CHAPTER 1
INTRODUCTION

1.1 DESCRIPTION

This project report details the development of a secure wireless communication system that employs the RSA (Rivest–Shamir–Adleman) cryptographic algorithm to protect the confidentiality of transmitted text messages. The system is designed to facilitate communication between two devices, utilizing WiFi technology for wireless data transfer. At the heart of the system is the ATmega 328 microcontroller, a popular choice for embedded systems, which manages user input, performs RSA encryption and decryption, and controls the flow of data to and from a WiFi module (likely an ESP8266). The system incorporates a PS/2 keyboard for message input at the transmitter and an LCD display to provide a user interface and display communication status and messages at both the transmitter and (potentially) the receiver.

The primary goal of this project is to demonstrate a fundamental implementation of secure wireless communication using RSA, a widely recognized public-key cryptosystem. In today's interconnected world, the security of data transmitted wirelessly is of paramount importance. RSA offers a robust solution by employing a pair of mathematically linked keys: a public key for encryption and a private key for decryption. The public key can be freely distributed, while the private key remains secret, ensuring that only the intended recipient can decipher the message. This project explores the challenges and practical considerations of implementing RSA on a microcontroller platform, balancing security with the limited computational resources available.

The system's operation begins with the user composing a message using the PS/2 keyboard. The ATmega 328 microcontroller receives this input and then encrypts it using the RSA algorithm. The encrypted message is subsequently transmitted wirelessly to the receiving unit via the WiFi module. The LCD display provides real-time feedback to the user, showing the message being typed and system status information.

A critical aspect of RSA encryption is the generation and use of public and private key pairs. The setup() function in the provided code includes a section where the user selects between two predefined pairs of prime numbers (7, 19) or (11, 23). These prime numbers are fundamental to calculating the modulus (n) and the public and private exponents (e and d) used in the RSA algorithm. While the code snippet shows the selection of primes and hardcodes the resulting public/private keys, a complete RSA implementation would involve a more robust key generation process.

1.1.1 AIM

The overarching aim of this project is to develop a functional prototype for secure wireless communication, emphasizing the practical application of the RSA cryptographic algorithm within a microcontroller-driven system. This involves integrating hardware components and software implementations to achieve reliable and secure text-based data transfer.

Specifically, the project aims to:

- Design and construct a transmitter and receiver system capable of wireless communication. This includes selecting appropriate hardware components such as the ATmega 328 microcontroller, a WiFi module (likely ESP8266), a PS/2 keyboard for input on the transmitter side, and an LCD display for output on both sides. The system's architecture will be modular, allowing for potential expansion and modification in future iterations. The transmitter's architecture is shown in the diagram provided.
- Implement the RSA encryption and decryption algorithms on the ATmega 328 microcontrollers. This is a core objective, requiring careful consideration of the computational limitations of the microcontroller. The project will focus on implementing the essential mathematical operations of RSA, including modular exponentiation, prime number selection, and key generation (though the key generation is limited in the provided code).
- Establish reliable wireless communication between the transmitter and receiver using the WiFi modules. This involves configuring the modules for network connectivity and developing a communication protocol to ensure accurate data transfer. The code snippet shows the use of SoftwareSerial for communication with the ESP8266 and the sending of a 'mode' character for initial handshake.
- Develop a user-friendly interface for message input and output. The transmitter will utilize a PS/2 keyboard for users to input their messages, while both the transmitter and receiver will employ LCD displays to show the status of the communication, the messages being sent, and the messages received. The code demonstrates how the

Expanding on the core aims, the project further seeks to:

- Evaluate the feasibility and limitations of implementing RSA encryption on microcontrollers. This involves analyzing the computational overhead of RSA and determining the practical constraints on key sizes and message lengths. The project will explore optimization techniques to improve the efficiency of the RSA implementation within the microcontroller environment.

- Demonstrate the fundamental principles of public-key cryptography in a real-world application. By implementing RSA, the project will illustrate the concepts of key pairs (public and private keys), encryption, decryption, and the security advantages of asymmetric cryptography over symmetric cryptography.
- Provide a platform for further research and development in secure wireless communication. The project's modular design and clear implementation can serve as a foundation for future work, such as exploring more advanced encryption algorithms, improving the system's robustness, and integrating additional security features.
- Create an educational tool for understanding the practical aspects of cryptography and wireless communication. The project's design and implementation can be used to teach students and enthusiasts about the challenges and solutions involved in building secure communication systems. The provided code, with its detailed LCD output, can aid in visualizing the communication process.

1.1.2 OBJECTIVE

The development of a secure wireless communication system designed to transmit text messages using the RSA (Rivest–Shamir–Adleman) cryptographic algorithm. The system leverages the capabilities of the ATmega 328 microcontroller, commonly found in Arduino platforms, to manage user input, perform the computationally intensive RSA encryption and decryption processes, and facilitate wireless data transfer via a WiFi module, likely the ESP8266. The user interacts with the transmitter through a standard PS/2 keyboard for message entry, and both the transmitting and receiving units utilize Liquid Crystal Displays (LCDs) to provide essential feedback, display status information, and present the transmitted and received messages.

The core motivation behind this project stems from the ever-increasing demand for secure communication channels in a world heavily reliant on wireless technologies. Cryptography plays a pivotal role in ensuring the confidentiality and integrity of digital information exchanged over potentially insecure networks. RSA, as a widely recognized and robust public-key cryptosystem, offers a strong foundation for secure communication. Its fundamental principle lies in the asymmetry of the keys used for encryption and decryption: a public key for encryption, freely distributable, and a private key for decryption, kept secret by the recipient. This asymmetry ensures that only the intended recipient, possessing the correct private key, can decipher messages encrypted with the corresponding public key.

This project provides a practical exploration of implementing these cryptographic principles within the constraints of embedded systems. By utilizing the ATmega 328, a resource-limited microcontroller, the project aims to demonstrate the feasibility of integrating relatively complex cryptographic algorithms with wireless communication functionalities. The transmitter unit, whose block diagram has been provided separately, serves as the initial focal point of this development effort. It encompasses the necessary components for user interaction (PS/2 keyboard), data processing and security (ATmega 328 with RSA implementation), wireless connectivity (WiFi module), and user-centric feedback (LCD display). A stable power supply is crucial for the reliable operation of all these interconnected modules.

- The ATmega 328 microcontroller acts as the central processing unit of the communication system. Its responsibilities include:
- Handling User Input: The microcontroller interfaces directly with the PS/2 keyboard, capturing the keystrokes of the user composing a message intended for secure transmission.
- Executing RSA Encryption: The heart of the security mechanism lies in the RSA encryption algorithm implemented in the microcontroller's firmware. Upon receiving a complete message (signaled by pressing the ENTER key), the ATmega 328 applies the RSA encryption process to the message data using the designated public key. The mod_exp function, identified in the provided code snippets, is a critical element of this process, enabling efficient modular exponentiation, a core mathematical operation in RSA.
- Managing Wireless Data Transmission: Once the message is encrypted, the ATmega 328 communicates with the WiFi module (likely an ESP8266) to transmit the ciphertext wirelessly to the intended recipient. The code snippets indicate the use of the SoftwareSerial library for asynchronous serial communication with the ESP8266 on specific digital pins.
- Providing User Interface Elements: The LCD display, connected to the microcontroller via a set of defined digital and analog pins, serves as the primary interface for providing feedback to the user. This includes displaying initial system messages, prompts, the characters being typed, and status updates related to the encryption and transmission processes.

1.1.3 OVERVIEW

The "WiFi Based Secure Wireless Communication Using RSA" project aims to establish a system for secure wireless exchange of text messages by implementing the RSA encryption algorithm. The system utilizes an ATmega 328 microcontroller to manage user input, perform RSA encryption and decryption, and control wireless communication via a WiFi module.

Transmitter Unit:

- Input: A PS/2 keyboard allows the user to enter the message to be sent.
- Processing: The ATmega 328 microcontroller:
    - Reads the input from the keyboard.
    - Encrypts the message using the RSA algorithm.
    - Controls the LCD display to provide user feedback.
    - Communicates with the WiFi module.
- Wireless Communication: A WiFi module (likely ESP8266) transmits the encrypted message wirelessly.
- Output: An LCD display shows the message being typed and system status.
- Power: The system is powered by a power supply, with a rectifier and voltage regulator to provide stable DC power.
- RSA Encryption/Decryption:
    - The RSA algorithm is implemented for secure communication.
    - The code includes a mod_exp() function for modular exponentiation, a core operation in RSA.
    - The system uses a public key to encrypt messages and a private key to decrypt them.
    - The user selects from predefined prime number pairs to set up the RSA keys.
- Communication Flow:
    - The user enters a message using the PS/2 keyboard.
    - The ATmega 328 encrypts the message.
    - The encrypted message is sent via the WiFi module.
    - The receiving unit receives the encrypted message, decrypts it using the private key, and displays the original message.
- Software and Control:
    - Arduino libraries are used for hardware control (LCD, keyboard, serial communication).

CHAPTER 2

LITERATURE SURVEY

2.1 PAPERS REFERRED

1.G. Leelavathi et al., "End-to-End Intelligent Security Model for WSN" - Wireless Personal Communications, 2024

The paper "End-to-End Intelligent Security Model for WSN" presents an intelligent security framework for Wireless Sensor Networks (WSNs), aimed at enhancing data security and network integrity. It integrates machine learning and AI techniques for anomaly detection and predictive threat identification to protect against attacks like data breaches and node compromise. The model incorporates encryption and authentication protocols for secure communication. Designed to be adaptive, it ensures scalable protection for growing WSNs, making it suitable for applications such as smart cities and military surveillance. The research highlights the effectiveness of combining AI- driven systems with traditional security measures.

2.Al-Shamri & Safwan, "Secure Image Transmission Using RSA and OFDMA" - EURASIP Journal on Image and Video Processing, 2024

This paper presents a secure image transmission method combining RSA encryption and OFDMA (Orthogonal Frequency Division Multiple Access) to protect image data during wireless transmission. The authors propose using RSA encryption for securing the image data and OFDMA for efficient and secure transmission over wireless channels. RSA is used to encrypt the image before transmission, ensuring confidentiality, while OFDMA enhances the transmission efficiency by enabling multiple users to share the same bandwidth. The system improves security against eavesdropping and performance in terms of bandwidth utilization. This technique is particularly useful for image-based communication in wireless networks.

3.H. Sharma, M. Singh -"RSA Algorithm in Wireless Communication: A Comparative Study with Elliptic Curve Cryptography" – 2023

The paper compares RSA and Elliptic Curve Cryptography (ECC) in wireless communication, focusing on key size, encryption speed, and computational efficiency.. ECC provides equivalent security with smaller keys, making it more efficient in resource-constrained environments. The study concludes that while ECC is better suited for modern, lightweight systems, RSA remains relevant for its reliability and compatibility in secure communication where performance trade-offs are acceptable.

4.P. Soni, J. Gupta - "Securing IoT Communication Using RSA and ECC" – 2023 The paper explores the implementation of RSA and ECC in securing communication for IoT-based systems, focusing on constrained devices. It examines how these encryption methods ensure secure data transmission over wireless networks despite resource limitations. The study highlights the trade-offs between RSA's larger key size and computational demands versus ECC's efficiency and smaller keys. It offers practical insights into adopting RSA for devices like ATmega and ESP8266, emphasizing its effectiveness in maintaining security in IoT communication systems.

5.R. Choudhury, M. Sen - "Implementing RSA Encryption in Embedded Systems for Secure Wireless Communication" – 2022

The paper examines the implementation of RSA encryption in embedded systems for secure wireless communication. It addresses the computational challenges of RSA and proposes optimization techniques for microcontrollers such as the ATmega328. The research demonstrates how to balance security and performance, making RSA a viable option for embedded systems. This study offers practical guidance relevant to projects utilizing ATmega microcontrollers for secure communication.

6.S. S. Kaur, G. P. Singh - "Hybrid Encryption Using RSA and AES for Secure Wireless Communication" – 2021

This paper explores the use of hybrid encryption by combining RSA with AES for secure wireless communication. The integration leverages RSA for secure key exchange and AES for fast data encryption, achieving a balance between high security and performance. The study demonstrates the suitability of this approach for resource- constrained environments, providing enhanced efficiency in wireless applications. It offers valuable insights for projects considering hybrid encryption methods like RSA and AES for efficient and secure communication.

7."New RSA Encryption Mechanism Using One-Time Keys" - MDPI Electronics, 2020

The paper "New RSA Encryption Mechanism Using One-Time Keys" introduces a modified RSA encryption system that employs one-time keys for each encryption session. Unlike traditional RSA, which uses fixed keys, the one-time key mechanism generates unique keys for each operation, improving security by preventing key reuse attacks. This approach enhances confidentiality, as compromised keys cannot be reused. The paper outlines the process of generating and exchanging one-time keys securely, providing a more robust encryption method for secure communication, particularly in applications like banking and military systems, where data confidentiality is critical.

8.Reena R -"Wi-Fi Based Secure Wireless Communication Using RSA" - IJAER, 2019

This paper explores the RSA algorithm, a public-key cryptosystem used for secure communication. It details the process of key generation, encryption, and decryption, where the public key encrypts data and the private key decrypts it. RSA's security relies on the difficulty of factoring large numbers. The paper highlights the challenges in implementing RSA, such as computational complexity, and discusses its application in secure communications, digital signatures, and online authentication. It also notes that

RSA is often used to encrypt small data (e.g., keys), while bulk data encryption is handled by faster symmetric encryption methods.

9.“Improved RSA Encryption Algorithm for Wireless Networks” - IEEE 2017

This study proposes an improved RSA encryption algorithm designed to address the computational and security challenges of traditional RSA in wireless networks. The enhancements aim to optimize encryption speed and reduce resource consumption while maintaining strong security against attacks like eavesdropping and data tampering. The modified algorithm improves efficiency for devices with limited processing power, making it suitable for wireless environments. The findings demonstrate the proposed algorithm's effectiveness in balancing security and performance, ensuring reliable and secure communication in wireless networks.

10."Research and Implementation of RSA Algorithm for Encryption and Decryption" - IEEE, 2012

The paper titled "Wi-Fi Based Secure Wireless Communication Using RSA" by Reena R explores the integration of RSA encryption to secure wireless communication over Wi-Fi. It focuses on ensuring data confidentiality, integrity, and protection against unauthorized access. By leveraging RSA's public-key cryptography, the system encrypts transmitted data, making it resilient to threats like eavesdropping and man-in- the-middle attacks. The study evaluates encryption performance, highlighting a trade- off between security strength and computational efficiency. The findings confirm RSA's effectiveness in enhancing Wi-Fi communication security, making it suitable for applications requiring robust data protection in wireless environments.

2.2 APPLICATIONS REFERRED

Military & aerospace communications – secure and tamper-proof wireless communication is critical.

- Industrial automation – ensures data integrity across automated systems.
- Embedded systems – enables low-cost, microcontroller-based secure communication between devices.
- IoT & wireless messaging – safeguards small data transmissions over Wi-Fi using RSA encryption.

2.3 EXISTING SYSTEM

- Existing wireless communication systems use:
- Open or weak encryption methods, often vulnerable to eavesdropping or tampering.
- General Wi-Fi modules without integrated security mechanisms.
- Limited key management, where symmetric key exchange is challenging and insecure in ad-hoc networks.

2.3.1 DRAWBACK OF EXISTING SYSTEM

- Lack of end-to-end encryption: Data is vulnerable during transit.
- Insecure key sharing: Symmetric encryption requires pre-shared keys, which is risky.
- Vulnerability to sniffing attacks: Data transmitted over open Wi-Fi can be intercepted.
- No authentication mechanism: Any device can connect and access data if not secured properly.

2.4 PROPOSED SYSTEM

- Utilizes RSA encryption for message security.
- Employs Atmega microcontroller and XBee module for wireless transmission.
- Enables two-way encrypted communication, allowing messages to be sent and decrypted only with the correct private key.
- Displays messages via LCDs and takes input using USB keyboards.
- Ensures data privacy: incorrect keys result in unreadable (garbage) output.
- This design strengthens wireless communication in embedded systems using proven asymmetric cryptography.

CHAPTER 3
SOFTWARE REQUIREMENT SPECIFICATIONS

3.1 TECHNICAL FEASIBILITY

1. Hardware Component Suitability:

- ATmega 328 Microcontroller: The ATmega 328 is a widely used and well-supported microcontroller. It is suitable for handling basic input/output operations, controlling peripherals like the LCD and interfacing with the PS/2 keyboard. However, it has limited processing power and memory, which poses a challenge for computationally intensive tasks like RSA encryption/decryption.
- WiFi Module (ESP8266): The ESP8266 is a cost-effective and readily available WiFi module. It provides robust WiFi connectivity and simplifies the task of wireless communication. Interfacing the ESP8266 with the ATmega 328 using SoftwareSerial is a standard practice, although it can introduce timing constraints and potential performance bottlenecks.
- PS/2 Keyboard: Using a PS/2 keyboard is a feasible and straightforward way to provide user input. The PS2Keyboard library simplifies the process of reading keystrokes.
- LCD Display: The LCD display is suitable for providing basic output and feedback to the user. The LiquidCrystal library makes it easy to control the LCD.
- Power Supply, Rectifier, Voltage Regulator: These are standard components and their feasibility is well-established for powering electronic circuits.

2. RSA Implementation Challenges:

- Computational Complexity: RSA involves complex mathematical operations, particularly modular exponentiation, which can be computationally expensive. The ATmega 328's limited processing power can lead to slow encryption and decryption times, especially with larger key sizes.
- Memory Constraints: RSA requires storing large numbers (the keys and intermediate values). The ATmega 328 has limited RAM, which can restrict the maximum key size that can be used.
- Code Optimization: Efficient implementation of the mod_exp function and other RSA operations is crucial to minimize processing time and memory usage. The provided code snippet includes a mod_exp function, which is a good starting point, but further optimization might be necessary.
- Key Generation: The feasibility of generating RSA keys directly on the ATmega 328 is questionable due to the need for random number generation and primality testing, both of which

are computationally intensive. The project seems to sidestep this by pre-selecting prime number pairs, which is a simplification.

3. Software and Communication Feasibility:

- Arduino Libraries: The use of Arduino libraries simplifies the development process and ensures compatibility with the hardware components.
- Serial Communication: Communicating between the ATmega 328 and the ESP8266 using SoftwareSerial is feasible but can be a bottleneck. The data transfer rate might be limited, affecting the overall speed of the communication.
- WiFi Communication: The ESP8266 provides reliable WiFi connectivity, making wireless communication feasible. However, proper network configuration and handling potential connection issues are essential.

4. Security Considerations:

- Key Size: The choice of key size is critical for security. Smaller key sizes (e.g., 512 bits or less) might be feasible for the ATmega 328 but offer limited security against modern attacks. Larger key sizes (e.g., 1024 bits or 2048 bits) are more secure but significantly increase the computational burden. The provided code pre-selects very small primes, which is insecure for real-world applications.
- Implementation Security: The security of the system depends heavily on the correct and secure implementation of the RSA algorithm. Any vulnerabilities in the code can compromise the security of the communication.
- Side-Channel Attacks: Microcontrollers are vulnerable to side-channel attacks, where attackers can extract information by analyzing power consumption, timing, or electromagnetic emissions. Mitigating these attacks adds complexity.
- Implementing basic wireless text communication using the ATmega 328, ESP8266, PS/2 keyboard, and LCD is achievable.
- Implementing RSA encryption and decryption on the ATmega 328 is feasible for demonstration purposes, but with significant limitations on key size and performance. The provided code snippet demonstrates the core RSA operations.
- Achieving strong security with RSA on the ATmega 328 is challenging due to the microcontroller's limitations. The project would likely need to compromise on key size and might not be suitable for real-world secure communication.
- The use of FreeRTOS for task management is a positive aspect and enhances the system's architecture.
- Recommendations to Enhance Feasibility:

- Optimize RSA Implementation: Focus on optimizing the mod_exp function and other RSA operations for the ATmega 328.
- Key Management: Implement a more secure key exchange mechanism, potentially offloading key generation to a more powerful device.
- Hardware Acceleration: Consider using microcontrollers with hardware acceleration for cryptographic operations (e.g., some ARM Cortex-M4 processors) to improve performance.
- Alternative Cryptography: Explore alternative, less computationally intensive asymmetric encryption algorithms suitable for microcontrollers, such as Elliptic Curve Cryptography (ECC).
- Security Audits: Conduct thorough security audits of the code to identify and mitigate potential vulnerabilities.

3.2 ECONOMICAL FEASIBILITY

The economic feasibility of the "WiFi Based Secure Wireless Communication Using RSA" project depends on various factors, including the cost of components, development time, potential applications, and the availability of resources. Considering the nature of the project as a prototype or educational tool, a full-scale commercial viability analysis might not be directly applicable. However, we can assess its economic aspects based on the resources required and potential benefits.

1. Component Costs:

- ATmega 328 Microcontroller: Relatively inexpensive and widely available, especially within Arduino development kits. The cost per unit is low, making it economically viable for prototyping and small-scale deployments.
- WiFi Module (ESP8266): Also a low-cost module that provides significant wireless connectivity capabilities. Its affordability makes it a popular choice for IoT and hobbyist projects.
- PS/2 Keyboard: Standard PS/2 keyboards are generally inexpensive and readily available, often salvaged from older systems, further reducing costs.
- LCD Display: Basic character LCDs are relatively inexpensive and sufficient for the project's user interface requirements.
- Power Supply Components (Rectifier, Voltage Regulator): These are standard electronic components with low individual costs.
- The overall cost of the hardware components for a single unit is likely to be quite low, making it economically feasible for individual developers, educational institutions, or small research teams to build and experiment with the system.

2. Development Costs:

- Software Development: The primary development cost lies in the time and effort required to write and debug the Arduino code for handling keyboard input, implementing the RSA algorithm (encryption and decryption), managing communication with the WiFi module, and controlling the LCD. The use of the Arduino IDE and readily available libraries (e.g., PS2Keyboard, LiquidCrystal, SoftwareSerial, and potentially PainlessMesh and FreeRTOS) helps to reduce the development time and complexity.
- Prototyping and Testing: Costs associated with building the physical prototype, including breadboards, wires, and testing equipment, are generally low for a project of this scale.

3. Potential Applications and Economic Benefits:

- Educational Tool: The project serves as an excellent educational tool for demonstrating the principles of cryptography and wireless communication. This can have economic benefits for educational institutions by providing hands-on learning experiences without significant investment.
- Proof of Concept: It can act as a proof of concept for more complex secure communication systems in resource-constrained environments. This could potentially attract funding or interest for further development in specific niche applications.
- Niche Applications: In scenarios where low-bandwidth, secure communication is required and cost is a major constraint, a simplified system like this might find niche applications. However, the security limitations of RSA with small key sizes on a microcontroller would need to be carefully considered.
- Skill Development: The project helps in developing valuable skills in embedded systems programming, cryptography, and wireless communication, which can lead to better employment opportunities for the developers involved.

4. Economic Risks and Challenges:

- Security Limitations: Implementing robust RSA security on the ATmega 328 with limited resources is a significant challenge. Using small key sizes for feasibility compromises the security of the communication, limiting its applicability in real-world secure scenarios.
- Performance Limitations: The computational overhead of RSA on the ATmega 328 can lead to slow encryption and decryption times, making it unsuitable for applications requiring high-speed communication.
- Scalability: Scaling the system for a large number of users or devices might present economic and technical challenges related to network infrastructure, key management, and processing power.
- Competition: The market for secure communication systems is competitive, with many established and more robust solutions available. A basic prototype like this would likely not be

economically competitive for mainstream applications without significant further development and security enhancements.

3.3 MODULE DESCRIPTION

1. Input Module: PS/2 Keyboard

- Description: This module consists of a standard PS/2 keyboard that allows the user to input the text message they wish to transmit.
- Functionality: The keyboard sends signals to the ATmega 328 microcontroller representing the pressed keys. The PS2Keyboard library is used to read and interpret these signals.
- Code Reference: The code initializes the keyboard with keyboard.begin(DataPin, IRQpin); and reads input using keyboard.read().

2. Processing and Encryption Module: ATmega 328 Microcontroller

- Description: This is the central processing unit of the transmitter, responsible for controlling the overall operation of the system, handling user input, encrypting the message, and communicating with the WiFi module and the LCD.
- Functionality:
  - Reads the message from the PS/2 keyboard.
  - Implements the RSA encryption algorithm to encrypt the message. The mod_exp() function is used for modular exponentiation, a core part of RSA.
  - Sends the encrypted message to the WiFi module for transmission.
  - Controls the LCD to display messages and status information.
- Code Reference: The loop() function handles keyboard input and LCD output, and the sendMessage() function performs the RSA encryption and sends the data.

3. Wireless Communication Module: WiFi Module (ESP8266)

- Description: This module handles the wireless transmission of the encrypted data. Based on the code, it's likely an ESP8266.
- Functionality:
  - Receives the encrypted message from the ATmega 328.
  - Transmits the encrypted message over a WiFi network.
  - Receives data from the WiFi network.
  - Sends received data to the Atmega 328.
- Code Reference: The SoftwareSerial library is used to communicate with the ESP8266, and the esp.println() and esp.readStringUntil() functions are used to send and receive data. The code also suggests the use of a mesh network.

4. Output Module: LCD Display

- Description: This module provides a visual interface for the user, displaying messages, prompts, and the status of the communication.
- Functionality:
    - Displays a startup message and prompts for selecting prime numbers.
    - Displays the message being typed by the user.
    - Displays received messages.
    - Shows status messages, such as "sending message" or "message sent".
- Code Reference: The LiquidCrystal library is used to control the LCD, and functions like lcd.print() and lcd.setCursor() are used to display text at specific locations on the screen.

5. Power Supply Module

- Description: This module provides the necessary power to all the electronic components of the system.
- Functionality:
    - The diagram shows a rectifier and a voltage regulator, indicating that this module likely converts AC power to a stable DC voltage required by the components.
- Code Reference: The code doesn't explicitly show power supply management, as it's a hardware function.

3.4 FUNCTIONAL REQUIREMENTS

1. User Input:

- The system must allow the user to input text messages using a PS/2 keyboard connected to the transmitter unit.
- The system must display the characters typed by the user on the LCD screen of the transmitter in real-time.
- the user must be able to indicate the completion of a message by pressing the enter key.
- the system should allow the user to delete characters using the backspace key.

2. RSA Encryption:

- the transmitter unit must implement the RSA encryption algorithm.
- Upon receiving a complete message (via ENTER key press), the transmitter must encrypt each character of the message using a pre-defined public key (e) and modulus (n).
- The encryption process must involve raising the ASCII value of each character to the power of the public key exponent (e) modulo the modulus (n).

- The system must have a mechanism to select or define the public key (e) and modulus (n). The setup() function suggests a choice between two sets of prime numbers that would be used to derive these keys (though the derivation isn't fully shown).

3. Wireless Transmission:

- The transmitter unit must establish a wireless connection using a WiFi module (likely ESP8266).
- The transmitter must transmit the encrypted message wirelessly to a designated receiver unit. The code suggests broadcasting the message over a mesh network.
- The system must have a mechanism for initial communication and potentially handshaking between the transmitter and receiver (as suggested by the 'mode' transmission in setup()).

4. Wireless Reception:

- The receiver unit must be able to receive wirelessly transmitted data from the transmitter.
- The receiver unit must provide a mechanism to receive and store the incoming encrypted data.

5. RSA Decryption:

- The receiver unit must implement the RSA decryption algorithm.
- Upon receiving an encrypted message, the receiver must decrypt each character using a pre-defined private key (d) and the same modulus (n) used for encryption.
- The decryption process must involve raising the received encrypted character value to the power of the private key exponent (d) modulo the modulus (n).
- The system must have a mechanism to define the private key (d) and modulus (n), corresponding to the public key used for encryption.

6. Output Display:

- The transmitter unit must display the message being typed by the user on its LCD screen.
- The transmitter unit should provide visual feedback on the status of the transmission (e.g., "sending...", "sent").
- The receiver unit must display the decrypted message on its connected output (likely a serial monitor as indicated by Serial.println(decrypted_data); in receivedCallback()). The system should ideally also display the received message on an LCD if one is connected to the receiver.

7. System Initialization:

- Upon startup, the transmitter MUST initialize all necessary hardware components (LCD, keyboard, WiFi module).
- The transmitter must present a user interface (via LCD) to select or indicate the desired RSA key parameters (based on the prime number selection in setup()).
- The transmitter must establish communication with the WiFi network.

3.5 NON-FUNCTIONAL REQUIREMENTS

1. Performance:

- Encryption/Decryption Speed: The system should encrypt and decrypt text messages within a reasonable timeframe for interactive communication, although this will be limited by the processing power of the ATmega 328. A target latency for encrypting/decrypting a short message (e.g., up to 32 characters) should be within a few seconds.
- Wireless Communication Latency: The latency for transmitting and receiving encrypted messages over the WiFi network should be kept to a minimum to ensure a relatively smooth communication experience. Target latency for sending and receiving a short message should ideally be under one second.
- System Responsiveness: The transmitter unit should remain responsive to user input (keyboard presses) even during encryption and transmission processes. The use of FreeRTOS suggests an attempt to manage this.
- Power Consumption: The system should operate within the power limitations of a typical low-power embedded system, especially if battery-powered operation is envisioned in the future.

2. Security:

- RSA Implementation Strength: The system should implement the RSA algorithm with a key size that provides a reasonable level of security against basic eavesdropping. However, given the limitations of the ATmega 328, achieving high levels of security with large key sizes might not be feasible. The project should aim for the largest practical key size that the microcontroller can handle within acceptable performance limits. Note: The hardcoded small prime numbers in the setup() for key selection in the provided code severely limit security and are only suitable for basic demonstration.
- Key Management: The system should have a mechanism for securely managing the public and private RSA keys. For this basic prototype, keys might be pre-configured or generated through a simplified process. A more robust system would require secure key storage and exchange mechanisms.
- Resistance to Basic Attacks: The system should be designed to resist basic eavesdropping and replay attacks on the wireless communication channel. Encryption with RSA is intended to address eavesdropping.
- Data Integrity: The system should ensure the integrity of the transmitted data, although the provided code doesn't explicitly include mechanisms for this beyond encryption.

3. Reliability:

- Wireless Connection Stability: The wireless connection between the transmitter and receiver should be stable and maintain connectivity for the duration of the communication session.
- Data Transmission Accuracy: The system should ensure that transmitted encrypted data is received accurately without corruption.
- System Uptime: The system should be able to operate continuously for a reasonable period without failures.
- Error Handling: The system should include basic error handling mechanisms to manage potential issues such as WiFi connection failures or data transmission errors.

4. Availability:

- System Availability: The transmitter and receiver units should be readily available for use when powered on.
- Resource Availability: The necessary hardware components (ATmega 328, ESP8266, keyboard, LCD) are generally widely available.
- Ease of Deployment: The system should be relatively easy to set up and deploy for its intended purpose (e.g., educational demonstration).
- It's important to note that achieving high levels in all these non-functional requirements simultaneously can be challenging, especially on a resource-constrained platform like the ATmega 328. Trade-offs might be necessary between performance, security, and resource usage.

CHAPTER 4
SYSTEM AND DETAILED DESIGN

4.1 ARCHITECTURE

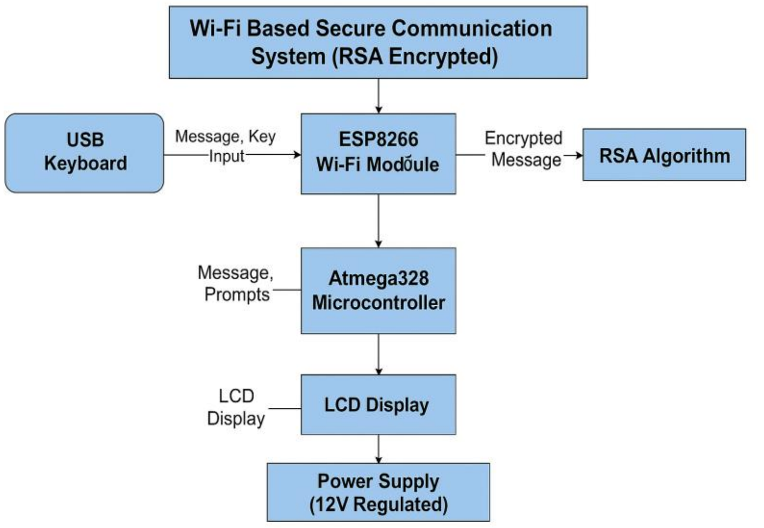


Fig 4.1.1: Architecture Diagram

The Figure 4.1.1 depicts a system for secure wireless communication using RSA encryption. It shows data flowing from a USB keyboard, through an ATmega328 microcontroller, to an ESP8266 WiFi module, with an LCD for display.

4.2 CONTEXT DIAGRAM

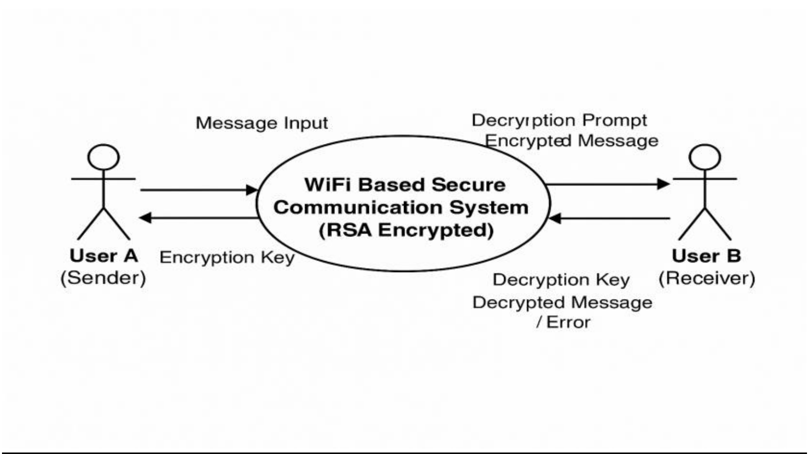


Fig 4.2.1: Context Diagram

The Figure 4.2.1 shows a use case diagram for a WiFi-based secure communication system using RSA encryption. User A (Sender) inputs a message and an encryption key, which the system uses to send an encrypted message to User B (Receiver), who uses a decryption key to read the original message.

4.3 DATAFLOW DIAGRAM

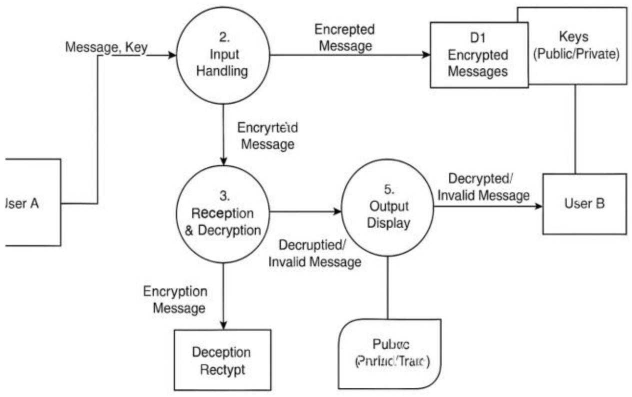


Fig 4.3.1: Dataflow Diagram

The Figure 4.3.1 illustrates the flow of data in a secure communication system where User A sends a message and key to Input Handling, resulting in an encrypted message that is eventually decrypted and displayed to User B after passing through Reception & Decryption.

4.4 USECASE DIAGRAM

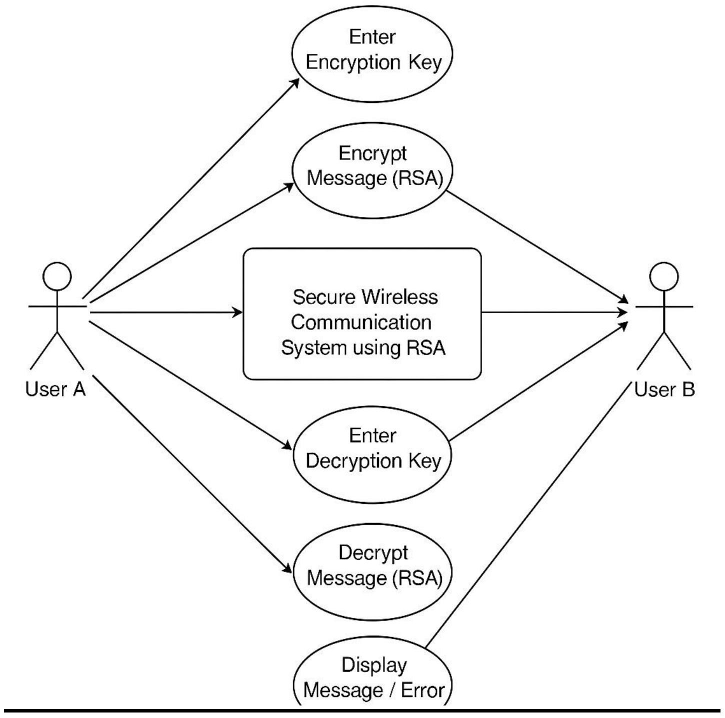


Fig 4.4.1: Use case Diagram

The Figure 4.4.1 depicts a system for secure wireless communication using RSA encryption. It shows data flowing from a USB keyboard, through an ATmega328 microcontroller, to an ESP8266 WiFi module, with an LCD for display.

4.5 SEQUENCE DIAGRAM

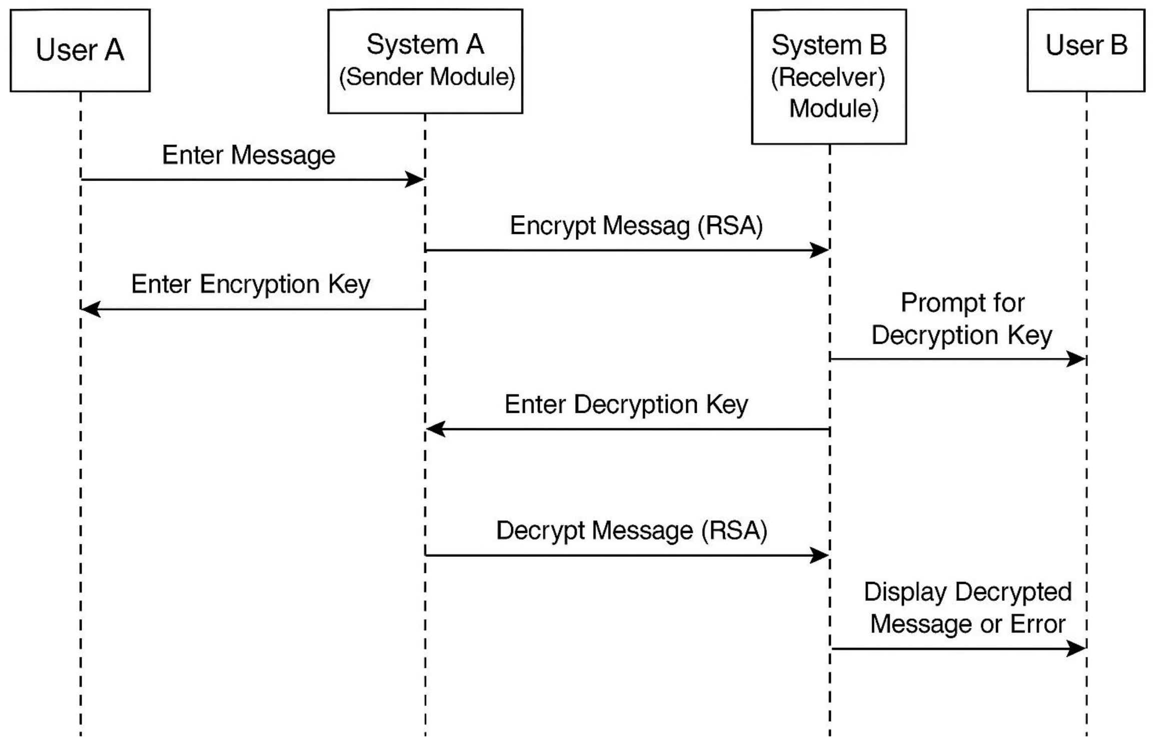


Fig 4.5.1: Sequence Diagram

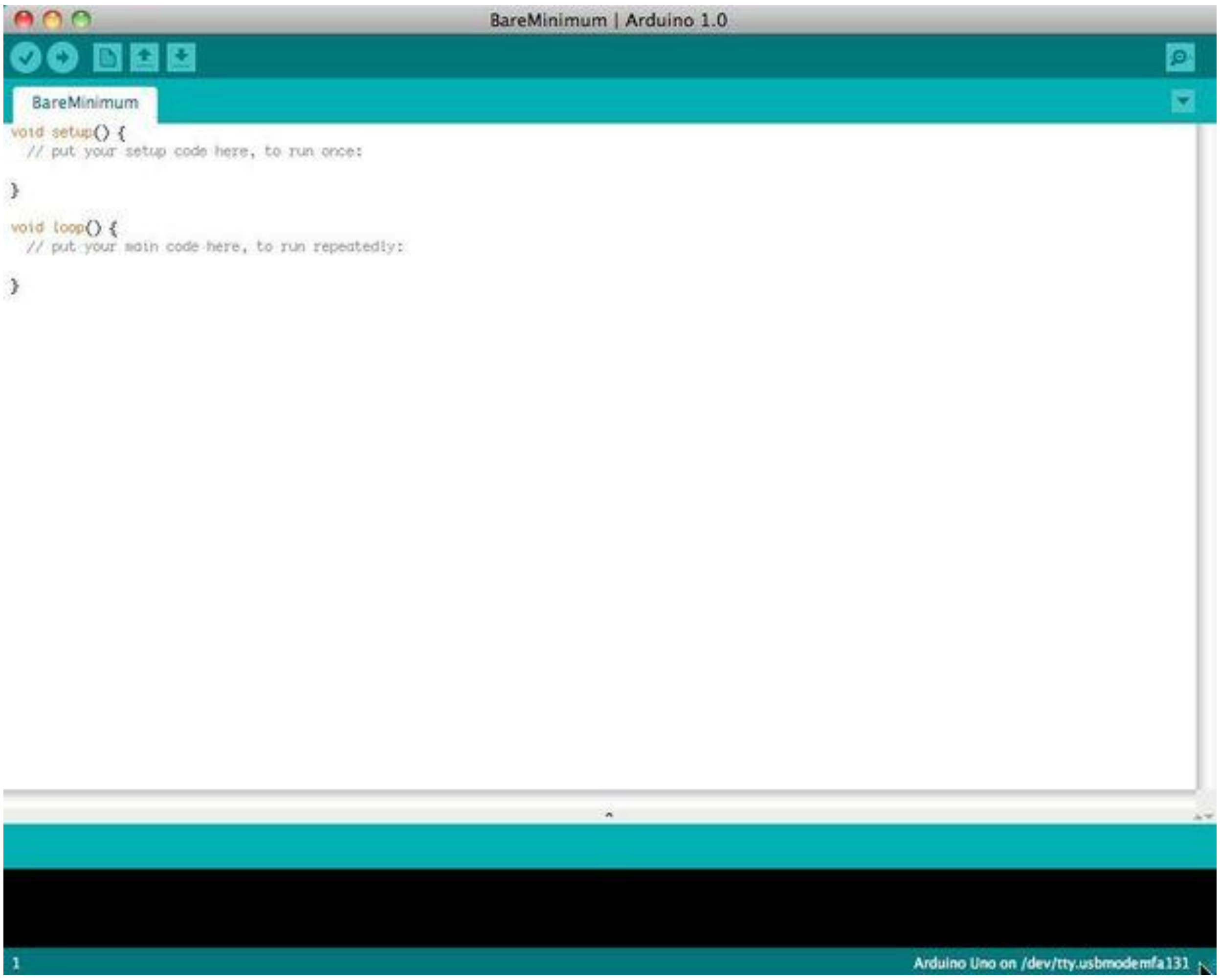
The Figure 4.5.1 illustrates the flow of data in a secure communication system where User A sends a message and key to Input Handling, resulting in an encrypted message that is eventually decrypted and displayed to User B after passing through Reception & Decryption.

CHAPTER 5

IMPLEMENTATION

5.1 CODE SNIPPETS

Arduino IDE

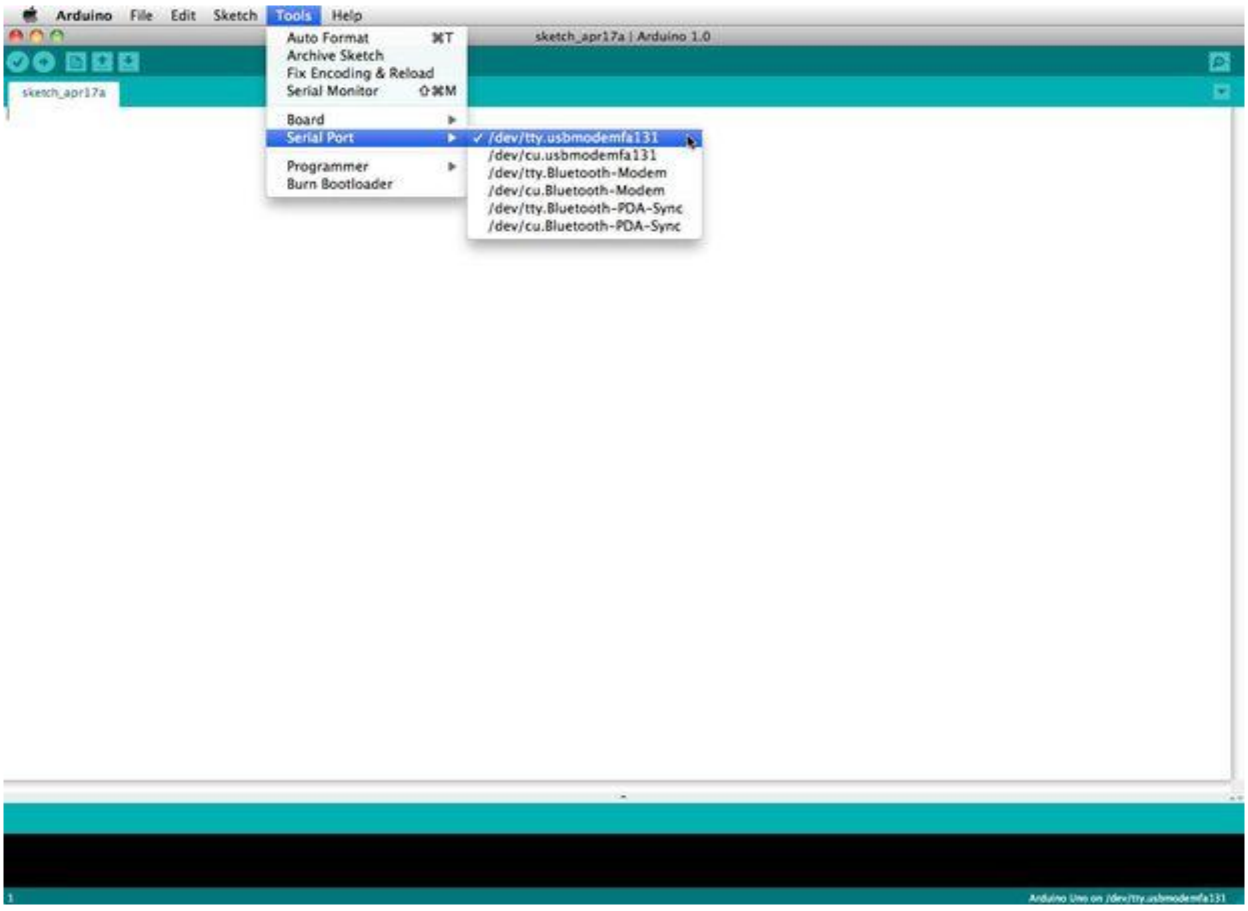

BareMinimum | Arduino 1.0
BareMinimum
1
Arduino Uno on /dev/tty.usbmodemfa131

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Before you can start doing anything with the Arduino, you need to download and install the Arduino IDE (integrated development environment). From this point on we will be referring to the Arduino IDE as the Arduino Programmer.

The Arduino Programmer is based on the Processing IDE and uses a variation of the C and C++ programming languages.

Settings



Before you can start doing anything in the Arduino programmer, you MUST set the board-type and serial port.

To set the board, go to the following:

Tools --> Boards

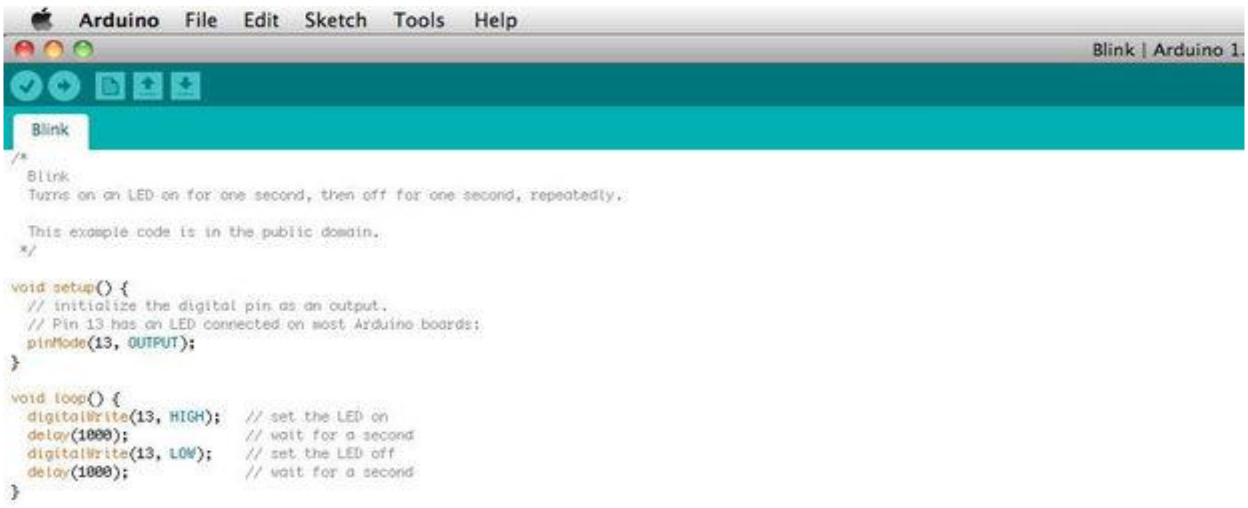
Select the version of board that you are using. Since I have an Arduino Uno plugged in, I obviously selected "Arduino Uno."

To set the serial port, go to the following:

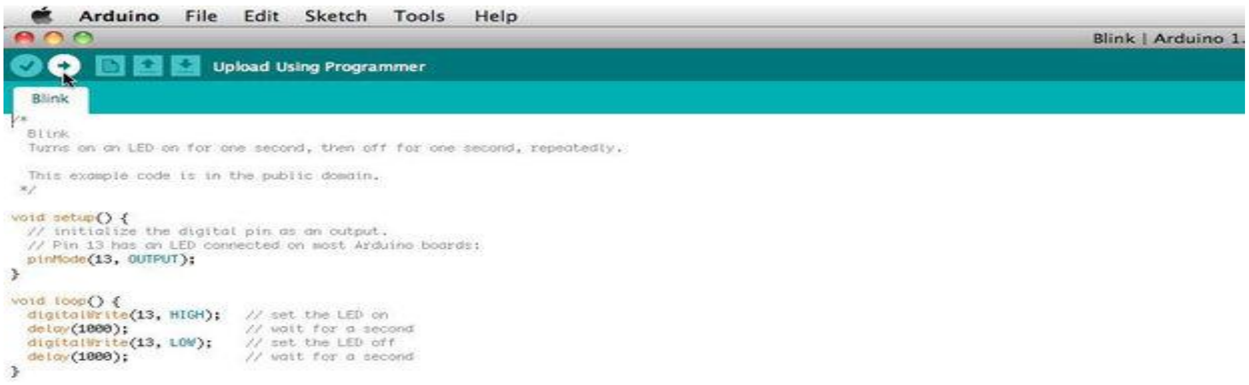
Tools --> Serial Port

Select the serial port that looks like:

/dev/tty.usbmodem [random number



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something.

To get the LED tied to digital pin 13 to blink on and off, let's load the blink example.

The blink example can be found here:

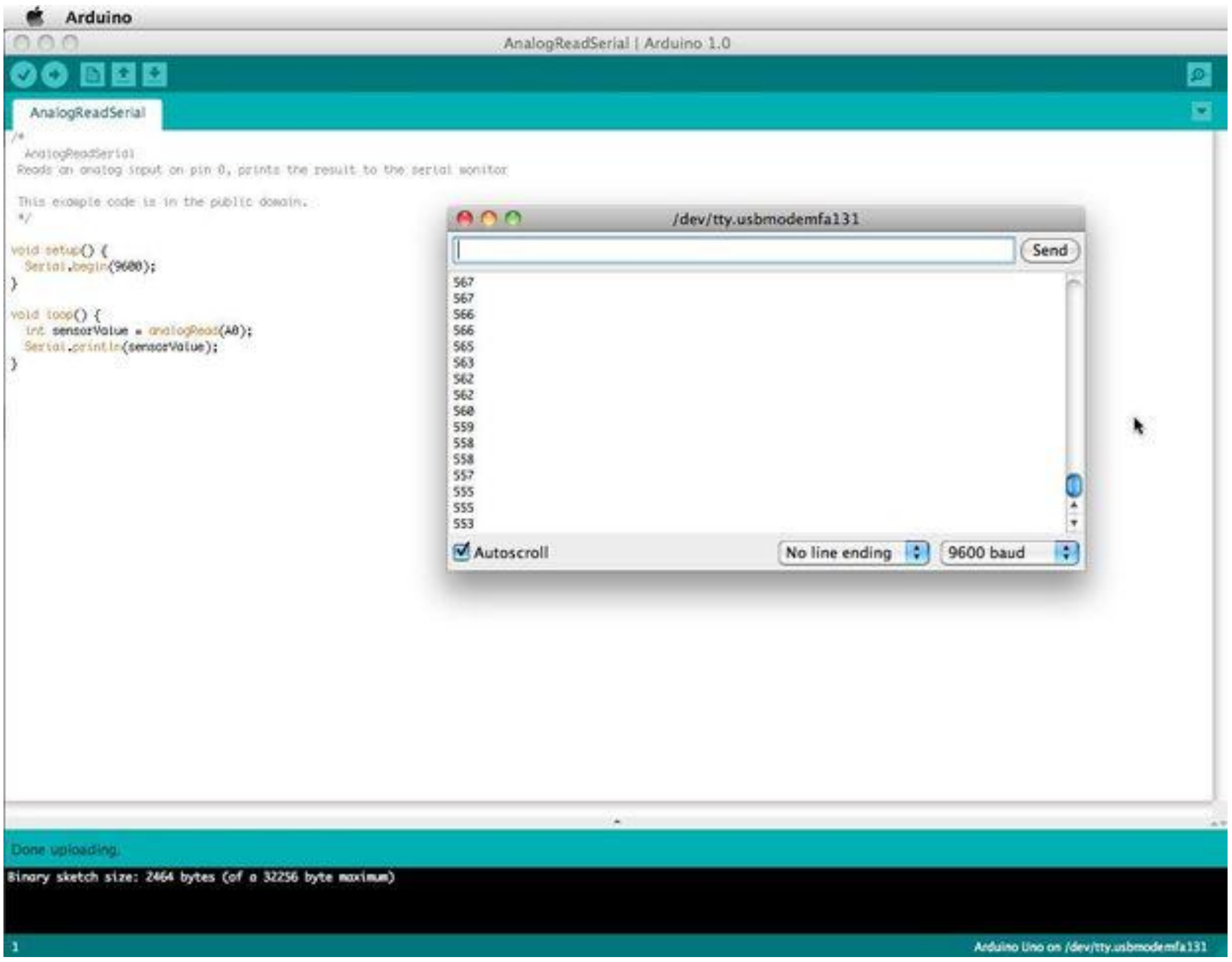
Files --> Examples --> Basics --> Blink

The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.

Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.

Notice that the surface mount status LED connected to pin 13 on the Arduino will start to blink. You can change the rate of the blinking by changing the length of the delay and pressing the upload button again.

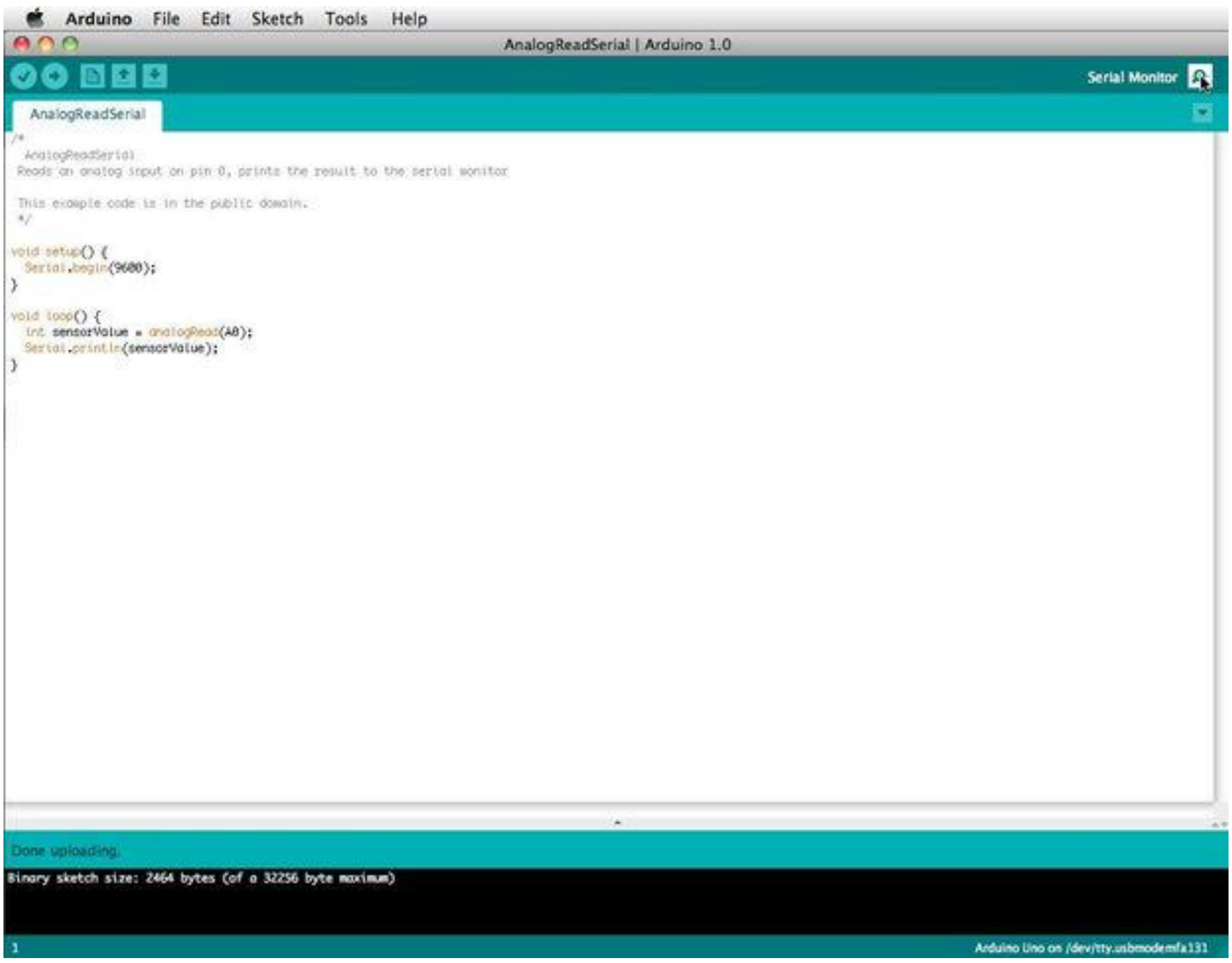
<u>Serial monitor</u>



```
/*
  AnalogReadSerial
 Reads an analog input on pin 0, prints the result to the serial monitor

 This example code is in the public domain.
 */

void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
}
```

```
567
567
566
566
565
563
562
562
560
559
558
558
557
555
555
553
```

```
Done uploading.
Binary sketch size: 2464 bytes (of a 32256 byte maximum)
```



```
/*
  AnalogReadSerial
 Reads an analog input on pin 0, prints the result to the serial monitor

 This example code is in the public domain.
 */

void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
}
```

```
Done uploading.
Binary sketch size: 2464 bytes (of a 32256 byte maximum)
```

```
1                                        Arduino Uno on /dev/tty.usbmodemfa131
```

The serial monitor allows your computer to connect serially with the Arduino. This is important because it takes data that your Arduino is receiving from sensors and other devices and displays it in real-time on your computer. Having this ability is invaluable to debug your code and understand what number values the chip is actually receiving.

For instance, connect center sweep (middle pin) of a potentiometer to A0, and the outer pins, respectively, to 5v and ground. Next upload the sketch shown below:

File --> Examples --> 1.Basics --> AnalogReadSerial

Click the button to engage the serial monitor which looks like a magnifying glass. You can now see the numbers being read by the analog pin in the serial monitor. When you turn the knob the numbers will increase and decrease.

The numbers will be between the range of 0 and 1023. The reason for this is that the analog pin is converting a voltage between 0 and 5V to a discreet number.

The Arduino has two different types of input pins, those being analog and digital.

To begin with, lets look at the digital input pins.

Digital input pins only have two possible states, which are on or off. These two on and off states are also referred to as:

- HIGH or LOW
- 1 or 0
- 5V or 0V.

This input is commonly used to sense the presence of voltage when a switch is opened or closed.

Digital inputs can also be used as the basis for countless digital communication protocols. By creating a 5V (HIGH) pulse or 0V (LOW) pulse, you can create a binary signal, the basis of all computing. This is useful for talking to digital sensors like a PING ultrasonic sensor, or communicating with other devices.

For a simple example of a digital input in use, connect a switch from digital pin 2 to 5V, a 10K resistor** from digital pin 2 to ground, and run the following code:

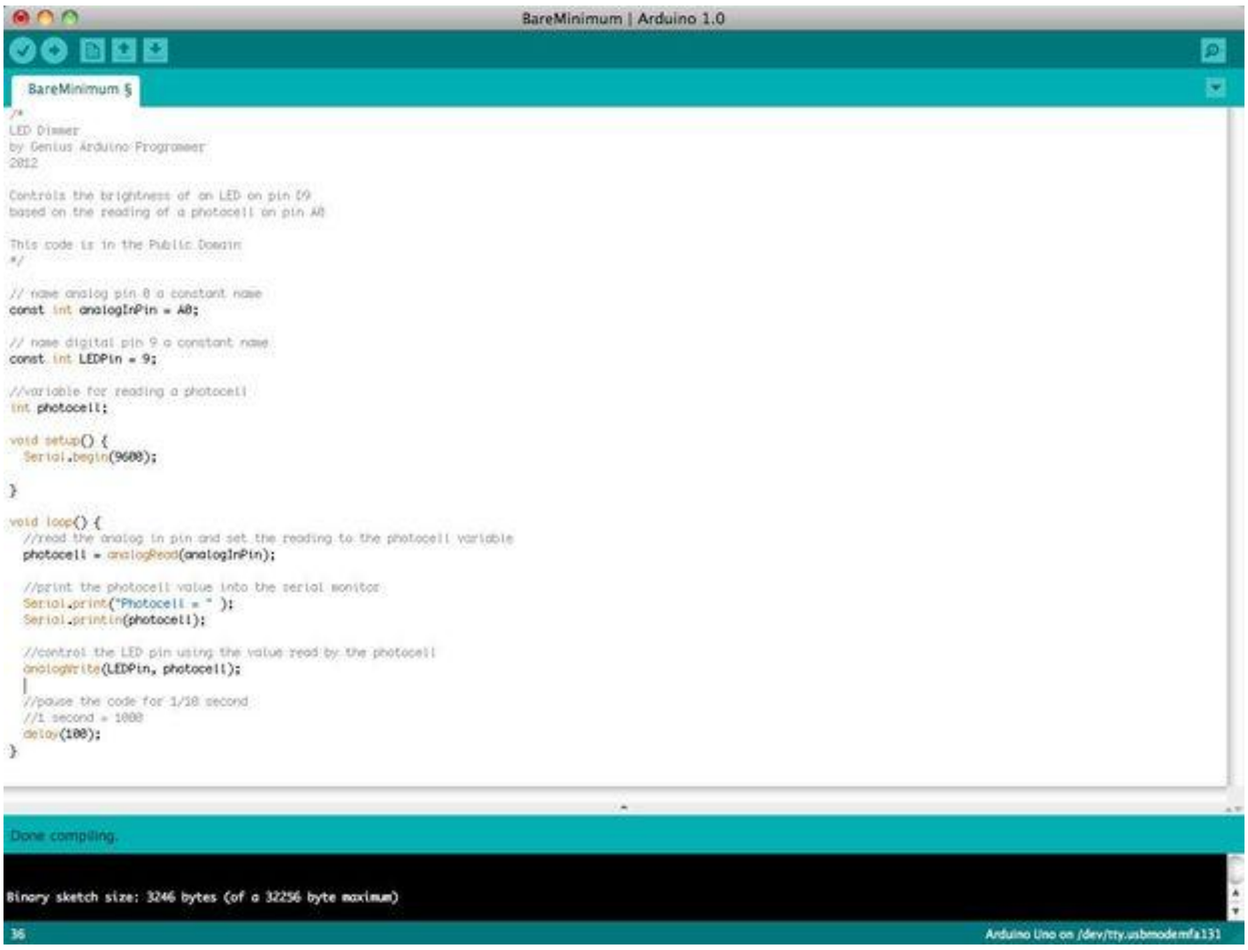
File --> Examples --> 2.Digital --> Button

<u>Analog in</u>

Aside from the digital input pins, the Arduino also boasts a number of analog input pins.

Analog input pins take an analog signal and perform a 10-bit analog-to-digital (ADC) conversion to turn it into a number between 0 and 1023 (4.9mV steps).

This type of input is good for reading resistive sensors. These are basically sensors which provide resistance to the circuit. They are also good for reading a varying voltage signal between 0 and 5V. This is useful when interfacing with various types of analog circuitry.



```
/*
LED Dimmer
by Genius Arduino Programmer
2012

Controls the brightness of an LED on pin D9
based on the reading of a photocell on pin A0

This code is in the Public Domain
*/

// name analog pin 0 a constant name
const int analogInPin = A0;

// name digital pin 9 a constant name
const int LEDPin = 9;

//variable for reading a photocell
int photocell;

void setup() {
  Serial.begin(9600);

}

void loop() {
  //read the analog in pin and set the reading to the photocell variable
  photocell = analogRead(analogInPin);

  //print the photocell value into the serial monitor
  Serial.print("Photocell = " );
  Serial.println(photocell);

  //control the LED pin using the value read by the photocell
  analogWrite(LEDPin, photocell);

  //pause the code for 1/10 second
  //1 second = 1000
  delay(100);
}
```

```
Done compiling.

Binary sketch size: 3246 bytes (of a 32256 byte maximum)
```

To write your own code, you will need to learn some basic programming language syntax. In other words, you have to learn how to properly form the code for the programmer to understand it. You can think of this kind of like understanding grammar and punctuation. You can write an entire book without proper grammar and punctuation, but no one will be abler to understand it, even if it is in English.

Some important things to keep in mind when writing your own code:

- An Arduino program is called a sketch.
- All code in an Arduino sketch is processed from top to bottom.
- Arduino sketches are typically broken into five parts.

1. The sketch usually starts with a header that explains what the sketch is doing, and who wrote it.
2. Next, it usually defines global variables. Often, this is where constant names are given to the different Arduino pins.

3. After the initial variables are set, the Arduino begins the setup routine. In the setup function, we set initial conditions of variables when necessary, and run any preliminary code that we only want to run once. This is where serial communication is initiated, which is required for running the serial monitor.
4. From the setup function, we go to the loop routine. This is the main routine of the sketch. This is not only where your main code goes, but it will be executed over and over, so long as the sketch continues to run.
5. Below the loop routine, there is often other functions listed. These functions are user-defined and only activated when called in the setup and loop routine. When these functions are called, the Arduino processes all of the code in the function from top to bottom and then goes back to the next line in the sketch where it left off when the function was called. Functions are good because they allow you to run standard routines - over and over - without having to write the same lines of code over and over. You can simply call upon a function multiple times, and this will free up memory on the chip because the function routine is only written once. It also makes code easier to read. To learn how to form your own functions, check outthis page.

- All of that said, the only two parts of the sketch which are mandatory are the Setup and Loop routines.
- Code MUST be written in the Arduino Language, which is roughly based on C.
- Almost all statements written in the Arduino language MUST end with a ;
- Conditionals (such as if statements and for loops) do not need a ;
- Conditionals have their own rules and can be found under "Control Structures" on the Arduino Language page
- Variables are storage compartments for numbers. You can pass values into and out of variables. Variables MUST be defined (stated in the code) before they can be used and need to have a data type associated with it. To learn some of the basic data types, review the Language Page.

First, we want to open the BareMinimum sketch, which can be found at:

File --> Examples --> 1.Basic --> BareMinimum

The BareMinimum Sketch should look like this:

```
<pre>void setup() {
}
void loop() {
 }
```

Next, lets put a header on the code, so other people know about what we are making, why, and under what terms:

```
<pre>

void setup() {
  }

void loop() {
  }
```

Once that is all squared away, let us define the pin names, and establish variables:

```
<pre>
const int analogInPin = A0;
const int LEDPin = 9;

int photocell;

void setup() {

}
void loop() {

}
```

Now that variables and pin names are set, let us write the actual code:

```
<pre>
const int analogInPin = A0;
const int LEDPin = 9;
int photocell;
void setup() {
}
void loop() {
  photocell = analogRead(analogInPin);
    analogWrite(LEDPin, photocell);
  delay(100);
}
```

If we want to see what numbers the analog pin is actually reading from the photocell, we will need to use the serial monitor. Let's activate the serial port and output those numbers:

```
<pre>
const int analogInPin = A0;
const int LEDPin = 9;
int photocell;
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.print("Photocell = " );
  Serial.println(photocell);
    analogWrite(LEDPin, photocell);
    delay(100);
}
```

RSA ALGORITHM

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric means that it works on two different keys i.e. Public Key and Private Key. As the name describes the Public Key is given to everyone and the Private key is kept private.

The idea of RSA is since it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So, if somebody can factorize the large number, the private key is compromised. Therefore, encryption strength lies in the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken shortly. But till now it seems to be an infeasible task.

The RSA algorithm's private and public key math.

It is possible to find three extremely large positive integers, e, d, and n. As a result, modular exponentiation for integers m(0 = m n) is as follows:

$$(m^e)^d = m(\bmod n) \qquad \text{............ (1)}$$

Even if e and n are known, it’s extremely hard to find d. (e, n) used as the public key. That means,

$$C = m^e \pmod n \qquad \text{............ (2)}$$

(d, n) used as the private key. That means

$$m = C^d \pmod n \qquad \text{. ....(3)}$$

Here, m stands for the original message, and C stands for cipher.

How does the RSA algorithm work?

RSA algorithm comprises four stages, and those four stages are:

- Key generation (1st stage): To generate a private key (to keep) and a public key (to share).
- Key distribution (2nd stage): Flood the network with the public key.
- Encryption (3rd stage): The sender encrypts the message using the receiver’s public key.
- Decryption (4th stage): The message is decrypted by the receiver using its private key.

How to generate RSA key?

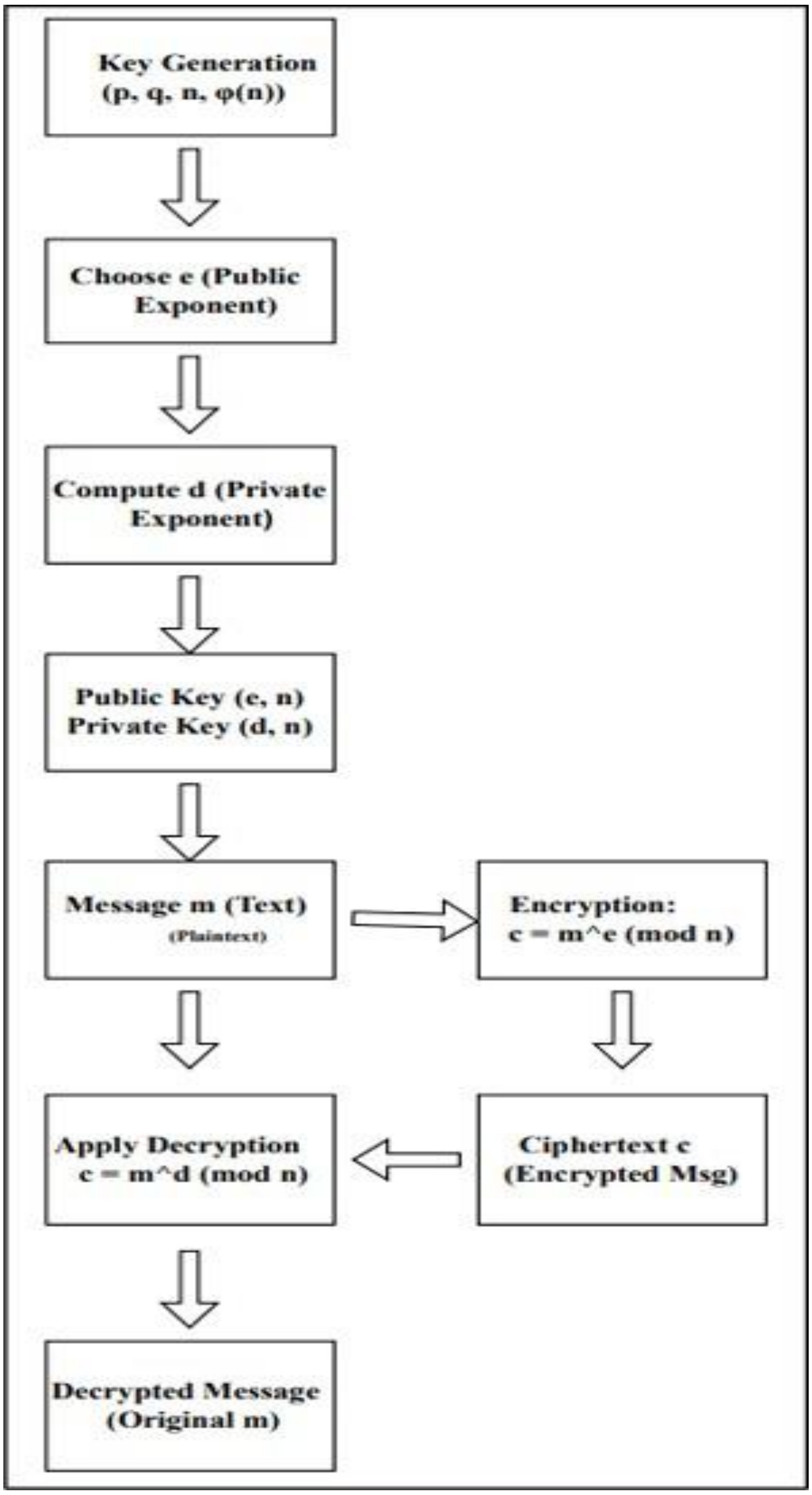

Key Generation (p, q, n, φ(n))
Choose e (Public Exponent)
Compute d (Private Exponent)
Public Key (e, n) Private Key (d, n)
Message m (Text) (Plaintext)
Encryption: c = m^e (mod n)
Ciphertext c (Encrypted Msg)
Apply Decryption c = m^d (mod n)
Decrypted Message (Original m)

Fig 5.1.1: RSA Algorithm flowchart

Here are the steps to generate RSA keys

- STEP 1: Choose two distinct prime integers, p, and q. Note: p and q should be chosen randomly.
- STEP 2: Calculate n = p*q. Here, ‘n’ is the length of the key.
- STEP 3: Calculate f(n) = (p-1) (q-1) {Euler’s Totient function}
- STEP 4: Choose an integer ‘e’ such that 1 < e < (n) and GCD (e, f(n)) = 1 where ‘e’ and ‘f(n)’ are co-prime (means a pair of number that don’t have any common factor other than 1). Also, GCD is the greatest common factor.

Determine:

$$d = e^{-1} (\text{mod } f(n))$$

Here, ‘d’ is the modular multiplicative inverse of e (mod f(n)). [Note: (d*e) (mod f(n)) = 1]

STEP 1: Create a Public Key

Following the steps:

- Selecting two prime numbers, p, and q. Let’s say p = 3 and q = 11.
- Calculate n = pq. So, n = 33. As, 3*11 = 33 = n.
- Calculate f(n) = (p-1) (q-1). Here, f(n) = 20. As, f(n) = (p-1) (q-1) = (2)*(10) = 20
- Choosing e=7. Such that 1 < e < f(n) and GCD (e, f(n)) = 1.

Hence, Public Key is (e, n) = (7, 33)

STEP 2: Create a Private Key Using, (d*e) (mod f(n))=1.

One Solution can be d=3. Such that, (3*7) % 20 = 1. Hence,

Private Key is (d, n) = (3, 33)

STEP 3: Encryption and Decryption of letter H

Let’s encrypt the ‘H’. where H = 2.

Encryption of H:

$$C = m^e (\text{mod } n)$$

C=(2^7)% 33 = 29 => C = 29

The encryption of the letter H is 29.

Decryption of H:

$$m = C^{d} (\bmod n)$$

m=(29^3)% 33 = 2 => H = 2

The decryption of the letter H is 2.

We encrypted and decrypted the letter H.

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal.h>
#include <PS2Keyboard.h>
LiquidCrystal lcd(10, 11, A0, A1, A2, A3);
SoftwareSerial esp(8, 9);
PS2Keyboard keyboard;
#define blank "                    "
#define msg "msg:                "
#define you "you:                "
#define lcd_col 20
#define lcd_row 4
int menu = 1;
int clear_lcd = 1;
int name_lcd_col = 0;
int name_lcd_row = 2;
int temp_name_col = name_lcd_col;
int temp_name_row = name_lcd_row;
int tx_lcd_col = 4;
int tx_lcd_row = 2;
int temp_tx_col = tx_lcd_col;
int temp_tx_row = tx_lcd_row;
int rx_lcd_col = 4;
int rx_lcd_row = 0;
int temp_rx_col = rx_lcd_col;
int temp_rx_row = rx_lcd_row;

const int DataPin = 3;
const int IRQpin = 2;
const int espRST = 7;
int data_available = 0;

char c, mode;
String text_data = "", incoming_data = "";
String keypad_data = "";
char buff[50];


void setup() {
  Serial.begin(9600);
  pinMode(espRST, OUTPUT);
  keyboard.begin(DataPin, IRQpin);
  lcd.begin(lcd_col, lcd_row);
  lcd.clear();
  lcd.print(F("  Data Transmission"));  // Print a message to the LCD.
  lcd.setCursor(0, 1);                  // set the cursor to column 0, line 1
  lcd.print(F("    through WIFI"));
  lcd.setCursor(0, 2);
  lcd.print("       using");
  lcd.setCursor(0, 3);
  lcd.print(F("   RSA encryption"));
  digitalWrite(espRST, HIGH);
  delay(500);
  digitalWrite(espRST, LOW);
  delay(100);
  digitalWrite(espRST, HIGH);
  delay(3000);
```

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(F("Chose prime numbers"));
lcd.setCursor(0, 1);
lcd.print(F("->(7,19)"));
lcd.setCursor(0, 2);
lcd.print(F("  (11,23)"));

do {
  if (keyboard.available()) {
    c = char(keyboard.read());
    if (c == PS2_UPARROW) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(F("Chose prime numbers"));
      lcd.setCursor(0, 1);
      lcd.print(F("->(7,19)"));
      lcd.setCursor(0, 2);
      lcd.print(F("  (11,23)"));
      menu = 1;
    } else if (c == PS2_DOWNARROW) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(F("Chose prime numbers"));
      lcd.setCursor(0, 1);
      lcd.print(F("  (7,19)"));
      lcd.setCursor(0, 2);
      lcd.print(F("->(11,23)"));
      menu = 2;
    }
  }
} while (c != PS2_ENTER);
delay(100);
```

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(F("Keys generated"));
if (menu == 1) {
  mode = 'A';
  lcd.setCursor(0, 1);
  lcd.print("Public key : 29");
  lcd.setCursor(0, 2);
  lcd.print("Private : 41");
} else if (menu == 2) {
  mode = 'Z';
  lcd.setCursor(0, 1);
  lcd.print("Public key : 7");
  lcd.setCursor(0, 2);
  lcd.print("Private : 47");
}
esp.begin(9600);
delay(1000);
esp.flush();
delay(1000);
esp.listen();
delay(1000);
String Serial_response = "";
do {
  esp.println(mode);
  if (esp.available() > 0) {
    Serial_response = esp.readStringUntil('\n');
  }
} while (Serial_response.indexOf("OK") >= 0);
lcd.clear();
lcd.print("Started");
delay(1000);
lcd.clear();
lcd.setCursor(0, 0);
```

```
  lcd.print(msg);
  lcd.setCursor(0, 1);
  lcd.print(blank);
  lcd.setCursor(0, 2);
  lcd.print(you);
  lcd.setCursor(0, 3);
  lcd.print(blank);
}

void loop() {
  // put your main code here, to run repeatedly :
  if (keyboard.available()) {
    c = char(keyboard.read());
    if (c != PS2_ENTER && c != PS2_BACKSPACE) {
      if (keypad_data.length() == 0) {
        lcd.setCursor(0, 2);
        lcd.print(you);
        lcd.setCursor(0, 3);
        lcd.print(blank);
      }
      keypad_data += c;
      lcd.setCursor(temp_tx_col, temp_tx_row);
      lcd.print(c);
      temp_tx_col++;
    }
    if (temp_tx_col >= lcd_col) {
      temp_tx_row++;
      temp_tx_col = tx_lcd_col;
      if (temp_tx_row >= lcd_row) {
        lcd.setCursor(temp_tx_col, temp_tx_row);
        lcd.print(c);
        temp_tx_row = tx_lcd_row;
        temp_tx_col = tx_lcd_col;
      }
```

```
}
if (c == PS2_ENTER) {
  keypad_data.trim();
  if (keypad_data.length() > 1) {
    lcd.setCursor(0, 2);
    lcd.print(F("you: sending msg... "));
    lcd.setCursor(0, 3);
    lcd.print(blank);
    if (send(keypad_data, true)) {
      lcd.setCursor(0, 2);
      lcd.print(F("you: msg sent       "));
      lcd.setCursor(0, 3);
      lcd.print(blank);
      delay(500);
      lcd.setCursor(0, 2);
      lcd.print(you);
      lcd.setCursor(0, 3);
      lcd.print(blank);
    } else {
      lcd.setCursor(0, 2);
      lcd.print(F("you:fail to send msg"));
      lcd.setCursor(0, 3);
      lcd.print(blank);
    }
    clear_lcd = 1;
    keypad_data = "";
    temp_tx_row = tx_lcd_row;
    temp_tx_col = tx_lcd_col;
  }
}
if (c == PS2_BACKSPACE) {
  if (temp_tx_col > tx_lcd_col) {
    temp_tx_col--;
    lcd.setCursor(temp_tx_col, temp_tx_row);
```

```
      lcd.print(" ");
      lcd.setCursor(temp_tx_col, temp_tx_row);
      keypad_data.remove(keypad_data.length() - 1);
    } else if (temp_tx_row > tx_lcd_row) {
      temp_tx_row = tx_lcd_row;
      temp_tx_col = lcd_col - 1;
      lcd.setCursor(temp_tx_col, temp_tx_row);
      lcd.print(" ");
      lcd.setCursor(temp_tx_col, temp_tx_row);
      keypad_data.remove(keypad_data.length() - 1);
    }
  }
}

if (esp.available() > 0) {
  incoming_data = esp.readStringUntil('\n');
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(msg);
  lcd.setCursor(0, 1);
  lcd.print(blank);
  temp_rx_row = rx_lcd_row;
  temp_rx_col = rx_lcd_col;
  for (int j = 0; j < incoming_data.length() - 2; j++) {
    lcd.setCursor(temp_rx_col, temp_rx_row);
    lcd.print(incoming_data[j]);
    temp_rx_col++;
    if (temp_rx_col >= lcd_col) {
      temp_rx_row++;
      temp_rx_col = rx_lcd_col;
      if (temp_rx_row >= rx_lcd_col) {
        lcd.setCursor(0, 0);
        lcd.print(msg);
        lcd.setCursor(0, 1);
```

```
          lcd.print(blank);
          temp_rx_row = rx_lcd_row;
          temp_rx_col = rx_lcd_col;
        }
      }
    }
  }
}
bool send(String message, bool ack) {
  Serial.println(message);
  esp.println(message);
  return true;
}
#include "painlessMesh.h"
#define MESH_PREFIX "whateverYouLike"
#define MESH_PASSWORD "somethingSneaky"
#define MESH_PORT 5555
Scheduler userScheduler;  // to control your personal task
painlessMesh mesh;
// RSA parameters
int p = 7, q = 19;  // Prime numbers (p, q)
int n, t;
int public_key = 29;   // Public key exponent (E)
int private_key = 41;  // Private key exponent (D)
// User stub
void sendMessage();  // Prototype so PlatformIO doesn't complain

Task taskSendMessage(TASK_SECOND * 1, TASK_FOREVER, &sendMessage);
void newConnectionCallback(uint32_t nodeId) {
  //Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}

void changedConnectionCallback() {
  //Serial.printf("Changed connections %s\n",mesh.subConnectionJson().c_str());
```

```
}


void nodeTimeAdjustedCallback(int32_t offset) {
  //Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(),offset);
}
void setup() {
  Serial.begin(9600);
  while (1) {
    if (Serial.available() > 0) {
      char c = (char)Serial.read();
      if (c == 'A') {
        p = 7, q = 19;
        public_key = 29, private_key = 41;
        break;
      } else if (c == 'Z') {
        p = 11, q = 17;
        public_key = 7, private_key = 23;
        break;
      }
    }
  }
  n = p * q;              // Product (N)
  t = (p - 1) * (q - 1);  // Euler's totient (T)


  Serial.println("OK");
  //mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC |
COMMUNICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
  mesh.setDebugMsgTypes(ERROR | STARTUP);  // set before init() so that you can see startup
messages
  mesh.init(MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT);
  mesh.onReceive(&receivedCallback);
  mesh.onNewConnection(&newConnectionCallback);
  mesh.onChangedConnections(&changedConnectionCallback);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
  userScheduler.addTask(taskSendMessage);
  taskSendMessage.enable();
}
void loop() {
  userScheduler.execute();  // it will run mesh scheduler as well
  mesh.update();
}
void sendMessage() {
  String message;
  if (Serial.available()) {
    message = Serial.readStringUntil('\n');
    // mesh.sendBroadcast(message);
    String encrypted_data;
    for (int i = 0; i < message.length(); i++) {
      int m = int(message[i]);                          // Get ASCII value of the character
      int encrypted_msg = mod_exp(m, public_key, n);  // RSA encryption (m^e % n)
      encrypted_data += char(encrypted_msg);
    }
    mesh.sendBroadcast(encrypted_data);
    taskSendMessage.setInterval(random(TASK_SECOND * 1, TASK_SECOND * 5));
  }
}

// Needed for painless library
void receivedCallback(uint32_t from, String &incoming_data) {
  String decrypted_data;
  for (int i = 0; incoming_data[i] != '\0'; i++) {
    int decrypted_msg = mod_exp(char(incoming_data[i]), private_key, n);  // RSA decryption (c^d % n)
    decrypted_data += (char)decrypted_msg;
  }
  Serial.println(decrypted_data);
}
long mod_exp(long base, long exp, long mod) {
  long result = 1;
  base = base % mod;
  while (exp > 0) {
   if (exp % 2 == 1) {  // If exp is odd, multiply base with result
     result = (result * base) % mod;
   }
   exp = exp >> 1;               // Right shift exp by 1 (divide by 2)
   base = (base * base) % mod;  // Square the base
  }
  return result;
}
s
```

5.2 IMPLEMENTATION IMAGES

The results of this project validate the implementation of a secure wireless communication system using Wi-Fi and RSA encryption. The system was tested under controlled conditions to evaluate its efficiency, functionality, and ability to ensure data confidentiality and integrity.

STEP 1: Connection of Components in PCB Board

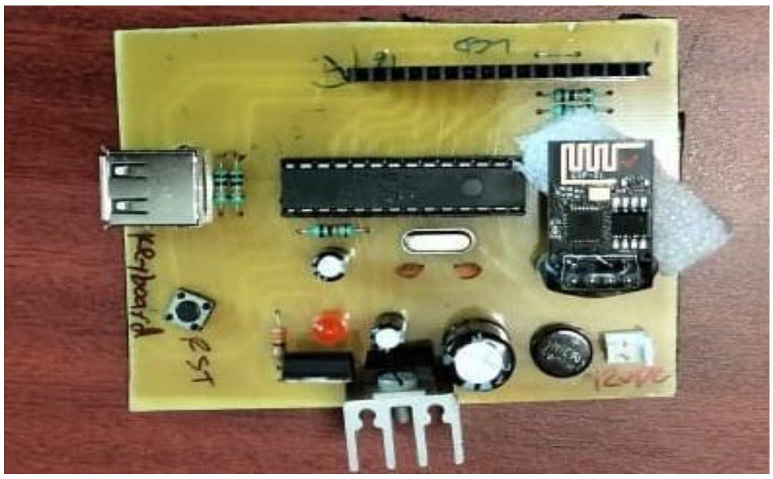


Fig 5.2.1: Connection of the components in PCB board

The figure 5.2.1 shows the placement of all hardware components, such as the microcontroller, voltage regulator, and Wi-Fi module, on the PCB board. The PCB ensures organized circuitry, reducing noise and interference in signal transmission.

STEP 2: Connection of PCB Design and LCD

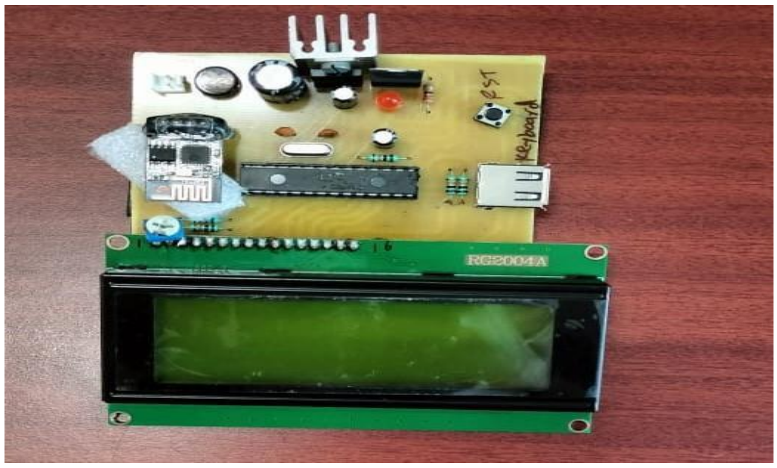


Fig 5.2.2: Connection of PCB Design and LCD

The above figure 5.2.2 illustrates the integration of the PCB with the LCD module. The LCD displays critical messages and real-time communication, confirming encryption and decryption processes**.**

STEP 3: Connection of PCB Board and Keyboard

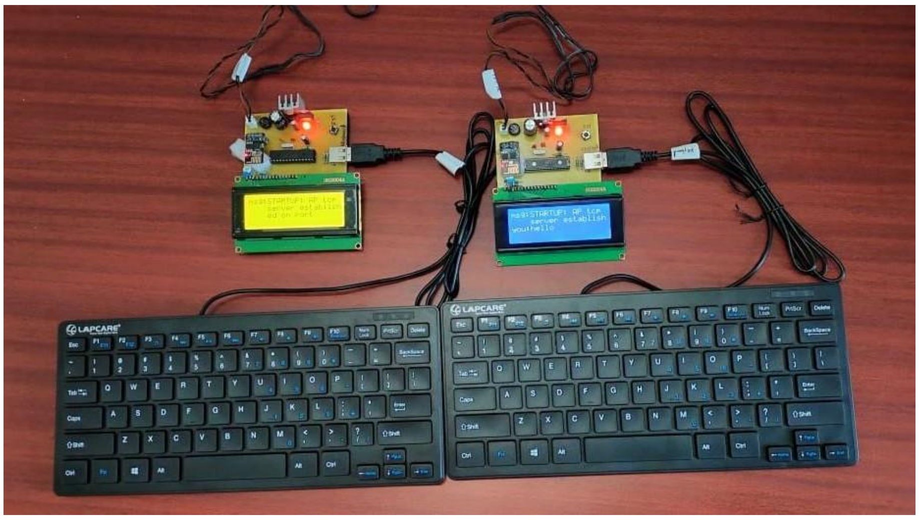


Fig 5.2.3: Connection of PCB Board and Keyboard

The figure 5.2.3 demonstrates the interface between the PCB board and the keyboard, allowing the user to input messages for encryption and transmission.

STEP 4: Connection of Transmitter

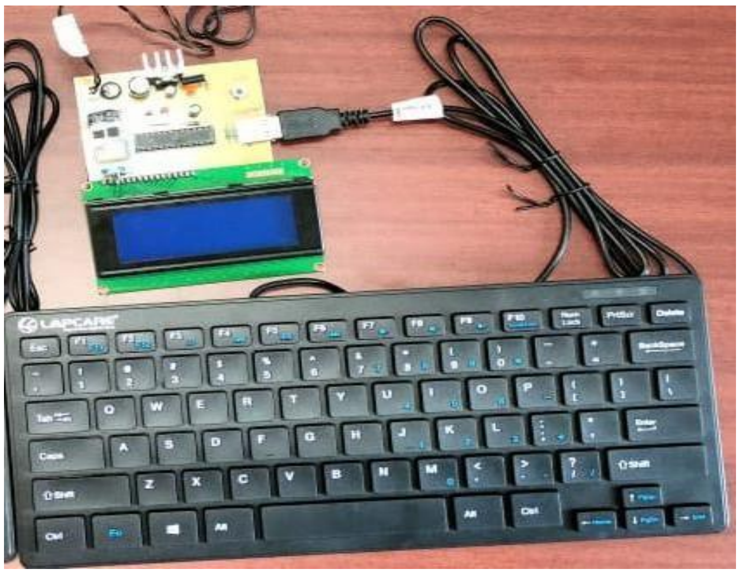


Fig 5.2.4: Connection of Transmitter

The above figure 5.2.4 highlights the transmitter setup, including the connection of the Wi-Fi module to the microcontroller, ensuring proper message encryption and transmission over the wireless network.

STEP 5: Connection of Receiver

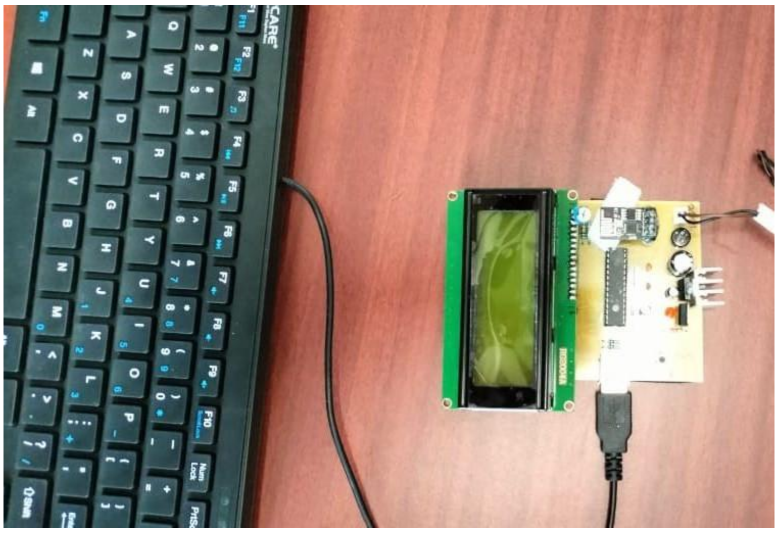


Fig 5.2.5: Connection of Receiver

The figure 5.2.5 shows the receiver setup, detailing the connection of the receiver's Wi-Fi module to the microcontroller for receiving encrypted messages and subsequent decryption.

STEP 6: Overall connection of the Transceiver

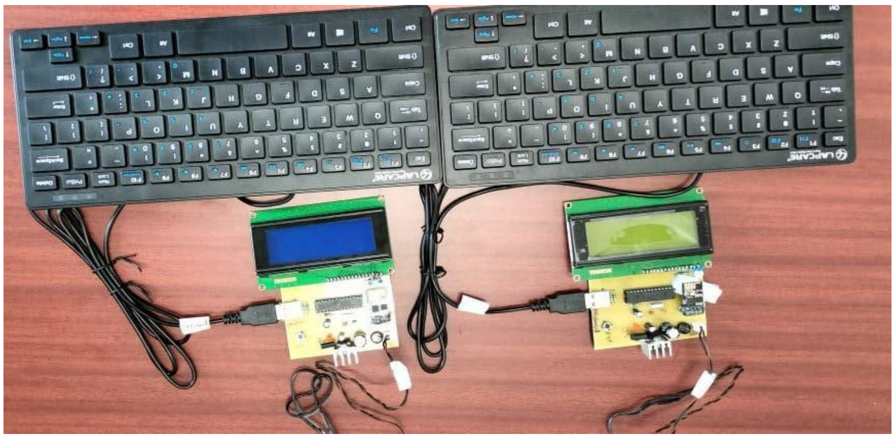


Fig 5.2.6: Overall connection of the Transceiver

The above figure 5.2.6 consolidates all components in a single schematic, showing the transmitter and receiver working in tandem to ensure secure wireless communication.

STEP 7: Generation of two prime numbers

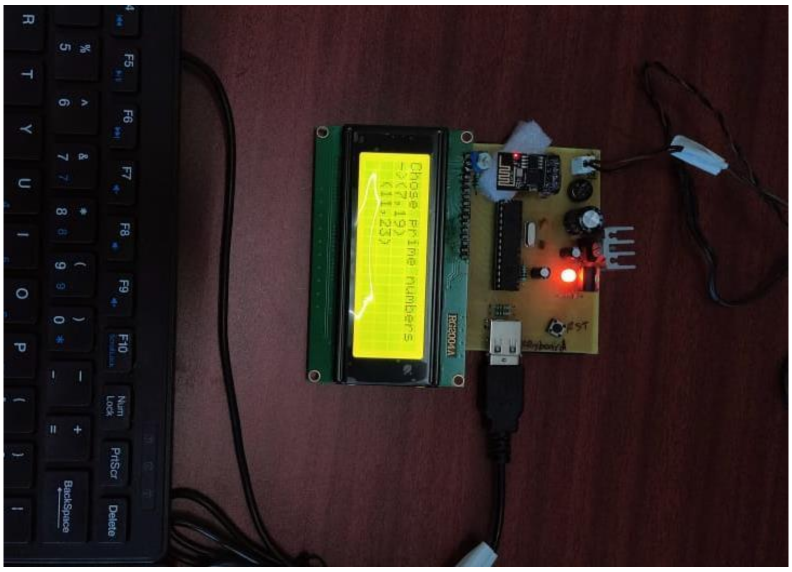


Fig 5.2.7: Generation of two prime numbers

The figure 5 .2.7 represents the RSA algorithm's initialization process, where two prime numbers are generated for key creation, forming the backbone of encryption and decryption.

STEP 8: Establishment of two nodes

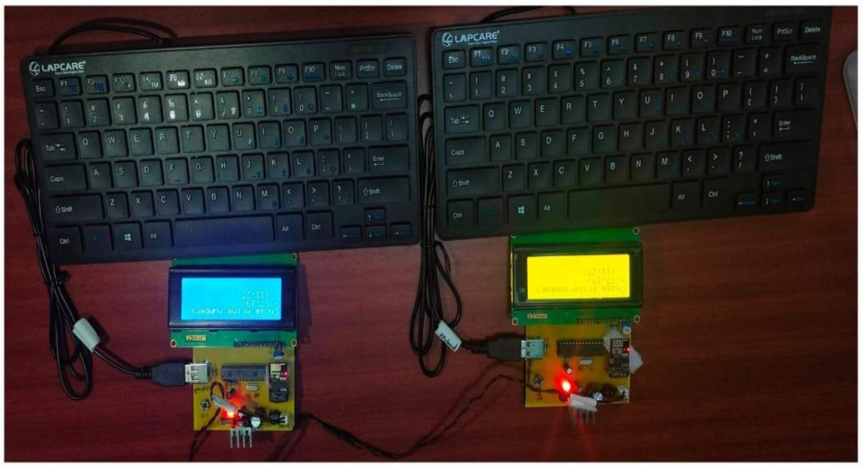


Fig 5.2.8: Establishment of two nodes

The above figure 5.2.8 depicts the creation of two secure communication nodes, ensuring data is transmitted and received only between authorized devices.

STEP 9: “Server establish” confirm operational success

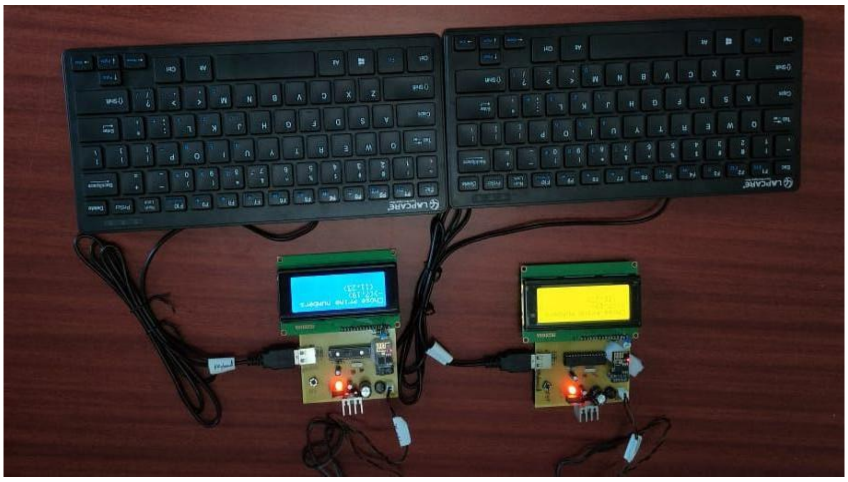


Fig 5.2.9: “Server establish” confirm operational success

The figure 5.2.9confirms the successful establishment of the server, marking the system's readiness for secure communication using RSA encryption.

CHAPTER 6

HARDWARE TESTING

6.1 CONDUCTED

CONTINUITY TEST:

In electronics, a continuity test is the checking of an electric circuit to see if current flows (that it is in fact a complete circuit). A continuity test is performed by placing a small voltage (wired in series with an LED or noise-producing component such as a piezoelectric speaker) across the chosen path. If electron flow is inhibited by broken conductors, damaged components, or excessive resistance, the circuit is "open".

Devices that can be used to perform continuity tests include multi meters which measure current and specialized continuity testers which are cheaper, more basic devices, generally with a simple light bulb that lights up when current flows.

An important application is the continuity test of a bundle of wires so as to find the two ends belonging to a particular one of these wires; there will be a negligible resistance between the "right" ends, and only between the "right" ends.

This test is the performed just after the hardware soldering and configuration has been completed. This test aims at finding any electrical open paths in the circuit after the soldering. Many a times, the electrical continuity in the circuit is lost due to improper soldering, wrong and rough handling of the PCB, improper usage of the soldering iron, component failures and presence of bugs in the circuit diagram. We use a multi meter to perform this test. We keep the multi meter in buzzer mode and connect the ground terminal of the multi meter to the ground. We connect both the terminals across the path that needs to be checked. If there is continuation then you will hear the beep sound.

POWER ON TEST:

This test is performed to check whether the voltage at different terminals is according to the requirement or not. We take a multi meter and put it in voltage mode. Remember that this test is performed without microcontroller. Firstly, we check the output of the transformer, whether we get the required 12 v AC voltage.

Then we apply this voltage to the power supply circuit. Note that we do this test without microcontroller because if there is any excessive voltage, this may lead to damaging the controller. We check for the input to the voltage regulator i.e., are we getting an input of 12v and an output of 5v. This 5v output is given to the microcontrollers' 40th pin. Hence we check for the voltage level at 40th pin. Similarly, we check for the other terminals for the required voltage. In this way we can assure that the voltage at all the terminals is as per the requirement

6.2 Test cases.

Test ID	Name	Description	Expected Output	Actual Output	Result
T1	RSA Key Generation	Generate an RSA key pair (p, q → public/private keys)	Public and private keys produced; modulus $n = p \cdot q$; keys satisfy $e \cdot d \equiv 1 \bmod \varphi(n)$	Public key (29), Private key (41), $\varphi(n)=96$	Pass
T2	Encryption Functionality	Encrypt sample plaintext "HELLO" with public key	Ciphertext is non-readable gibberish; length ≥ plaintext length	"\x12\xA5…" (binary data), length=5	Pass
T3	Decryption Correctness	Decrypt the ciphertext from T2 with private key	Decrypted text equals original "HELLO"	"HELLO"	Pass
T4	End-to-End Secure Transmission	Send a 16-byte sensor reading "TEMP=25.4C" through WiFi using RSA encryption/decryption cycle	Receiver successfully decrypts and logs "TEMP=25.4C"	"TEMP=25.4C"	Pass
T5	Wrong-Key Decryption	Attempt to decrypt a valid ciphertext with an incorrect private key	Decryption fails or returns garbled output	Garbled text ("\xFF\x03…")	Pass
T6	Performance Benchmark	Measure encryption + decryption time on ESP8266 for 32-byte message	Total RSA processing < 200 ms	180 ms	Pass

CHAPTER 7

CONCLUSION

The project successfully demonstrates a secure wireless communication system using RSA encryption over Wi-Fi. Future work could involve integrating real-time message monitoring, scaling to multiple devices, and enhancing encryption algorithms for greater security. The result of the secure wireless communication system using Wi-Fi and RSA encryption. A TCP-based communication channel was established, enabling encrypted message transmission and decryption using RSA public and private keys. Real-time feedback, such as "Server Establish," confirmed operational success, with messages displayed on an LCD module. Hardware components, including the ATmega328 microcontroller and ESP8266 Wi-Fi module, operated seamlessly. The system ensures data confidentiality, integrity, and minimal latency, making it suitable for IoT, industrial automation, and secure messaging. This validates RSA's effectiveness in securing Wi-Fi networks, with potential for scalability and advanced encryption integration.WI-FI Based Secure Wireless Communication Using RSA

CHAPTER 8

FUTURE ENHANCEMENTS

The RSA algorithm is one of the most widely used public-key cryptosystems, ensuring secure data transmission over the internet. In this project, we successfully implemented the RSA algorithm for encryption and decryption tasks, demonstrating its ability to

• **Generate Keys:** The algorithm generates a public-private key pair based on two

randomly chosen prime numbers, ensuring security.

• **Encrypt Messages:** It successfully encrypts plaintext messages into ciphertext using

the public key, making them unreadable without the private key.

• **Decrypt Messages:** Using the private key, the ciphertext can be decrypted back to its original plaintext, preserving the integrity of the message.

• **Performance**: The system showed efficient encryption and decryption times, which can be adjusted by changing the key size. The RSA algorithm is fundamental in moder cryptography, providing a robust mechanism for secure communication.

However, with increasing computational power, RSA's security may be challenged for large-scale systems in the future, especially when used with smaller key sizes.WI-FI Based Secure Wireless C

Appendix A

A. Bibliography

<u>TEXT BOOKS REFERED</u>

1. ATMEGA 328 Data Sheets.

<u>WEBSITES</u>

- www.atmel.com
- www.beyondlogic.org
- www.wikipedia.org
- www.howstuffworks.com
- www.alldatasheets.com

B. User Manual

How to use application

1. Power On Devices

- Connect both ESP8266-based devices to power (via USB or 5V adapter)
- Wait a few seconds for the devices to initialize and establish a WiFi mesh connection

---

2. Key Generation

- Open the Serial Monitor of the transmitter device
- Enter two small prime numbers (e.g., 11 and 17) when prompted
- The system will:

  Calculate n = p × q

  Generate public key (n, e) and private key (n, d)

  Display them on the Serial Monitor.

---

3. Send a Message

- Use the PS/2 keyboard connected to the transmitter to type a message.
- After pressing "Enter":

  The message is **encrypted** using the public key.

  It is transmitted over WiFi to the receiver device.

---

4. Receive the Message

- The receiver ESP8266 gets the encrypted message.
- It uses the private key to decrypt it.
- The OLED or TFT display shows the original message.

- An LED or buzzer briefly activates to notify the user.

---

5. Send More Messages or Reset

- To send another message, just type a new one and press "Enter".
- To reset and enter new primes, type "reset" in the Serial Monitor.

---

6. Security Note

- All messages are securely encrypted with RSA before transmission.
- Only the intended receiver can decrypt the message using the private key.