APT3090 Project Report

# Part 1: RSA Over TCP

## Objective

To implement RSA encryption and decryption over TCP sockets between two parties: Alice (server) and Bob (client).

## Background

RSA is a public-key cryptographic algorithm. Alice generates RSA keys and sends the public key to Bob. Bob encrypts a message with this public key and sends it back. Alice decrypts it using the private key.

## Methodology

RSA parameters: p, q, n = p\*q, φ(n), e, d. Used Python’s socket module for TCP and PyCryptodome for RSA.

## Python Code

alice.py:  
import socket  
from Crypto.Util.number import getPrime, inverse, bytes\_to\_long, long\_to\_bytes  
  
p = getPrime(512)  
q = getPrime(512)  
n = p \* q  
phi = (p - 1) \* (q - 1)  
e = 65537  
d = inverse(e, phi)  
  
server = socket.socket()  
server.bind(('localhost', 12345))  
server.listen(1)  
conn, \_ = server.accept()  
conn.send(f"{n},{e}".encode())  
  
ciphertext = int(conn.recv(2048).decode())  
plaintext = pow(ciphertext, d, n)  
print(long\_to\_bytes(plaintext).decode())  
conn.close()

bob.py:  
import socket  
from Crypto.Util.number import bytes\_to\_long  
  
client = socket.socket()  
client.connect(('localhost', 12345))  
  
key\_data = client.recv(2048).decode()  
n, e = map(int, key\_data.split(','))  
message = input("Enter message: ")  
m = bytes\_to\_long(message.encode())  
ciphertext = pow(m, e, n)  
client.send(str(ciphertext).encode())  
client.close()

## Results

Screenshots of successful message encryption and decryption between Alice and Bob.

## Conclusion

RSA over TCP was successfully implemented using sockets and cryptographic math.

# Part 2: Credit Card Vault

## Objective

To build a secure vault for storing credit card details using MySQL, user roles, and AES encryption.

## Information Classification

Public: Name  
Confidential: Email, Address  
Sensitive: Card Number, CVV, Expiry

## Access Roles

Admin: Full access  
Clerk: Insert & Select  
Auditor: Select only

## Schema in 3NF

CREATE TABLE Customers (  
 customer\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 full\_name VARCHAR(100),  
 email VARCHAR(100)  
);  
  
CREATE TABLE CreditCards (  
 card\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 customer\_id INT,  
 card\_number VARBINARY(255),  
 expiry\_date DATE,  
 cvv VARBINARY(100),  
 FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)  
);  
  
CREATE TABLE Users (  
 user\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 username VARCHAR(50),  
 password\_hash CHAR(64),  
 role ENUM('admin', 'clerk', 'auditor')  
);

## Encryption SQL Scripts

-- Insert encrypted card data  
INSERT INTO CreditCards (customer\_id, card\_number, expiry\_date, cvv)  
VALUES (1, AES\_ENCRYPT('4111111111111111', 'secretkey'), '2026-01-01', AES\_ENCRYPT('123', 'secretkey'));  
  
-- Decrypt  
SELECT AES\_DECRYPT(card\_number, 'secretkey') AS card\_number FROM CreditCards;  
  
-- SHA2 Password  
INSERT INTO Users (username, password\_hash, role)  
VALUES ('john', SHA2('password123', 256), 'clerk');

## Views

CREATE VIEW vw\_cardholders AS  
SELECT c.full\_name, cc.card\_number, cc.expiry\_date  
FROM Customers c JOIN CreditCards cc ON c.customer\_id = cc.customer\_id;  
  
CREATE VIEW vw\_admin\_access AS  
SELECT \* FROM Customers JOIN CreditCards USING (customer\_id);  
  
CREATE VIEW vw\_audit\_log AS  
SELECT full\_name, email FROM Customers;

## Conclusion

Successfully built a secure, role-based credit card vault system with AES and hashed credentials.