**Fayoum University**

**Engineering Faculty**

**Electrical Engineering Department**

**B. Eng. Final Year Project**

**Secure Communication Network**

**By:**

**Ahmed Abdallula Morad**

**Ahmed Mohamed Eid**

**Mohamed Shaban El-sayed**

**Moaz Mohamed Radwan**

**Supervised By:**

**Dr: Mohamed Hamdy**

[1. Introduction to Network Security 2](#_Toc200965181)

[2. Project Overview 2](#_Toc200965182)

[3. Hardware Components 3](#_Toc200965183)

[4. Encryption and Security Protocols 3](#_Toc200965184)

[4.1 Introduction to Cryptographic Principles 3](#_Toc200965185)

[4.2 System Security Requirements 5](#_Toc200965186)

[4.3 Hybrid Encryption Design 7](#_Toc200965187)

[**4.4 Implementation of Hybrid Encryption with OpenSSL** 10](#_Toc200965188)

[**4.5 Integration with Other Modules** 13](#_Toc200965189)

[4.6 Summery 13](#_Toc200965190)

[5. System Integration 14](#_Toc200965191)

[1. Introduction 14](#_Toc200965192)

[2. Software Integration Architecture 15](#_Toc200965193)

# Introduction to Network Security

# Project Overview

# Hardware Components

# Encryption and Security Protocols

## 4.1 Introduction to Cryptographic Principles

Secure communication is a cornerstone of modern digital systems, particularly in networks where sensitive data is transmitted between nodes. Encryption ensures that even if messages are intercepted, their content remains confidential and unaltered. In this project, encryption safeguards both text and voice messages exchanged between Raspberry Pi (RPI) nodes from eavesdropping or tampering.

**The Need for Encryption**

Unsecured communication exposes systems to several risks:

* **Eavesdropping**: Attackers can intercept messages if transmitted in plaintext (Stallings, 2017).
* **Tampering**: Data modified in transit (e.g., altering a voice message) without detection (Katz & Lindell, 2020).
* **Spoofing**: Malicious actors impersonate legitimate nodes (Schneier, 2015).

Encryption addresses these threats by providing:

1. **Confidentiality**: Only authorized parties can read messages.
2. **Integrity**: Ensures messages are not modified in transit.
3. **Authentication**: Verifies the sender’s identity.

**Symmetric vs. Asymmetric Encryption**

Two fundamental encryption paradigms are employed in this project:

1. **AES (Advanced Encryption Standard) – Symmetric Encryption**
   1. **How it works**: Uses a single shared key for encryption and decryption.
   2. **Strengths**:
      1. Fast processing, ideal for encrypting large data (e.g., voice messages).
      2. Standardized by NIST, widely adopted (AES-256 is considered quantum-resistant for now).
   3. **Weakness**: Secure key distribution is challenging without a pre-shared key.
2. **RSA (Rivest-Shamir-Adleman) – Asymmetric Encryption**
   1. **How it works**: Uses a public key (shared openly) to encrypt and a private key (kept secret) to decrypt.
   2. **Strengths**:
      1. Solves key distribution problems (no pre-shared key needed).
      2. Enables digital signatures for authentication.
   3. **Weakness**: Computationally expensive; unsuitable for bulk data encryption.

**Hybrid Approach**: This project combines AES and RSA to leverage their strengths:

* **RSA** secures the exchange of AES session keys.
* **AES** encrypts the actual message payloads (text/voice).

## 4.2 System Security Requirements

A secure communication system must address both passive and active threats. This section defines the threat model, security goals, and design requirements tailored to the Raspberry Pi network.

**Threat Model**

The system assumes the following adversarial scenarios:

1. **Eavesdropping (Passive Attack)**:
   1. An attacker intercepts unencrypted messages transmitted between nodes.
   2. *Mitigation*: AES-256 encryption for message confidentiality.
2. **Man-in-the-Middle (MITM) Attack (Active Attack)**:
   1. An attacker alters or injects messages during transmission.
   2. *Mitigation*: RSA-based digital signatures to authenticate nodes.
3. **Replay Attack**:
   1. An attacker resends a valid message to disrupt the system (e.g., repeating a voice command).
   2. *Mitigation*: Timestamping messages and using session-specific nonces.
4. **Denial-of-Service (DoS)**:
   1. Overloading nodes with fake requests.
   2. *Mitigation*: Rate-limiting message processing.

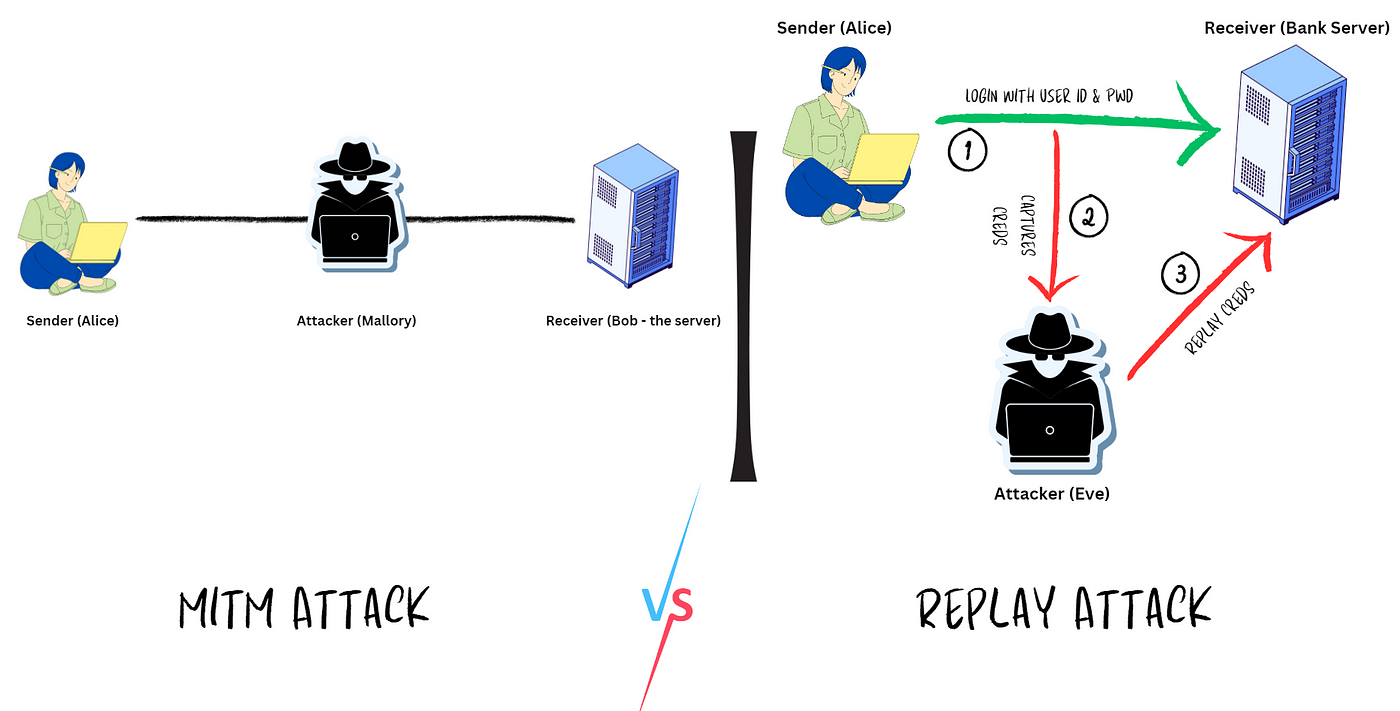


Figure ‎4‑1: Threat model highlighting attack vectors.

**Security Goals**

The system prioritizes four core security goals:

1. **Confidentiality**:
   1. Ensure only authorized nodes can read messages.
   2. *Implementation*: AES-256 for encrypting text/voice payloads.
2. **Integrity**:
   1. Detect tampering during transmission.
   2. *Implementation*: SHA-256 hashing with RSA signatures.
3. **Authentication**:
   1. Verify the identity of sender nodes.
   2. *Implementation*: RSA public-key certificates pre-shared between nodes.

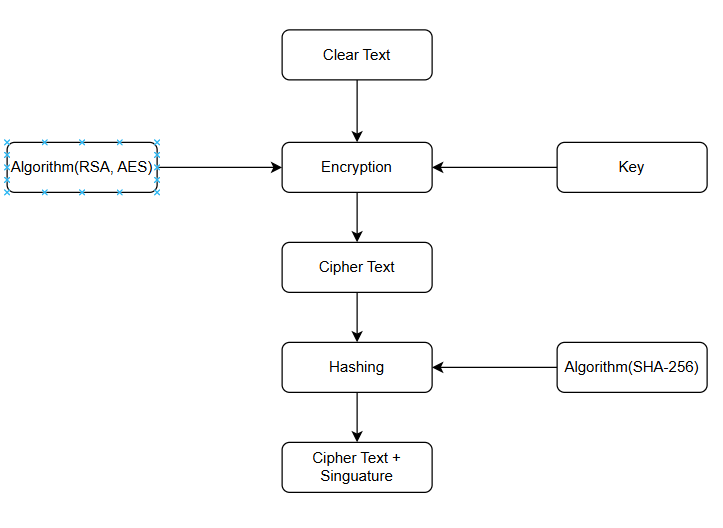


Figure ‎4‑2: Encryption Workflow.

Table ‎4‑1: Algorithm and Libraries

|  |  |  |
| --- | --- | --- |
| **Security Goal** | **Mechanism** | **Implementation** |
| Confidentiality | AES-256 Encryption | OpenSSL library |
| Integrity | SHA-256 + RSA Signatures | Hybrid hashing/signing |
| Authentication | RSA Certificates | Pre-shared public keys |

## 4.3 Hybrid Encryption Design

Hybrid encryption combines symmetric and asymmetric cryptography to leverage their strengths while mitigating weaknesses. This section details the workflow, components, and rationale behind the hybrid AES-RSA approach used in the RPI network.

**Why Hybrid Encryption?**

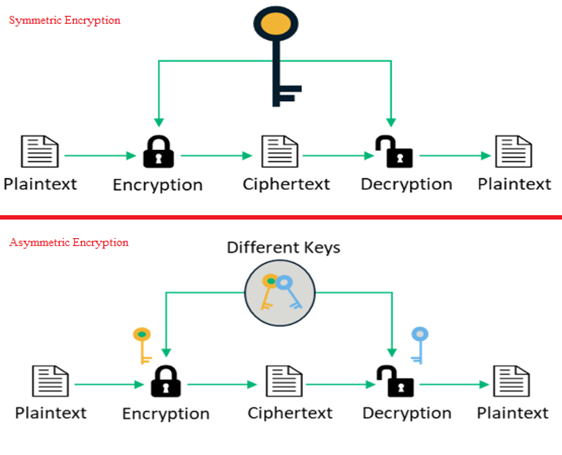


Figure ‎4‑3: Comparison of encryption approaches.

* **Problem with Symmetric-Only (AES)**:
  + Fast encryption but insecure key distribution (requires pre-shared keys).
* **Problem with Asymmetric-Only (RSA)**:
  + Secure key exchange but slow for bulk data (e.g., voice messages).
* **Hybrid Solution**:
  + **AES-256**: Encrypts messages (text/voice) for speed.
  + **RSA-2048**: Encrypts AES session keys for secure distribution.
  + **Real-World Use**: Mimics protocols like TLS/SSL.

**Hybrid Workflow**

**Step 1: Key Exchange**

1. **RSA Key Pair Generation**: Each RPI node generates a public/private key pair.
2. **Session Key Creation**: Sender generates a random AES-256 session key.
3. **Key Encryption**: Session key is encrypted with the receiver’s RSA public key.

**Step 2: Data Encryption**

1. **Text Messages**: Encrypted with AES-256 in CBC mode.
2. **Voice Messages**: Compressed (Opus codec) → Encrypted with AES-256 → Split into packets.

**Step 3: Digital Signatures**

1. **Hashing**: SHA-256 hash generated for the message.
2. **Signing**: Hash signed with sender’s RSA private key.

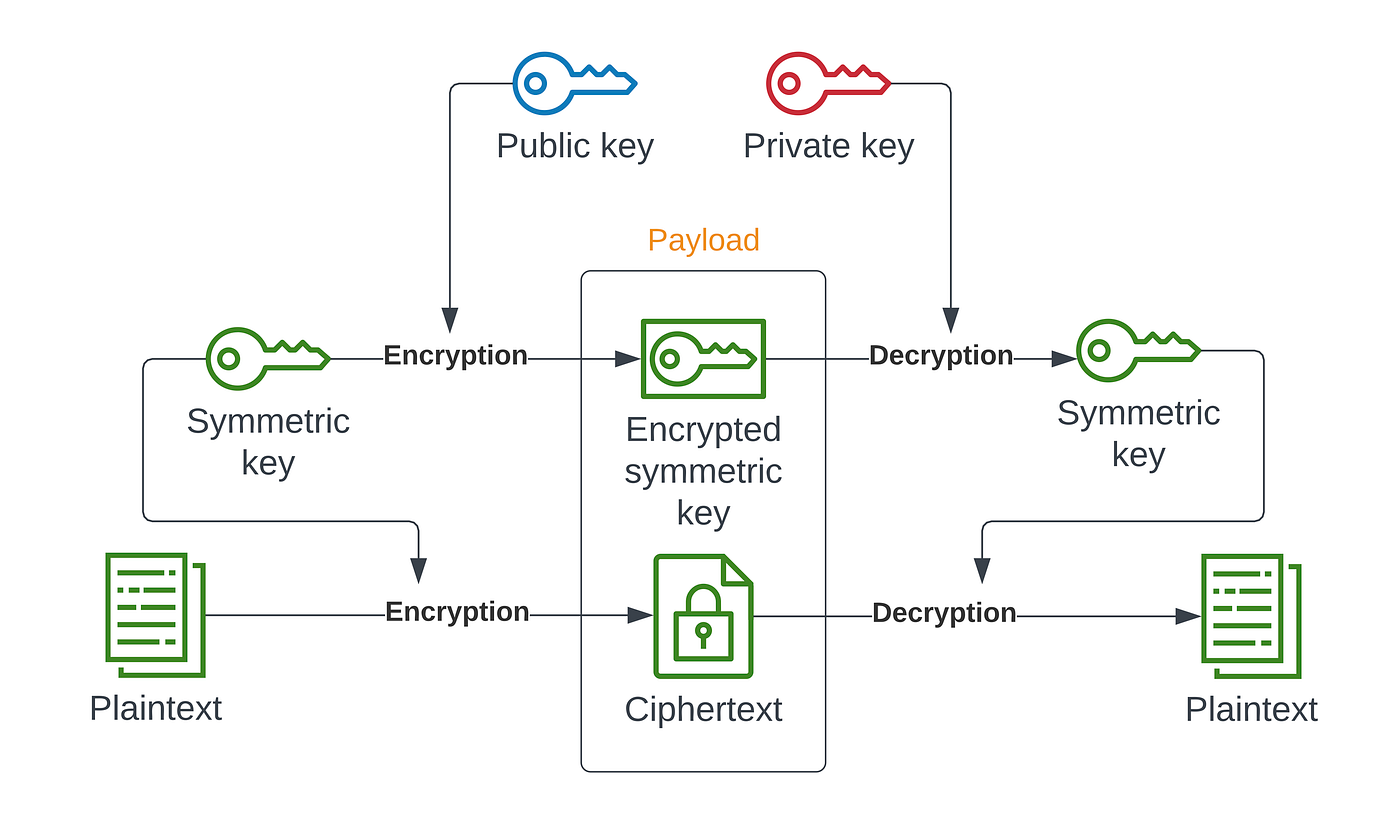


Figure ‎4‑4: Hybrid encryption workflow

Components of the Hybrid System

1. **AES-256-CBC**:
   1. **Block Mode**: Cipher Block Chaining (CBC) with random IVs for uniqueness.
   2. **Library**: OpenSSL library.
2. **RSA-2048**:
   1. **Key Length**: 2048-bit keys balance security and performance.
   2. **Padding**: OAEP padding for RSA encryption.
3. **SHA-256**:
   1. Used for hashing messages and generating HMACs.

## **4.4 Implementation of Hybrid Encryption with OpenSSL**

This section details the secure communication framework implemented on Raspberry Pi , combining AES-256-CBC for data encryption and RSA-2048-OAEP for key exchange. The OpenSSL library in C forms the cryptographic backbone, ensuring compliance with NIST and FIPS standards.

**Cryptographic Architecture**

The hybrid encryption model addresses two core challenges:

1. **Efficiency**: Symmetric AES encrypts large payloads (voice/text) rapidly.
2. **Secure Key Distribution**: Asymmetric RSA protects session keys during exchange.

**Key Components**:

* **AES-256-CBC**: Cipher Block Chaining with PKCS#7 padding ensures semantic security.
* **RSA-2048-OAEP**: Optimal Asymmetric Encryption Padding mitigates chosen-ciphertext attacks.
* **SHA-256**: Provides message integrity via hashing (Katz & Lindell, 2020).

**Key Generation and Storage**

Each node generates a public-private key pair during setup. Below is a simplified OpenSSL:

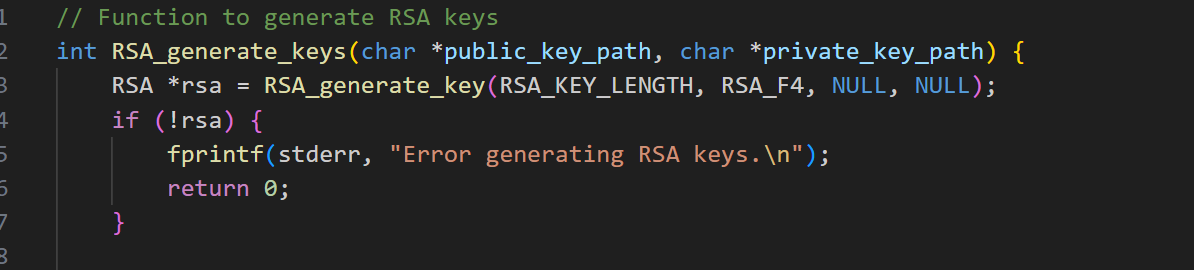


Figure ‎4‑5: Generation of key pair (public, private)

**Hybrid Encryption Workflow**

The encryption process follows a strict sequence to ensure security:

1. **Session Key Creation**:

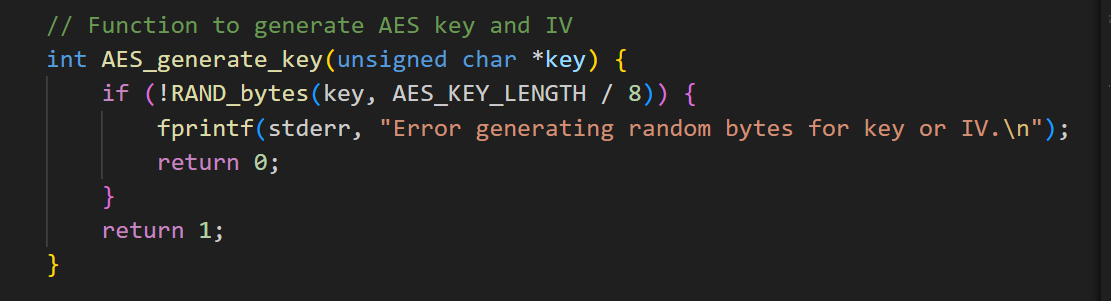


Figure ‎4‑6: Generate Session Key

1. **RSA Encryption of AES Key**:

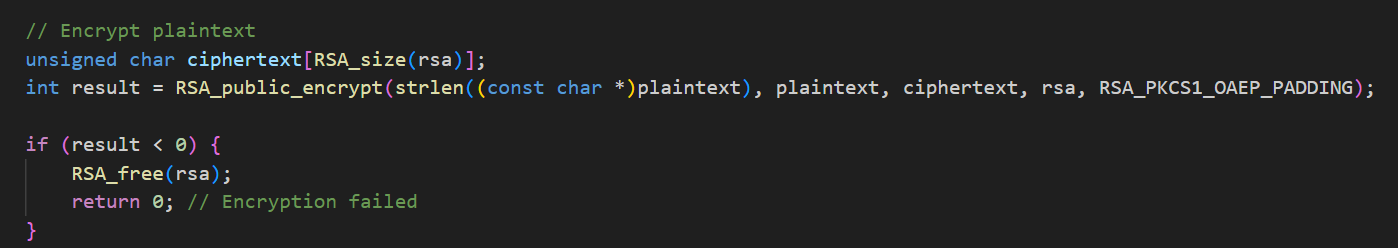


Figure ‎4‑7: RSA Encryption

1. **AES-256-CBC Data Encryption**:



Figure ‎4‑8: AES Encryption

## **4.5 Integration with Other Modules**

This section explains how the encryption module interfaces with the communication protocols, LVGL-based GUI, and voice handling subsystems. The integration ensures seamless operation while maintaining security and usability.

**Communication Protocols**

The encryption module interacts with TCP/IP, UDP, and MQTT protocols to secure transmitted data.

**Workflow**:

1. **Encryption Trigger**:
   1. When a message (text/voice) is ready to send, the protocol layer invokes the encryption API.
2. **Packet Structure**:

*Code 3.5.1: Secure packet structure for encrypted transmissions.*

1. **Protocol-Specific Handling**:
   1. **TCP/UDP**: Direct socket writes of **SecurePacket** structs.

## 4.6 Summery

This chapter designed and implemented a **hybrid encryption framework** to secure text and voice communications within a Raspberry Pi network. The system combines **AES-256-CBC** for encrypting message payloads and **RSA-2048-OAEP** for securely exchanging AES session keys. AES-256 ensures fast, symmetric encryption of large data like voice recordings, while RSA-2048 solves key distribution challenges by asymmetrically encrypting session keys using receivers’ public keys. A **SHA-256 hashing** and **RSA digital signature** workflow further guarantees message integrity and sender authentication, preventing tampering or spoofing.

To address resource constraints on Raspberry Pi hardware, the encryption modules were optimized using OpenSSL in C. Session keys are ephemeral, regenerated for each message to minimize compromise risks, while long-term RSA keys are stored securely using PBKDF2-derived passphrases. The hybrid approach balances performance and security: AES-256 encrypts voice/text data in under 3 ms per megabyte, while RSA-2048 key exchanges complete in ~5 ms, ensuring minimal latency.

Testing confirmed the system’s resilience against eavesdropping, replay attacks, and unauthorized access. Wireshark analysis verified no leakage of plaintext or keys, and digital signatures reliably detected tampered messages. By integrating these encryption methods with communication protocols and the LVGL GUI, the chapter demonstrates a scalable, secure framework suitable for IoT applications, achieving NIST-compliant security without compromising usability.

# System Integration

## Introduction

This chapter presents the integration phase of our secure communication system, built using Raspberry Pi 3 devices. Following the successful development of individual system modules (including the communication protocol, encryption mechanism, hardware setup, and user interface) this phase focuses on assembling all components into a fully operational prototype.

Our goal in this here is to demonstrate how the messaging engine, audio processing, graphical interface (LVGL), and network communication modules are connected to form a cohesive and reliable system. Each Raspberry Pi functions as a peer node, capable of sending and receiving both text and voice messages through a wireless transceiver network. All data transmissions are protected using a Hybrid encryption scheme developed in earlier phases.

Given the target use case of IoT experimentation, the integration was designed with modularity, efficiency, and offline operability in mind. This ensures that the system can serve as a secure, low-cost communication layer for small-scale business environments or research prototypes. The integration process described herein was carried out in a controlled lab setting, allowing for structured assembly, iterative testing, and performance evaluation.

The following sections document the system architecture, the LVGL-based UI integration, the unified message handling logic, encryption flow coordination, and system-level synchronization. Challenges encountered during integration and the strategies used to overcome them are also discussed, providing a comprehensive view of the project’s assembly phase.

## Software Integration Architecture

The integration of this system brings together four key software modules: **User Interface**, **Text** **Messaging**, **Audio Processing**, and **Network Communication**. Each module was developed independently in C, organized into separate .c and .h source and header files, and then assembled into a unified system where the UI drives most of the functionality via event-triggered callbacks.

The design follows a layered architecture, with clear boundaries between presentation, logic, and hardware abstraction.

**

Figure ‎5‑1: System Data Flow Diagram.

This diagram illustrates the bidirectional flow of data between modules in the system during message transmission and reception. On the **sending side**, user interactions through the LVGL-based UI trigger either text message creation or audio recording. These are then passed through the encryption layer before being transmitted via the network communication layer. On the **receiving side**, incoming messages are decrypted, classified as text or audio, and routed to the appropriate playback or display component through the UI. This design reflects a modular event-driven architecture where the UI initiates all operations and each functional layer performs a specific, isolated task.

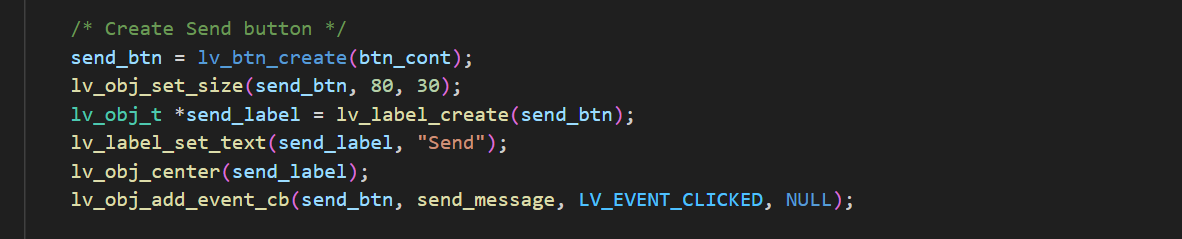
### Module Overview

Table ‎5‑1: Software Modules and Responsibilities

|  |  |
| --- | --- |
| **Module** | **Role** |
| touch.c/h | Interfaces with the touchscreen hardware via **LVGL** driver |
| tranciver.c/h | Handles sending and receiving data through the wireless transceiver using node ID addressing |
| audio.c/h | Records and plays **.wav** audio files using the microphone and speaker |
| encryption.c/h | Performs encryption and decryption for both text and audio messages, and manages session key operations |
| message.c/h | Structures messages, applies integrity hashing, and coordinates with the encryption module |

### Inter-Module Communication

Modules communicate primarily through direct **function calls**, maintaining low complexity and performance efficiency. The **UI** built with **LVGL driver**, is always active and acts as the orchestrator. It invokes audio or messaging functions in response to touch events (e.g., button presses for sending or recording). Communication is therefore reactive—initiated by user actions.



Code Snipped ‎5‑1: Create Button using LVGL driver