5

Implementation of RSA algorithm with applied hybrid cryptography and Digital Signature with Hash functions

“Quality is the ally of schedule and cost, not their adversary. If we have to sacrifice quality to meet schedule, it’s because we are doing the job wrong from the very beginning.”

–James A. Ward

5.1 Feasibility Study

The feasibility study is necessary to determine whether the proposed system is feasible considering the technical, operational and economic factors.

5.1.1 Technical Feasibility

The proposed system is implemented in ANSI C. The application is targeted for the Windows platform so the GUI is designed using the Windows API. However the underlying code for the application itself is in ANSI C and requires only an ANSI C compatible compiler for compilation. Other popular OS platforms like UNIX, Linux supports C language so only GUI have to be designed particularly for the OS used.

Despite the high computational costs involved in the RSA algorithm, the hardware requirements have been kept minimum as noted later. The rapid advancement in hardware and technology causes reduction in their costs makes the application feasible.
Since the required hardware and software are available easily in the market domain which makes this application technical feasible.

5.1.2 Economic Feasibility

The purpose of any security application is to protect information assets from misuse and destruction. The loss resulting from the misuse or destruction of valuable confidential data cannot be easily recovered by the individual/organization. If the individual/organization had a security application in place then such losses can easily be avoided. The purposed system guarantees security by means of authentication and data integrity and thus it is economically feasible.

5.1.3 Operational Feasibility

The application has a very easy and user friendly graphical user interface with step-by-step help on each screen of operation. The GUI is dialog base designed and is kept consistent throughout the application. This hides the internal complexity of the procedure from the user. The user interacts only through the well-designed & user friendly interface. Hence the application is operationally feasible.

5.2 System Analysis

Now the comparison between the proposed system with existing systems will be done, point out the weaknesses of the existing systems and finally discuss the advantages provided by the proposed application to justify its requirement. The programming language to be used for the application development is also being discussed.

5.2.1 Existing Systems and their limitations

There is no dearth of securities applications similar to the one proposed but the most popular of those is Pretty Good Privacy (PGP). PGP provides a
wide range of services for the transmission of data in a secure manner. It is a hybrid cryptosystem in that it uses a combination of symmetric and asymmetric algorithms including RSA. It uses a strong symmetric algorithm IDEA firstly to encrypt the data. It uses MD5 hash function for generating digital signatures.

However, PGP provides a broad range of services some of which were not required in this case e.g. symmetric algorithms are not required for encryption of data except for protecting the private key. Also additional features like email plug-ins, digital certificate management, RSA keys of upto 4096 bits were not needed. Key size upto 4096 bits make it slower. MD5 hash algorithm for digital signature has much vulnerability which is stated earlier. The system needed to be modular in nature so that not only it can function as a completely independent application but also can easily be incorporated in a bigger security application. PGP uses continuous random entropy collection in the background which makes it a little slower.

5.2.2 The Proposed system RSAAPP

The proposed system “RSAAPP” is a hybrid cryptosystem as it uses the power of both public and private cryptosystems. It provides RSA key generation of key sizes of 512, 1024 and 2048 bits which are actually required and faster in execution. The private key of generated key pair is being encrypted for its protection using well known algorithm DES with password of 8 characters long. Widely used RSA algorithm is used for encryption, decryption and authentication using digital signatures. The proposed application can be used encrypt any kind of data i.e. text or binary. It uses SHA2, SHA1 and MD5 hash algorithms for the digital signature. SHA2 is much more secure then SHA1 and MD5. The proposed system tries to eliminate problems in the following manner:
1. Only RSA algorithm is used for encryption of data, no symmetric algorithm is used for it. But the symmetric algorithm DES is used to protect the private key as protection of key is very much important.
2. It is completely modular in nature: key generation, encryption and decryption as well as create digital signature at the same time. All these basic functions have been completely modularised and made independent to each other.
3. “RSAAPP” has been designed in such a manner that it can easily be reused or included in a large security application.
4. SHA2 and SHA1 have been provided in addition to MD5. SHA2 is much more secure than SHA1 and MD5.
5. No background random entropy collection algorithm is used to minimize the use of system resources.
6. Any extra features present in the existing system were not included as they were not necessary thus reducing any extra overheads.

The selection of programming language was made keeping the following facts in mind:

1. Should be simple, easy to use and flexible.
2. Should be standardized.
3. Should be easily portable.
4. Most importantly, it should be one of the fastest and widely used.
5. Should be as close as possible to the machine level and allow low level programming.
6. It should be machine independent.
7. Tools for the language should be easily available.
8. Windows application programming interfaces (APIs) should exist.

The C language is like a carving knife: simple, sharp and extremely useful in skilled hands. Like any sharp tool, C can injure people who don’t know how to
handle it by Andrew Koeing (AT & T). The C language clearly emerged as the one which satisfied all of the above factors. C++ can also be considered but have some difficulties for this application e.g. C++ tends to magnify problems that exists in C making debugging even more difficult.

5.3 Requirement Analysis and Specification

The outcome of the system engineering process is the specification of a computer based system or product at the different levels. But the challenge facing system engineers (and software engineers) is profound: How can we ensure that we have specified a system that properly meets the customer’s needs and satisfies the customer’s expectations? There is no foolproof answer to this difficult question, but a solid requirements engineering process is the best solution we currently have.

Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system [123].

The requirement analysis phase translates the ideas of the minds of the clients into a formal document. Technical feasibility centers on existing computer resources and to the extent it can support the proposed system.

5.3.1 Functional requirements

1. **Graphical User Interface**: The user should be provided with an easy to use GUI for performing all the functions provided by the application.

2. **Independent functions**: Each of basic functionality is to be provided independent of each other in the form of separate executables.
3. **Key Generation:** The key generation module should generate private and public keys in pairs using RSA algorithm. If required, the keys can be viewed in hex format after generation. The corresponding private keys generated are encrypted using DES after taking an 8-characters password as user input. Both keys are made read-only; also as a naïve but effective security option.

4. **Encryption:** The user interface should allow user to select any file and any public keyfile for the purpose of encrypting the file. The name of the encrypted file is to be taken from the user.

5. **Decryption:** The user interface should allow the user to select any RSA encrypted file and the corresponding DES encrypted private keyfile for the purpose of decrypting the file. The private keyfile is decrypted using an 8-characters password as user input before being used in the RSA decryption. The name of the decrypted file is to be taken from the user.

6. **Digital Signature generation:** The user interface should allow the user to select any file and any DES encrypted private keyfile for generating the digital signature required for authenticating the file. As before, the private keyfile is decrypted using an 8-characters password as user input before being used in the RSA signing operation. Hash function is used to generate the hash of the file. The signature file is made read-only.

7. **Hash Function:** SHA or SHA1 must be used as the hash function in the generation of digital signature. In addition MD5 may also be provided.

8. **Digital Signature verification:** The user interface should allow user to select any file and its digital signature file and the corresponding public keyfile for purpose of verifying the authenticity of the file.

9. **Editing and Composing:** There should be an editor included for editing and composing purposes.
10. **Status Display**: All the required help, output results and status messages must be displayed in as output box.

### 5.3.2 Non-Functional requirements

1. **Platform and development tool**: The development of the application is to be in Microsoft Visual C++ 6.0 under Windows as the GUI frontend interface is designed in windows.
2. **Fixed key sizes**: Only three key sizes were allowed for key generation - 512, 1024 and 2048 bits. They were to be user selectable through a list box or radio button.
3. **Response time**: The response time should be reasonably fast.
4. **User friendly**: The system should be easy to understand and use.
5. **Error handling**: The response to the user errors (such as missing data) and undesired situations are taken care of to ensure that system operates without halting.

### 5.3.3 Requirement Specification

The RSAAPP security application is developed for the purpose of key generation, encryption, decryption and authentication of any confidential data.

1. **Key Generation**:
   a. Input: The size of key, the choice of public key, 8-characters long password for encrypting private key, name of private keyfile and name of the public keyfile if generating a new public key.
   b. Processing: Use RSA algorithm to generate the keypair (public and private keyfiles are separated). Encrypt the new private key with DES using 8-characters long password and save it with the user specified name. If specified save the new public key with the user specified name.
c. Output: DES encrypted private keyfile with the user specified name and If specified save the new public key with the user specified name.

2. **Encryption:**
   a. Input: Public keyfile, the file to be encrypted and name to save the encrypted file.
   b. Processing: Use RSA encryption to encrypt the specified file with the specified public key.
   c. Output: Encrypted file with user specified name.

3. **Decryption:**
   a. Input: DES encrypted private keyfile, 8-characters long password for decrypt it, the encrypted file to be decrypted and name to save the decrypted file.
   b. Processing: Decrypt the private keyfile. Use RSA decryption to decrypt the specified file with the user’s private key.
   c. Output: Decrypted file with user specified name.

4. **Digital Sign Generation:**
   a. Input: DES encrypted private keyfile, 8-characters long password for decrypt it, the file to be signed and name to save the signed file.
   b. Processing: Decrypt the private keyfile. Generate hash result of the input file and encrypt it with the private keyfile using RSA.
   c. Output: Signature file with user specified name.

5. **Digital Sign Verification:**
   b. Processing: Decrypt the signature file using RSA and the public keyfile. Generate hash result of the input file and compare it with the result of decryption.
   c. Output: Message in the output box stating whether the file has been verified or not.
5.3.4 **Hardware requirements**

Electronic devices that provide computing capability, the interconnectivity devices (e.g., network switches, telecommunications devices) that enable the flow of data, and electromechanical devices (e.g., sensors, motors, pumps) that provide external world function. [22] Minimum hardware requirement is -

- CPU: Pentium IV @ 1 GHz
- RAM: 512 MB

5.3.5 **Software requirements**

Computer programs, data structures, and related documentation that serve to affect the logical method, procedure, or control that is required. [22] Minimum software requirement is Windows XP or higher.

5.4 **System Design**

System design is actually a multi-step process that focuses on four distinct attributes of the program: data structures, software architecture, procedural detail and interface characterization. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. It involves designing files and information processing functions to be performed by the system. This will lead to data flow diagrams, normalized data structures, program specifications etc. It is an iterative process where we start with some structure and refine and enhance it until the designer is satisfied.

5.4.1 **Software Architecture**

Software architecture is an orderly approach that works from the high level to the lowest level details in which the user needs are presented through the
use of hierarchical data flow diagrams and structure charts.

Figure 22: Structure Chart of RSAAPP Application

5.4.2 Input Design

The design of the input includes specifying the means by which users direct the system in which actions to take. The arrangements of messages and comments as well as placement of data, headings and titles are also a part of the input design. The following details are included in the design:

- The data to input
- The dialog to guide users in providing input.
- Data items needing validation to detect errors

The decision should be taken about how the data would be entered i.e. whether it is direct or it is through file. In my application, the input is through a file as specified by the user. For the public key generation, the user has many predefined options, including a public keyfile.

Consistent labels, standard abbreviations and predictable colors are followed throughout the system design.

The interaction of the user with the system is flexible. Consistency has been maintained between the data input and information display. Various
default values are already in place to reduce difficulties on the part of the user. For e.g. the hash algorithm is kept as SHA-2 as default with MD5 as another option.

The system is protected from user data input errors that might cause it to fail. All inputs including files checked for their existence before proceeding further.

5.4.3 Output Design

Output generally refers to the results and information that are generated by the system. In the context of RSAAPP, at the sender’s end the output includes public and private keyfiles, encrypted files or signature files. At the receiver’s end the output includes decrypted files, verification of the signature and the integrity of the received data.

5.4.4 Data Flow Diagrams

Context level Data Flow Diagram

![Context Level Data Flow Diagram of Application](image)

Figure 23: Context Level Data Flow Diagram of Application
Data Flow Diagrams of various constructed modules

Figure 24: DFD of Key Generation Module

Figure 25: DFD of Encryption Module
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 26: DFD of Decryption Module

Figure 27: DFD of Digital Signature Generation Module
5.4.5 Construction of Modules

During the system design phases, we partition the problem into modules. A module can be defined as a system containing a tightly bound group of functional units and their relationships.

Executable Modules
The RSAAPP application was divided into 5 modules from the executable point of view:

1. RSAAPP Module - This module integrates all the other modules and provides a combined access to all functions of RSAAPP through a GUI.
2. Key Generation Module - Used to generate public and private keys in the form of key files.
3. Encryption Module - Used for encrypting any input file using the RSA algorithm.
4. Decryption Module - Used for decrypting any encrypted input file using the RSA algorithm.

5. Digital Signature Generation Module - Used for generation of digital signature of any input file using the RSA algorithm.

6. Digital Signature Verification Module - Used for checking the authenticity of any input file using its digital signature through the RSA algorithm.

Functional Modules
From the functional point of view the application was divided into the following modules:

1. The Large Integer multi-precision library (LINT)
   This provides the functionality of handling multi precision data of large bit sizes. Since C itself provides not more than 32-bit integer handling capabilities, this separate library is created for mathematical operations on data consisting of up to 2048-bit) integers. It consists of 3 files:
   a) Mplint.c - provides basic operations like addition, subtraction, multiplication, division,' and modulus operations on multi-precision data. The 2048-bit integer is actually implemented - using an array of 32-bit unsigned long integers taken as a composite unit.
   b) Splint.c - provides the single precision equivalent of Mplint.c
   c) Lintutils.c - provides functions for printing and file I/O of multi-precision data. This provides random data in the form of large integers to the prime number generation module.

2. The Random number generator
   This provides random data in the form of large integers to the prime number generation module.

3. The Prime number generator
   This module takes random data from the random number generator,
converts it into an odd integer and then tests whether it is Prime and not using the Miller-Rabin probabilistic primality test.

4. The DES module
   This implements the basic DES algorithm in C for encrypting the private keyfiles.

5. The RSA encryption and decryption module
   It provides the basic functionality of the RSA encryption and decryption algorithms.

6. The Hash function module
   It is used to implement the three hash algorithms provided to the user: MD5, SHA1 and SHA2.

7. The Graphical User Interface (GUI) module
   This was the only module visible to the user. Designed with the Win32 API, it provided the user with easy access to all the functionality of RSAAPP. It is based on dialog boxes.

8. Utilities
   An editor, Meta-pad and ‘About software’ are also provided in the application interface.

The interaction between the various functional modules can be well-demonstrated using the module interaction diagram (Fig. 21) as shown below:

![Module Interaction Diagram of RSAAPP](image-url)
5.5 System Testing and Validation

Software testing is a critical element of software quality assurance and represents the ultimate review of specifications, design and coding. In other words, it is the testing of behaviour of a complete and fully integrated software product based on the software requirements specification (SRS) document. In main focus of this testing is to evaluate Business / Functional / End-user requirements.

Customary framework testing and approval of the developed framework is obliged to guarantee that model quality and uprightness is kept up all through the advancement process. This area secures the framework testing measures and the methods that will be utilized to check the models have been met. It is accepted that segment model advancement groups have unit tried their part preceding making it accessible for framework testing. It includes testing of individual parts and unit-testing of individual subroutine and modules inside segments.

It is a type of black box testing in which systems’ external working is evaluated with the help of requirement documents which is Users point of view. Knowledge of internal design/ code structure does not require for this testing. This testing is to be carried out only after System Integration Testing is completed where both Functional & Non-Functional requirements are verified. In the integration testing finds bugs/defects in integrated modules while the Software System Testing finds bugs/defects related to software design, software application behaviour and end-user expectations.

There are two general classes of model assessments: frequent short test runs and infrequent long validation integrations.
Model testing alludes to short (3 to 31 day) model runs intended to check that the basic mechanics and execution of the coupled model keeps on meeting specifications. This incorporates confirming that the model really begins up and runs, benchmarking model execution and relative speed/cost of every model part and in addition watching that the model restarts precisely. These tests are done on each of the target stages/platforms. Model testing does not address whether the model answer is right, it only checks that it mechanically works as determined.

Model validation includes longer (no less than 1 year) incorporations to guarantee that the model results are in worthy concurrence with both past model atmosphere insights and watched attributes of the genuine atmosphere framework. Model approval happens with every minor form or at the solicitation of the framework researchers and working gatherings. When asked for, model acceptance is just completed after framework researchers have been counselled and the model testing stage is effectively finished.

Port validation is defined as verification that the differences between two otherwise identical model simulations obtained on different machines or using different environments are caused by machine round off errors only [124].

The system testing is important as:

a) In Software Development Life Cycle (SDLC) the System Testing is perform as the first level of testing where the System is tested as a whole.

b) In this step of testing check if system meets functional requirement or not.

c) System Testing enables you to test, validate and verify both the Application Architecture and Business requirements.
d) The application/System is tested in an environment that particularly resembles the effective production environment where the application/software will be lastly deployed.

Generally, a separate and dedicated team is responsible for **system testing**. And, System Testing is performed on staging server which is similar to production server. So this means you are testing software application as good as production environment.

### 5.5.1 Different Hierarchical levels of testing

Similarly as with any specialized methodology, software testing has a recommended request in which things ought to be finished. Distinctive levels of testing are utilized as a part of the testing process; every level of testing intends to test diverse parts of the framework. The accompanying is arrangements of software testing classes orchestrated in successively arrange. Fig. 22 below depicts hierarchical levels of testing.

![Hierarchical levels of testing](image)
**Code Walkthrough** – It is a first primary step of formal testing technique where source code is traced by a group with a small set of test cases, while the state of program variables is manually monitored, to analyze the programmer's logic and assumptions. All of the code written for this project was read through thoroughly to pick up any error or inconsistencies that may have arisen. This included omissions in the code that may have resulted in runtime errors or incorrect functionalities. These code walkthroughs also dealt with problems relating to programming style and commenting to increase readability and reusability so that the code could be modified in the future.

**Unit testing** – Testing is done in the development process while developer completes the unit development. The object of this testing is to verify correctness of the module. The purpose of unit testing is to check that as individual parts are functioning as expected. Basically Unit testing is typically carried out by the developer. This testing focuses verification efforts on the smallest unit of software design, *the module*. It is always white-box oriented and the steps can be conducted in parallel for multiple modules. Each of the functional modules stated earlier (except the GUI module) were first developed for DOS and tested in separately for errors [125].

Data flow testing was done to check if any errors occurred during intermediate processing and that the output files were created properly.

Boundary testing was also done to check the actions that took place when user supplied the input for lowest and highest values.

**Integration testing** – it is started after the individual software modules are integrated as a group. A typical software project consists of multiple modules & these are developed by different developers. So in integration testing is focuses to check that after integrating modules if two modules are communicating with each other or not. It is critical to test every module’s
effect on the entire program model. Most of the issues are observed in this type of testing [125].

**System testing** – This is the first time end to end testing of application on the complete and fully integrated software product before it is launch.

**Acceptance testing** – User acceptance is a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This is beta testing of the product & evaluated by the actual end users. The main purpose of this testing is to validate the end to end business flow [125].

**Validation Testing** – The system was tested repeatedly with test cases and was found to functionally correct. The screen shots are given below of sample tests performed on RSAAPP application:
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 31: User Interface of RSAAPP Application
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 32: RSA Key Generation Module of RSAAPP
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 33: Execution of Key Generation Module

![RSA Key Generation Interface](image.png)

- Select Size of key (in Bits): 512, 1024, 2048
- Select Public Key: PK1, PK2, PK3, PK4, PK5, PK6 (New Random Public Key)
- Password for encrypting Private Keyfile (exactly 8 characters)
- Save Generated Keyfiles:
  - C:\Users\DSP\Desktop\test\test.skey
  - C:\Users\DSP\Desktop\test\test.pkey
- Your keypair has been generated.
  - It took 2.734 seconds.
  - You may now want to view it as text or save it as a keyfile for further use.
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 34: RSA Key-pair Generated shown in NOTEPAD
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 35: Encryption Module of RSAAPP

Help:

Step 1: Select Public Keyfile
Step 2: Select Mode of encryption - Text or Binary (Text mode handles encryption of text files containing printable ASCII characters (.txt, .asc, .htm, .xml etc.) Binary mode handles encryption of executables, images, documents (.exe, .ppt, .doc, .xls, .zip, .rtf etc.)
Step 3: Open file to be encrypted
Step 4: Enter the name of encrypted file to be saved
Step 5: Press the Start button.
Implementation of RSA algorithm with applied hybrid cryptography

Figure 36: File to be encrypted (HELP.TXT)
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 37: Output of the encryption Module (Encrypted HELP.TXT file generated in 0.0171 Sec.)
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 38 : Decryption Module of RSAAPP
Figure 39: Output of the decryption Module (Decrypted HELP.DEC file generated in 4.230 Sec.)
Implementation of RSA algorithm with applied hybrid cryptography……

Figure 40: Digital Signature Generation Module (HELP.SIG file generated in 0.421 Sec.)
Implementation of RSA algorithm with applied hybrid cryptography….

Figure 41: Digital Signature Verification Module
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 42: Execution of the Signature Verification Module (Without tempered file)
Implementation of RSA algorithm with applied hybrid cryptography…….

Figure 43: Execution of the Signature Verification Module (With tempered file – HELP (Tempered).TXT)