# Flow Diagram and Detailed Explanation of the Secure Sender Program

1. Key Generation and Serialization

* **Step 1: Generate RSA Key Pair**
  + The generate\_rsa\_keys() function generates a 2048-bit RSA key pair for secure communication.
  + **Mathematical Concept:** RSA key pair involves selecting two large prime numbers pp and qq, and computing: n=p×qn = p \times q ϕ(n)=(p−1)×(q−1)\phi(n) = (p - 1) \times (q - 1) The public key exponent ee is chosen (commonly 65537), and the private key exponent dd satisfies: e×d≡1mod  ϕ(n)e \times d \equiv 1 \mod \phi(n)
* **Step 2: Serialize Keys**
  + The serialize\_rsa\_keys() function serializes the public and private keys into PEM format for transmission.

1. Secure Connection Establishment

* A TCP connection is established using the socket module, with the sender connecting to the receiver at localhost:65432.
* The sender receives the receiver's public RSA key.
* The sender sends its public RSA key to the receiver.

## Chaotic Key Generation for AES

* The chaotic.key\_generation() function generates an AES key using chaotic functions.
  + **Mathematical Insight:** Chaotic systems are sensitive to initial conditions, providing strong randomness.
  + The binary output is converted to bytes for AES.

## AES Encryption

* **Mathematical Concept:** AES operates on 128-bit blocks.
  + **Encryption Formula:** C=Ek(P)C = E\_k(P) Where:
    - CC is the ciphertext.
    - EkE\_k is the AES encryption function using key kk.
    - PP is the plaintext.
* **PKCS7 Padding:** Ensures that the data length aligns with AES block size.
* The encrypt\_message\_with\_aes() function applies AES-CBC encryption.

## RSA Encryption of AES Key

* The AES key is encrypted using the receiver's public RSA key: Ck=ERSA(k)C\_k = E\_{RSA}(k)
* **Mathematical Concept:** Ck=kemod  nC\_k = k^e \mod n

## Digital Signature Generation

* The sign\_message() function signs the encrypted message using the sender's RSA private key.
* **Mathematical Concept:** RSA signature is computed as: S=H(M)dmod  nS = H(M)^d \mod n Where H(M)H(M) is the hash of the message.

## Data Transmission

* The encrypted AES key, IV, encrypted message, and signature are sent to the receiver.
* Lengths of the encrypted message and signature are sent first using struct.pack().

## Summary

This program demonstrates secure data transmission by integrating RSA for key exchange, AES for data encryption, and digital signatures for data integrity. The use of chaotic key generation enhances the randomness of cryptographic keys, ensuring secure communication.

# Flow Diagram and Detailed Explanation of the Secure Receiver Program

## Key Generation and Serialization

* **Step 1: Generate RSA Key Pair**
  + The generate\_rsa\_keys() function generates a 2048-bit RSA key pair for secure communication.
  + **Mathematical Concept:** RSA key pair involves selecting two large prime numbers pp and qq, and computing: n=p×qn = p \times q ϕ(n)=(p−1)×(q−1)\phi(n) = (p - 1) \times (q - 1) The public key exponent ee is chosen (commonly 65537), and the private key exponent dd satisfies: e×d≡1mod  ϕ(n)e \times d \equiv 1 \mod \phi(n)
* **Step 2: Serialize Keys**
  + The serialize\_rsa\_keys() function serializes the public and private keys into PEM format for transmission.

## Secure Connection Establishment

* A TCP server is created using the socket module.
* The receiver binds to localhost:65432 and listens for a connection.
* The receiver sends its public RSA key to the sender and receives the sender's public RSA key.

## Data Reception

* The recv\_all() function ensures that the exact number of bytes is received for each data component.
* **Data Flow:**
  + Encrypted AES key (256 bytes)
  + Initialization vector (IV) (16 bytes)
  + Encrypted message length (4 bytes)
  + Signature length (4 bytes)
  + Encrypted message
  + Digital signature

## RSA Decryption of AES Key

* The AES key is decrypted using the receiver's private RSA key: k=Ckdmod  nk = C\_k^d \mod n Where CkC\_k is the encrypted AES key.

## Signature Verification

* The verify\_signature() function verifies the digital signature using the sender's public RSA key.
* **Mathematical Concept:** The verification process checks: H(M)=?Semod  nH(M) \stackrel{?}{=} S^e \mod n Where H(M)H(M) is the hash of the message, and SS is the signature.

## AES Decryption

* **Mathematical Concept:** AES decryption formula: P=Dk(C)P = D\_k(C) Where:
  + PP is the plaintext.
  + DkD\_k is the AES decryption function using key kk.
  + CC is the ciphertext.
* The decrypt\_message\_with\_aes() function applies AES-CBC decryption and removes PKCS7 padding.

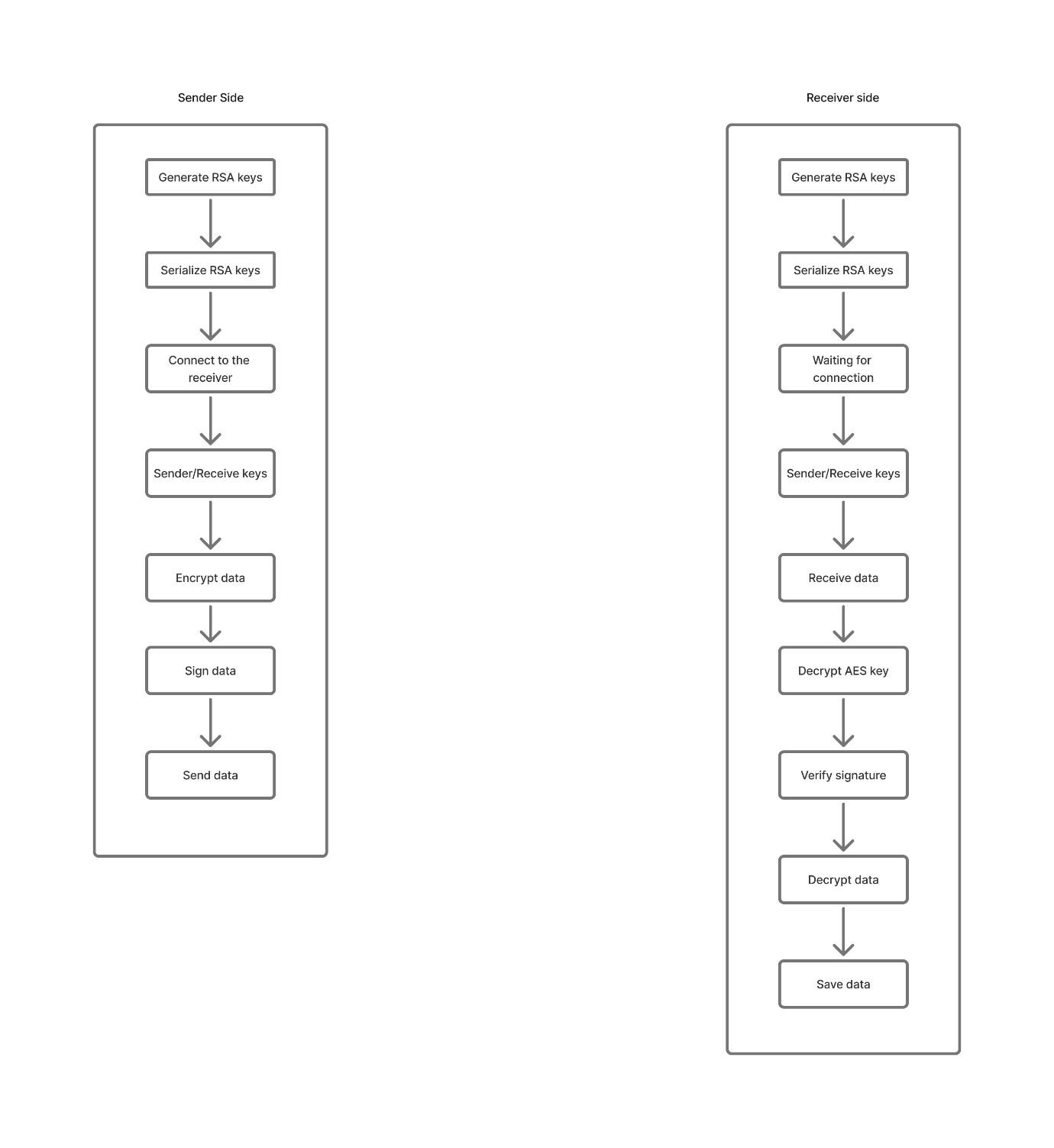
## Data Restoration

* The decrypted message is saved to a CSV file as received\_file.csv.

## Summary

This program demonstrates secure data reception by integrating RSA for key exchange, AES for data decryption, and digital signatures for data integrity verification. The careful handling of message sizes and padding ensures secure and efficient communication.

Flow Diagram



**Resource Usage Test Results**

|  |  |  |
| --- | --- | --- |
| Function | Time (s) | Memory (KB) |
| generate\_rsa\_keys | 0.0270 | 0.05 |
| serialize\_rsa\_keys | 0.0000 | 4.30 |
| encrypt\_message\_with\_aes | 0.0013 | 12.80 |
| sign\_message | 0.0069 | 6.17 |
| decrypt\_message\_with\_aes | 0.0000 | 2.39 |
| verify\_signature | 0.0000 | 1.65 |
| recv\_all | 0.0018 | 1.88 |

**Explanation of Results**

1. **generate\_rsa\_keys**:
   * Time: 0.0270 seconds
   * Memory: 0.05 KB
   * Generates RSA public and private keys.
2. **serialize\_rsa\_keys**:
   * Time: 0.0000 seconds
   * Memory: 4.30 KB
   * Serializes RSA keys into PEM format.
3. **encrypt\_message\_with\_aes**:
   * Time: 0.0013 seconds
   * Memory: 12.80 KB
   * Encrypts a message using AES in CBC mode.
4. **sign\_message**:
   * Time: 0.0069 seconds
   * Memory: 6.17 KB
   * Signs a message using RSA private key.
5. **decrypt\_message\_with\_aes**:
   * Time: 0.0000 seconds
   * Memory: 2.39 KB
   * Decrypts an AES-encrypted message.
6. **verify\_signature**:
   * Time: 0.0000 seconds
   * Memory: 1.65 KB
   * Verifies the digital signature of a message. Signature is valid.
7. **recv\_all**:
   * Time: 0.0018 seconds
   * Memory: 1.88 KB
   * Receives 90 bytes of data (expected 10 bytes).

**Summary**

* The functions perform efficiently with minimal time and memory usage.
* Signature verification confirms the integrity of the transmitted data.
* The recv\_all function successfully handles data reception, even with unexpected sizes.