# Final Project Report : Cryptography Using DSA in C Programming

**Prepared by:**ByteXL

**Educator:** Jaswanth Parlapalli

## Executive Summary

This report details the development of a cryptographic project focused on implementing Data Structures and Algorithms (DSA) in C programming. The project includes encryption, decryption, and hashing, utilizing core cryptographic techniques to secure data and demonstrate real-world applications of cryptography. Key features involve RSA encryption, hashing functions, modular arithmetic, and prime number generation, emphasizing practical cryptographic methods and secure data handling within a C programming environment.

### 1. Tools and Technologies Utilized

- **Programming Language**: C

- **Data Structures:** Arrays, Dynamic Arrays

- **Algorithms:** Number Theory, Modular Arithmetic, RSA, Hashing

- **Testing and Debugging:** GCC Compiler, GDB Debugger

- **Documentation and Design:** Nimbus Platform

### 2. Project Overview

The project presents a structured cryptographic system built on C programming fundamentals. It covers implementing RSA encryption and decryption, hashing, and digital signatures using data structures and algorithms. The project follows best practices, including modularity, data validation, and error handling, creating a secure, efficient, and educational cryptographic application.

### 3. System Requirements

**Software Requirements**

- GCC Compiler(latest version)

- Nimbus Platform for documentation and testing

- Compatible IDEs: Code::Blocks, Visual Studio Code

**Hardware Requirements**

- Minimum RAM: 2GB

- Processor:Dual-core or better

- Operating System: 64-bit (Linux, Windows)

### 4. Functional Requirements

The project must fulfill the following cryptographic functionalities:

- **RSA Encryption/Decryption:** Perform secure encryption and decryption of data using RSA.

- **Hashing Functions:** Implement hashing mechanisms for data integrity checks.

- **Digital Signature:** Verify authenticity using cryptographic signatures.

- **Key Management:** Generate, store, and manage keys efficiently using arrays.

### 5. User Interface Requirements

The program provides a command-line interface (CLI) that includes:

- **Menu-driven Interface:** Allows users to choose encryption, decryption, hashing, and other cryptographic functions.

- **Error Messages and Feedback:** Real-time feedback for successful or unsuccessful operations.

- **Input Validation:** Ensures input data meets cryptographic requirements.

### 6. Inputs and Outputs

**Inputs**

- Plaintext for encryption

- Encrypted text for decryption

- User-defined keys for encryption

- Prime numbers for RSA key generation

**Outputs**

- Encrypted data output

- Decrypted data output

- Hashed values for input data

- Error and success messages for each operation

### 7. System Subcomponents

The project comprises these primary components:

- **Encryption Subsystem:** Handles RSA encryption and decryption operations.

- **Hashing Subsystem:** Manages hashing operations for data integrity.

- **Key Management Subsystem:** Generates and stores keys for cryptographic functions.

- **Digital Signature Subsystem:** Facilitates data authenticity using signatures.

### 8. Potential for Other Applications

The cryptographic system could be adapted to other areas, including:

- **Secure Communication Systems:** Encry[pting message exchanges for privacy.

- **Data Integrity Verification:** Utilizing hashing for secure file storage.

- **Authentication Systems:** Applying digital signatures for user authentication in software applications.

### 9. Test Case Design

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Function** | **Expected Outcome** |
| **TC001** | Generate RSA Key Pairs | Successfully generated public/private keys |
| **TC002** | Encrypt Data | Data encrypted correctly with RSA |
| **TC003** | Decrypt Data | Original data retrieved after decryption |
| **TC004** | Hash Function | Hash value generated for input data |
| **TC005** | Validate Digital Signature | Signature verification passed |
| **TC006** | Error Handling | Proper error message displayed for invalid input |

### 10. Future Enhancements

Suggested improvements for subsequent versions include:

- **Enhanced Key Management:** Introducing more robust storage and key management.

- **Algorithm Optimization:** Optimize cryptographic algorithms for better performance.

- **Expanded Interface:** Develop a graphical user interface (GUI) for a more user-friendly experience.

### 11. References

- RSA Encryption and Decryption Guides

- Cryptography and Network Security Documentation

- C Programming Best Practices

- Nimbus Platform Usage for Documentation

### 12. Project Reflection

**Technical Challenges Encountered**

- **Prime Number Generation:** Ensuring accurate, efficient generation of prime numbers for RSA.

- **Error Handling:** Managing complex error scenarios across encryption, decryption, and hashing functions.

- **Key Management:** Balancing memory usage while securely storing cryptographic keys in arrays.

**Software Engineering Insights**

Understanding modular design principles and applying data structures efficiently facilitated a smooth development process. The project reinforced key programming concepts, particularly in C.

**Personal Development**

This project improved my knowledge in cryptographic protocols and DSA, emphasizing secure data handling and C programming practices.

**Additional Knowledge Gained**

Future work could benefit from integrating advanced C features and learning assembly-level programming for optimized cryptographic operations.