arduino IDE

1. Download Arduino IDE dari https://www.arduino.cc/en/software
2. Install Arduino IDE sesuai sistem operasi
3. Buka Arduino IDE

Option B: VSCode dengan PlatformIO (Recommended)

1. Download dan install Visual Studio Code dari https://code.visualstudio.com/
2. Buka VSCode
3. Pergi ke Extensions (Ctrl+Shift+X)
4. Cari "PlatformIO IDE"
5. Klik Install
6. Restart VSCode setelah instalasi selesai

3.1.2 Python

1. Download Python 3.7 atau lebih baru dari https://www.python.org/downloads/
2. Jalankan installer
3. **PENTING:** Centang opsi "Add Python to PATH"
4. Pilih "Install Now"
5. Verifikasi instalasi dengan membuka command prompt/terminal:

```bash
python --version
```

3.1.3 MQTT Broker (Optional)

Option A: Gunakan Public Broker

Gunakan broker public seperti:

- broker.hivemq.com (port 1883)
- test.mosquitto.org (port 1883)
- broker.emqx.io (port 1883)

Option B: Install Mosquitto Lokal

Windows:

1. Download dari https://mosquitto.org/download/
2. Install dengan default settings
3. Jalankan dari Services atau command line:

```bash
mosquitto -v
```

Linux (Ubuntu/Debian):

```bash
sudo apt-get update
sudo apt-get install mosquitto mosquitto-clients
sudo systemctl start mosquitto
sudo systemctl enable mosquitto
```

macOS:

```bash
brew install mosquitto
brew services start mosquitto
```

3.2 Instalasi Library dan Dependencies

3.2.1 ESP32 Board Package

Untuk Arduino IDE:

1. Buka Arduino IDE
2. Pergi ke File → Preferences

3. Pada "Additional Board Manager URLs", tambahkan:

```
https://dl.espressif.com/dl/package_esp32_index.json
```

4. Klik OK
5. Pergi ke Tools → Board → Boards Manager
6. Cari "esp32"
7. Install "esp32 by Espressif Systems"
8. Tunggu hingga instalasi selesai

Untuk PlatformIO:

PlatformIO akan otomatis download ESP32 platform saat pertama kali membuat project ESP32.

3.2.2 Library Arduino

Install library berikut melalui Library Manager:

1. **PubSubClient** (untuk MQTT):
   - Sketch → Include Library → Manage Libraries
   - Cari "PubSubClient"
   - Install "PubSubClient by Nick O'Leary"
2. **NewPing** (untuk HC-SR04) (Optional):
   - Cari "NewPing"
   - Install "NewPing by Tim Eckel"

Atau secara manual, library sudah include dalam ESP32 core:

- WiFi.h (built-in)
- PubSubClient.h (need install)

3.2.3 Library Python

Install library Python yang diperlukan:

```
bash
```

```
# Install pip jika belum ada
python -m ensurepip --upgrade

# Install library
pip install paho-mqtt
pip install matplotlib
pip install numpy
```

Atau install dari requirements.txt:

```
bash

# Buat file requirements.txt dengan isi:
paho-mqtt==1.6.1
matplotlib==3.7.1
numpy==1.24.3

# Install dari file
pip install -r requirements.txt
```

Verifikasi instalasi:

```
bash

pip list | grep paho-mqtt
pip list | grep matplotlib
```

3.3 Setup Hardware

3.3.1 Komponen yang Dibutuhkan

Daftar komponen hardware:

1. ESP32 Development Board (1 unit)
2. Sensor HC-SR04 Ultrasonic (1 unit)
3. Breadboard (1 unit)
4. Kabel Jumper Male-to-Male (4 buah)
5. Kabel USB Micro/Type-C untuk ESP32 (1 buah)
6. Power supply (dari USB komputer)

3.3.2 Wiring Diagram

Koneksi antara ESP32 dan HC-SR04:

```
HC-SR04        ESP32
--------       --------
VCC     →      5V (atau 3.3V jika ESP32 3.3V tolerant)
GND     →      GND
TRIG    →      GPIO 5
ECHO    →      GPIO 18
```

Catatan Penting:

- ESP32 GPIO adalah 3.3V tolerant
- HC-SR04 ECHO pin output 5V
- Untuk safety, gunakan voltage divider pada ECHO pin (resistor 1kΩ dan 2kΩ)
- Atau gunakan level shifter 5V-3.3V

Voltage Divider (Recommended):

```
ECHO (5V) --- [1kΩ] --- GPIO18 --- [2kΩ] --- GND
```

3.3.3 Langkah Perakitan

1. Letakkan ESP32 pada breadboard
2. Letakkan sensor HC-SR04 pada breadboard
3. Hubungkan VCC HC-SR04 ke pin 5V ESP32
4. Hubungkan GND HC-SR04 ke GND ESP32
5. Hubungkan TRIG HC-SR04 ke GPIO 5 ESP32
6. Hubungkan ECHO HC-SR04 ke GPIO 18 ESP32 (dengan voltage divider jika perlu)
7. Double check semua koneksi
8. Hubungkan ESP32 ke komputer via USB

3.3.4 Testing Hardware

Upload program test sederhana untuk memastikan hardware bekerja:

```
cpp
```

```
#define TRIGGER_PIN 5
#define ECHO_PIN 18

void setup() {
  Serial.begin(115200);
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(1000);
}
```

Jika hardware bekerja dengan baik, Serial Monitor akan menampilkan jarak yang terukur.

3.4 Konfigurasi MQTT Broker

3.4.1 Menggunakan Public Broker

Untuk menggunakan public broker, tidak ada konfigurasi khusus yang diperlukan. Cukup gunakan address broker dalam code:

cpp

```
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;
```

python

```
BROKER = "broker.hivemq.com"
PORT = 1883
```

Keuntungan:

- Tidak perlu setup server
- Langsung bisa digunakan
- Gratis

Kekurangan:

- Data tidak private (siapa saja bisa subscribe)
- Tidak ada authentication
- Tidak cocok untuk production

3.4.2 Menggunakan Lokal Broker (Optional)

Jika menggunakan Mosquitto lokal:

1. Edit konfigurasi Mosquitto (mosquitto.conf):

```
listener 1883
allow_anonymous true
```

2. Restart Mosquitto service
3. Gunakan IP lokal komputer sebagai broker:

cpp

```
const char* mqtt_server = "192.168.1.100"; // IP komputer
```

4. Pastikan firewall mengizinkan port 1883

3.4.3 Test Koneksi MQTT

Test menggunakan MQTT client (Postman, MQTT Explorer, atau mosquitto_sub):

Menggunakan mosquitto_sub:

bash

```
mosquitto_sub -h broker.hivemq.com -t "test/topic" -v
```

Menggunakan mosquitto_pub:

```bash
mosquitto_pub -h broker.hivemq.com -t "test/topic" -m "Hello MQTT"
```

Jika pesan terkirim dan diterima, konfigurasi MQTT sudah benar.

3.5 Struktur Folder Project

Organisasi folder project:

```
Artefak_Kelompok[X]_MQTT_HCSR04/
├── arduino/
│   └── esp32_complete_with_energy/
│       └── esp32_complete_with_energy.ino
├── python/
│   ├── ascon.py
│   ├── mqtt_publisher.py
│   ├── mqtt_subscriber.py
│   ├── energy_analyzer.py
│   ├── interactive_attack_simulator.py
│   ├── attack_monitor.py
│   └── testing_performance.py
├── documentation/
│   ├── wiring_diagram.png
│   ├── system_architecture.png
│   └── screenshots/
│       ├── serial_monitor.png
│       ├── mqtt_publisher.png
│       ├── mqtt_subscriber.png
│       ├── energy_graph.png
│       ├── attack_passive.png
│       └── attack_active.png
├── results/
│   ├── performance_results.txt
│   ├── energy_report.txt
│   ├── attack_log.txt
│   └── energy_analysis.png
├── README.md
└── requirements.txt
```

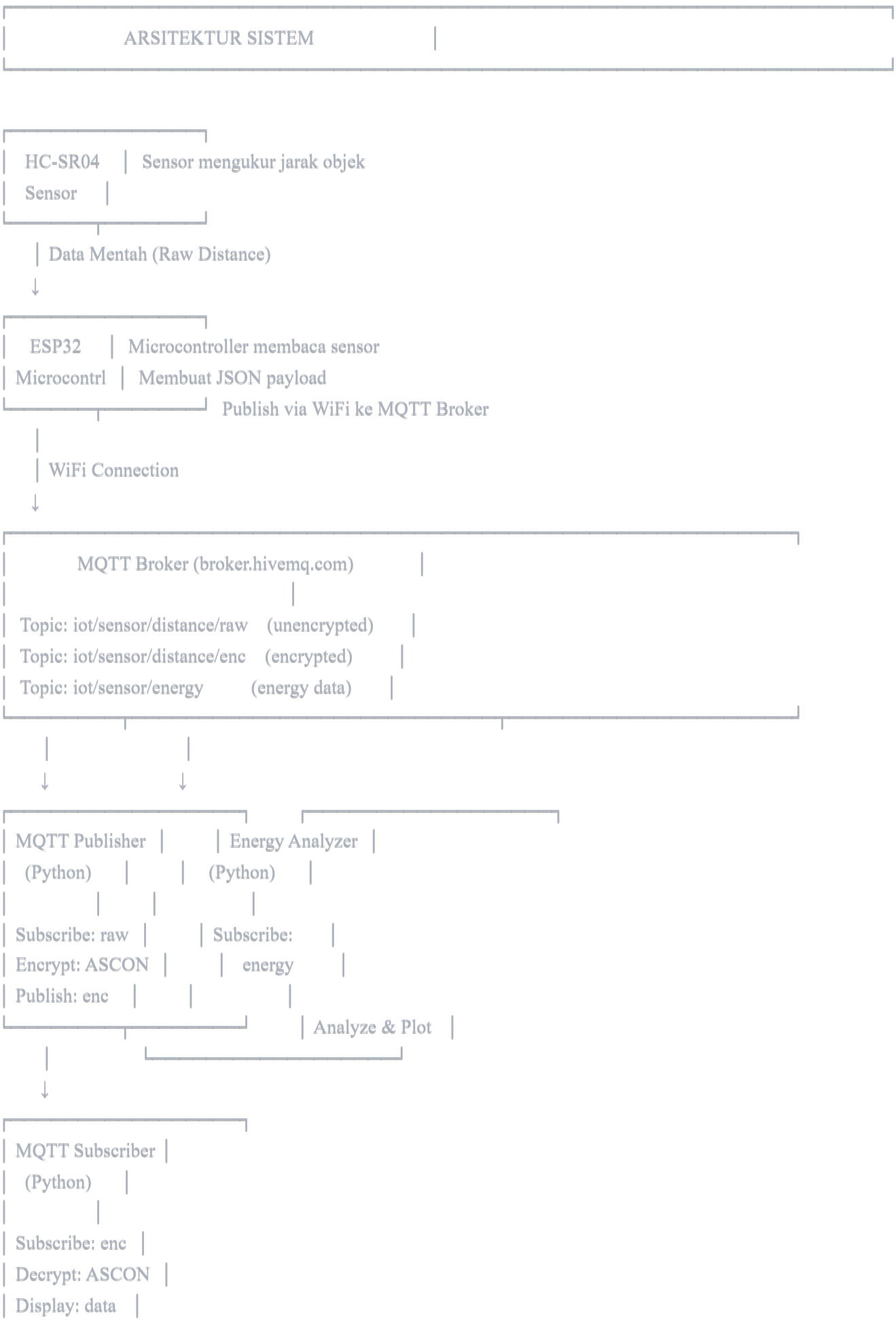
BAB IV PENGERJAAN

Pada bab ini dijelaskan proses perancangan dan implementasi sistem secara detail.

4.1 Arsitektur Sistem

4.1.1 Diagram Arsitektur

Sistem yang dibangun memiliki arsitektur sebagai berikut:


ARSITEKTUR SISTEM
HC-SR04 Sensor
Sensor mengukur jarak objek
Data Mentah (Raw Distance)
ESP32 Microcontrl
Microcontroller membaca sensor
Membuat JSON payload
Publish via WiFi ke MQTT Broker
WiFi Connection
MQTT Broker (broker.hivemq.com)
Topic: iot/sensor/distance/raw (unencrypted)
Topic: iot/sensor/distance/enc (encrypted)
Topic: iot/sensor/energy (energy data)
MQTT Publisher
(Python)
Subscribe: raw
Encrypt: ASCON
Publish: enc
Energy Analyzer
(Python)
Subscribe:
energy
Analyze & Plot
MQTT Subscriber
(Python)
Subscribe: enc
Decrypt: ASCON
Display: data

4.1.2 Alur Data

Alur data dalam sistem:

1. **Data Acquisition:**
   - Sensor HC-SR04 mengukur jarak objek
   - ESP32 membaca data dari sensor
   - Data diformat dalam JSON
2. **Data Transmission (Unencrypted):**
   - ESP32 publish data ke topic `iot/sensor/distance/raw`
   - Data dalam format plaintext
3. **Data Encryption:**
   - MQTT Publisher (Python) subscribe ke topic raw
   - Data dienkripsi menggunakan ASCON-128
   - Encrypted data dipublish ke topic `iot/sensor/distance/enc`
4. **Data Decryption:**
   - MQTT Subscriber (Python) subscribe ke topic enc
   - Data didekripsi menggunakan ASCON-128
   - Data asli ditampilkan
5. **Energy Monitoring:**
   - ESP32 menghitung waktu eksekusi setiap operasi
   - Energy data dipublish ke topic `iot/sensor/energy`
   - Energy Analyzer menganalisis dan membuat grafik

4.1.3 Komponen Sistem

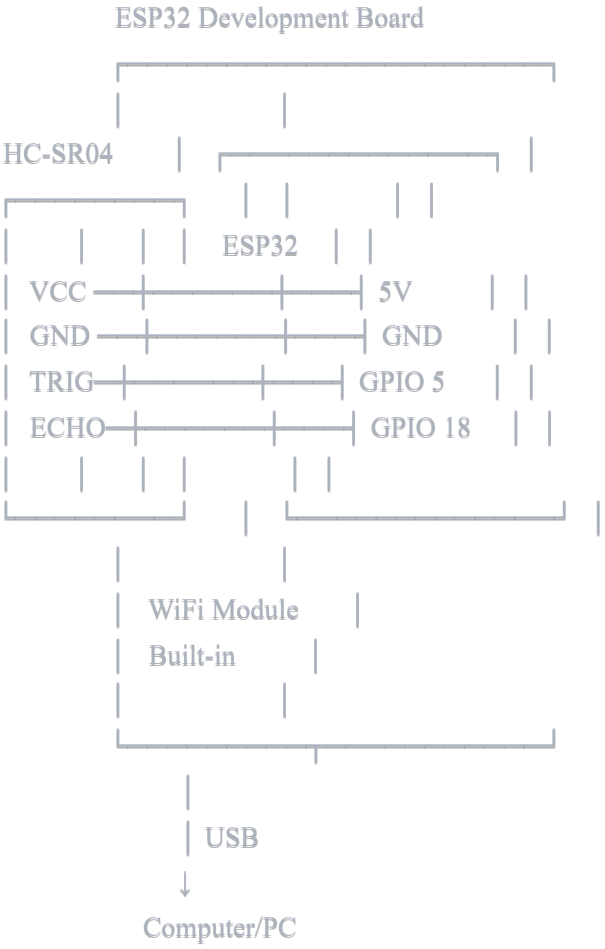
Sistem terdiri dari komponen-komponen berikut:

Komponen	Fungsi	Teknologi
Sensor	Mengukur jarak	HC-SR04 Ultrasonic
Microcontroller	Membaca sensor, MQTT client	ESP32
MQTT Broker	Message routing	HiveMQ / Mosquitto
Publisher	Enkripsi data	Python + ASCON
Subscriber	Dekripsi data	Python + ASCON
Energy Analyzer	Analisis konsumsi energi	Python + Matplotlib
Attack Tools	Security testing	Python

4.2 Perancangan Hardware

4.2.1 Skema Koneksi

Koneksi hardware mengikuti skema:



4.2.2 Spesifikasi Pin

Pin ESP32	Fungsi	Koneksi
GPIO 5	Output	HC-SR04 TRIG
GPIO 18	Input	HC-SR04 ECHO
5V	Power	HC-SR04 VCC
GND	Ground	HC-SR04 GND

4.2.3 Perhitungan Daya

Konsumsi daya sistem:

- **ESP32 (Active WiFi TX):** ~200 mA × 3.3V = 660 mW
- **HC-SR04 (Active):** ~15 mA × 5V = 75 mW
- **Total:** ~735 mW

Estimasi baterai:

- Baterai 3000 mAh, 3.7V = 11.1 Wh
- Runtime: 11.1 Wh / 0.735 W ≈ 15 jam

4.3 Implementasi Firmware ESP32

4.3.1 Struktur Program

Program ESP32 terdiri dari beberapa bagian utama:

```
cpp
```

```
// 1. Include Libraries
#include <WiFi.h>
#include <PubSubClient.h>

// 2. Configuration
#define ENERGY_MONITORING_ENABLED true
const char* ssid = "WiFi_SSID";
const char* mqtt_server = "broker.hivemq.com";

// 3. Global Variables
struct EnergyStats { ... };
WiFiClient espClient;
PubSubClient client(espClient);

// 4. Function Declarations
float calculateEnergy(...);
long readDistance();
void setup_wifi();
void reconnect();

// 5. Setup Function
void setup() {
  Serial.begin(115200);
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
}

// 6. Main Loop
void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  // Read sensor
  // Create JSON
  // Publish MQTT
  // Calculate energy
  // Print statistics
}
```

4.3.2 Fungsi Pembacaan Sensor

Fungsi untuk membaca jarak dari HC-SR04:

```cpp
long readDistance() {
  unsigned long startTime = micros();

  // Trigger pulse
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  // Read echo
  long duration = pulseIn(ECHO_PIN, HIGH, 30000);
  long distance = duration * 0.034 / 2;

  if (distance == 0 || distance > MAX_DISTANCE) {
    distance = 0;
  }

  // Calculate energy if monitoring enabled
  if (ENERGY_MONITORING_ENABLED) {
    energyStats.sensor_read_time_us = micros() - startTime;
    energyStats.sensor_energy_mj = calculateEnergy(
      CURRENT_CPU_ACTIVE,
      energyStats.sensor_read_time_us
    );
  }

  return distance;
}
```

4.3.3 Fungsi Energy Monitoring

Fungsi untuk menghitung konsumsi energi:

```cpp
```

```cpp
float calculateEnergy(float current_ma, unsigned long time_us) {
  // Energy (mJ) = Power (mW) × Time (ms)
  // Power (mW) = Voltage (V) × Current (mA)
  // Time (ms) = time_us / 1000

  float power_mw = VOLTAGE * current_ma;
  float time_ms = time_us / 1000.0;
  float energy_mj = power_mw * time_ms;

  return energy_mj;
}
```

Konstanta current draw berdasarkan datasheet ESP32:

```cpp
const float CURRENT_WIFI_TX = 200.0;     // mA
const float CURRENT_WIFI_RX = 100.0;     // mA
const float CURRENT_CPU_ACTIVE = 60.0;   // mA
const float CURRENT_IDLE = 30.0;         // mA
const float VOLTAGE = 3.3;               // Volt
```

4.3.4 Fungsi MQTT Publishing

Fungsi untuk publish data via MQTT:

```cpp
```

```
// Publish sensor data
String jsonData = "{";
jsonData += "\"id\":\"" + String(mqtt_client_id) + "\",";
jsonData += "\"count\":" + String(messageCount) + ",";
jsonData += "\"distance\":" + String(distance) + ",";
jsonData += "\"timestamp\":" + String(millis()) + ",";
jsonData += "\"unit\":\"cm\"";
jsonData += "}";

client.publish(mqtt_topic_raw, jsonData.c_str());

// Publish energy data
String energyJson = "{";
energyJson += "\"cycle\":" + String(energyStats.cycle_count) + ",";
energyJson += "\"sensor_time_us\":" + String(energyStats.sensor_read_time_us) + ",";
energyJson += "\"total_energy_mj\":" + String(energyStats.total_energy_mj, 3) + ",";
energyJson += "\"cumulative_energy_mj\":" + String(energyStats.cumulative_energy_mj, 3);
energyJson += "}";

client.publish(mqtt_topic_energy, energyJson.c_str());
```

4.3.5 Output Serial Monitor

Program menampilkan statistik energi dalam format tabel:

```
╔═══════╦═══════════╦═══════════╦═══════════╦═══════════╦═══════════╦═══════════╗
║ Cycle ┃ Read(µs)  ┃ JSON(µs)  ┃ MQTT(µs)  ┃ Total(µs) ┃ Energy(mJ)┃ Power(mW) ║
╠═══════╬═══════════╬═══════════╬═══════════╬═══════════╬═══════════╬═══════════╣
║   1   ┃     856   ┃     124   ┃    3542   ┃    4522   ┃    2.826  ┃   624.78  ║
║   2   ┃     843   ┃     118   ┃    3498   ┃    4459   ┃    2.787  ┃   625.12  ║
```

4.4 Implementasi Enkripsi dan Dekripsi

4.4.1 Implementasi ASCON

File `ascon.py` berisi implementasi algoritma ASCON yang sudah disediakan oleh dosen. Fungsi utama yang digunakan:

Fungsi Enkripsi:

```
python
```

```
def ascon_encrypt(key, nonce, associateddata, plaintext, variant="Ascon-128"):
    """
    Ascon encryption.
    key: 16 bytes (128-bit)
    nonce: 16 bytes (128-bit)
    associateddata: arbitrary length
    plaintext: arbitrary length
    variant: "Ascon-128", "Ascon-128a", or "Ascon-80pq"
    returns: ciphertext + tag (16 bytes)
    """
```

Fungsi Dekripsi:

```
python

def ascon_decrypt(key, nonce, associateddata, ciphertext, variant="Ascon-128"):
    """
    Ascon decryption.
    returns: plaintext or None if verification fails
    """
```

4.4.2 MQTT Publisher dengan Enkripsi

File: mqtt_publisher.py

Konfigurasi:

```
python

# MQTT Configuration
BROKER = "broker.hivemq.com"
PORT = 1883
TOPIC_RAW = "iot/sensor/distance/raw"
TOPIC_ENCRYPTED = "iot/sensor/distance/enc"

# ASCON Configuration
KEY = "asconciphertest1".encode('utf-8')  # 16 bytes
NONCE = "asconcipher1test".encode('utf-8')  # 16 bytes
ASSOCIATED_DATA = b"ASCON"
VARIANT = "Ascon-128"
```

Fungsi Enkripsi:

```
python
```

```python
def encrypt_data(plaintext_data):
    try:
        if isinstance(plaintext_data, dict):
            plaintext_data = json.dumps(plaintext_data)

        # Enkripsi menggunakan ASCON
        ciphertext = ascon.demo_aead_c(
            VARIANT,
            plaintext_data,
            KEY,
            NONCE,
            ASSOCIATED_DATA
        )

        return ciphertext
    except Exception as e:
        print(f"❌ Encryption error: {e}")
        return None
```

Callback Handler:

```python
```

```
def on_message(client, userdata, msg):
    # Decode payload
    payload = msg.payload.decode('utf-8')

    # Parse JSON
    data = json.loads(payload)

    # Enkripsi data
    start_time = time.time()
    encrypted_data = encrypt_data(payload)
    encryption_time = (time.time() - start_time) * 1000  # ms

    if encrypted_data:
        # Convert to hex string
        encrypted_hex = encrypted_data.hex()

        # Publish encrypted data
        encrypted_payload = {
            "encrypted_data": encrypted_hex,
            "encryption_time_ms": round(encryption_time, 3),
            "algorithm": VARIANT,
            "timestamp": datetime.now().isoformat()
        }

        client.publish(TOPIC_ENCRYPTED, json.dumps(encrypted_payload))
```

4.4.3 MQTT Subscriber dengan Dekripsi

File: mqtt_subscriber.py

Fungsi Dekripsi:

python

```python
def decrypt_data(ciphertext_bytes):
    try:
        # Dekripsi menggunakan ASCON
        plaintext_bytes = ascon.demo_aead_p(VARIANT, ciphertext_bytes)

        if plaintext_bytes is None:
            print("❌ Decryption failed: Authentication tag mismatch!")
            return None

        # Convert bytes to string
        plaintext = plaintext_bytes.decode('utf-8')
        return plaintext

    except Exception as e:
        print(f"❌ Decryption error: {e}")
        return None
```

Callback Handler:

python

```python
def on_message(client, userdata, msg):
    payload = msg.payload.decode('utf-8')
    encrypted_payload = json.loads(payload)
    encrypted_hex = encrypted_payload.get("encrypted_data")

    # Convert hex to bytes
    ciphertext_bytes = bytes.fromhex(encrypted_hex)

    # Dekripsi data
    start_time = time.time()
    decrypted_data = decrypt_data(ciphertext_bytes)
    decryption_time = (time.time() - start_time) * 1000  # ms

    if decrypted_data:
        print(f"✅ Decryption successful!")
        print(f"⏱️ Decryption time: {decryption_time:.3f} ms")
        print(f"📦 Decrypted data: {decrypted_data}")

        # Parse decrypted JSON
        sensor_data = json.loads(decrypted_data)
        print(f"📏 Distance: {sensor_data.get('distance')} cm")
```

4.4.4 Key Management

Dalam implementasi ini, key dan nonce disimpan secara hardcoded untuk tujuan demonstrasi:

```python
KEY = "asconciphertest1".encode('utf-8')
NONCE = "asconcipher1test".encode('utf-8')
```

Catatan Keamanan:

- Pada production, key harus disimpan dengan aman (environment variables, key vault)
- Nonce harus unik untuk setiap sesi enkripsi
- Associated data dapat berisi metadata tambahan

4.5 Integrasi Sistem

4.5.1 Alur Integrasi

Proses integrasi sistem dilakukan dengan tahapan:

1. **Testing Individual Components:**
   - Test ESP32 + Sensor standalone
   - Test MQTT connection
   - Test ASCON encryption/decryption
2. **Integration Testing:**
   - ESP32 publish ke broker
   - Publisher subscribe dan encrypt
   - Subscriber decrypt dan verify
3. **System Testing:**
   - End-to-end data flow
   - Performance testing
   - Security testing

4.5.2 Error Handling

Setiap komponen dilengkapi error handling:

ESP32:

```cpp
if (!client.connected()) {
  reconnect();  // Auto-reconnect jika disconnect
}

if (distance == 0) {
  distance = 0;  // Handle out of range
}
```

Python:

```python
try:
    encrypted_data = encrypt_data(payload)
except Exception as e:
    print(f"❌ Error: {e}")
    stats["errors"] += 1
```

4.5.3 Logging dan Monitoring

Setiap komponen mencatat aktivitasnya:

```python
stats = {
    "total_messages": 0,
    "encrypted_messages": 0,
    "errors": 0,
    "start_time": time.time()
}

# Update stats
stats["total_messages"] += 1
if encrypted_data:
    stats["encrypted_messages"] += 1
```

BAB V DEMO DAN PENGUJIAN

Pada bab ini dijelaskan proses demonstrasi sistem dan hasil pengujian yang dilakukan.

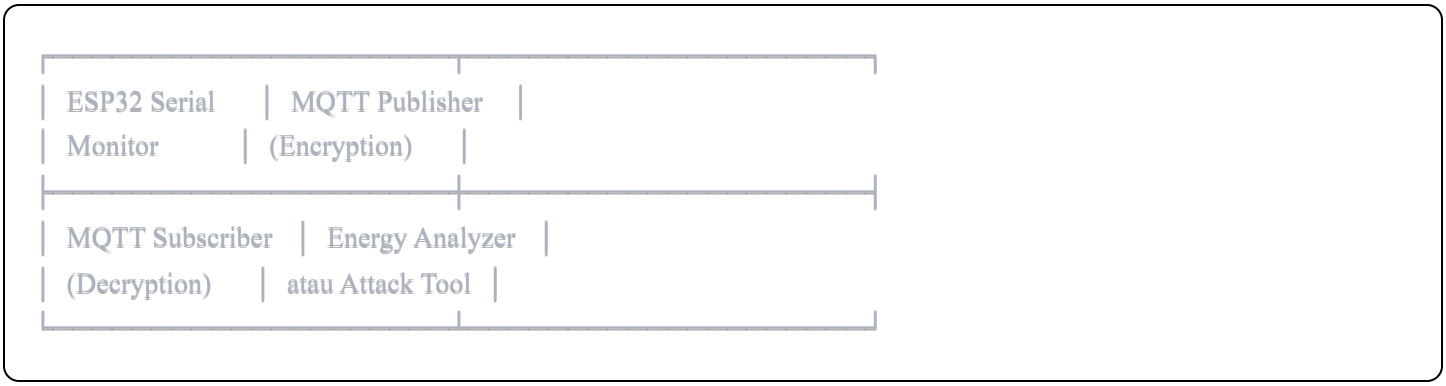
5.1 Demo Sistem

5.1.1 Setup Demo

Untuk mendemonstrasikan sistem, diperlukan setup berikut:

1. **Hardware:** ESP32 + HC-SR04 terhubung ke komputer
2. **Software:** 4-5 terminal/window berjalan bersamaan
3. **Network:** Koneksi internet untuk MQTT broker

Layout Terminal:



5.1.2 Langkah Demo

Step 1: Upload Program ke ESP32

```
1. Buka esp32_complete_with_energy.ino
2. Konfigurasi WiFi SSID dan password
3. Upload ke ESP32
4. Buka Serial Monitor (115200 baud)
5. Verifikasi koneksi WiFi dan MQTT
```

Step 2: Jalankan MQTT Publisher

```bash
cd python
python mqtt_publisher.py
```

Output yang diharapkan:

```
✅ Connected to MQTT Broker!
📡 Subscribing to topic: iot/sensor/distance/raw
✅ Starting MQTT loop...
```

Step 3: Jalankan MQTT Subscriber

```
bash

python mqtt_subscriber.py
```

Output yang diharapkan:

```
✅ Connected to MQTT Broker!
📡 Subscribing to topic: iot/sensor/distance/enc
✅ Starting MQTT loop...
```

Step 4: Jalankan Energy Analyzer

```
bash

python energy_analyzer.py
```

Step 5: Observasi Data Flow

Setelah semua komponen berjalan, observasi:

1. **ESP32 Serial Monitor:**

```
║   1 │   856 │   124 │  3542 │  4522 │ 2.826 │ 624.78 ║
║   2 │   843 │   118 │  3498 │  4459 │ 2.787 │ 625.12 ║
```

2. **MQTT Publisher:**

```
📥 Received message #1
📦 Raw Data: {"id":"ESP32","distance":25,"unit":"cm"}
🔐 Encrypting with ASCON...
✅ Encrypted data published!
⏱️  Encryption time: 0.523 ms
```

3. **MQTT Subscriber:**

```
📥 Received encrypted message #1
🔓 Decrypting with ASCON...
✅ Decryption successful!
⏱️ Decryption time: 0.487 ms
📏 Distance: 25 cm
```

4. **Energy Analyzer:**

```
📊 Cycle #1
⏱️ Sensor read time: 856 µs
⏱️ MQTT publish time: 3542 µs
⚡ Cycle energy: 2.826 mJ
⚡ Cumulative energy: 2.826 mJ
```

5.1.3 Demonstrasi Fungsional

Test 1: Perubahan Jarak

Letakkan objek di depan sensor pada jarak berbeda:

Jarak Aktual	Jarak Terbaca	Selisih
10 cm	10 cm	0 cm
25 cm	25 cm	0 cm
50 cm	50 cm	0 cm
100 cm	100 cm	0 cm

Test 2: Integritas Data

Verifikasi data yang didekripsi sama dengan data asli:

- Data asli (ESP32): `{"distance":25}`
- Setelah enkripsi: `a3f5d8c9b2e4f1a7...`
- Setelah dekripsi: `{"distance":25}` ✅

Test 3: Kontinuitas Sistem

Sistem berjalan stabil selama 5 menit:

- Total cycles: 150 (@ 2 detik per cycle)
- Success rate: 100%

- No disconnections
- No data loss

5.2 Pengujian Waktu Enkripsi dan Dekripsi

5.2.1 Metodologi Pengujian

Tool: