RSA Encryption & Decryption using JAVA

by

Marliyana Bt. Ramli
3128

A project dissertation submitted in partial fulfillment of
The requirements for the
Bachelor of Technology (Hons)
(Information System)

JANUARY 2006

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

t
QA
76.9
.A25
M348
2006 1) Computer security
2) Cryptography

CERTIFICATION OF APPROVAL

RSA Encryption & Decryption using JAVA

by

Marliyana bt. Ramli

A project dissertation submitted to the
Business Information System Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(BUSINESS INFORMATION SYSTEM)

Approved by,

(Mr. Low Tan Jung)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
January 2006

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

Marliyana bt. Ramli

ABSTRACT

Encryption refers to algorithmic schemes that encode plain text into non-readable form or cyphertext, providing privacy. The receiver of the encrypted text uses a "key" to decrypt the message, returning it to its original plain text form. The key is the trigger mechanism to the algorithm. Until the advent of the Internet, encryption was rarely used by the public, but was largely a military tool. Today, with online marketing, banking, healthcare and other services, even the average householder is aware of encryption. The implementation of this project will be based on Rapid Application Design Methodology (RAD) and will be more focusing on research and finding, ideas and the implementation of the algorithm, and finally running and testing the algorithm. References and theories to support the research of 'RSA Encryption/Decryption using Java' have been disclosed in Literature Review section. The results of the project are discussed in that particular chapter, followed by the conclusion and recommendations.

ACKNOWLEDGEMENTS

First and foremost, Alhamdulillah with the permission from Allah the God Almighty, I manage to complete my Final Year Project (FYP). I would like to convey my highest appreciation to my supportive and dedicated supervisor, Mr. Low Tan Jung, Lecturer of Department Information Technology / Business Information System for their constant guidance and supervision.

I also would like to express my warmest gratitude to all parties who had contributed so much towards my Final Year Project, especially to all UTP lecturers for their encouragement and advices, last but not least my parents and my friends as a whole. Without all the commitments and supports from those parties, I will not be able to complete my Final Year Project, as well as my first degree.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS AND NOMENCLATURES

1. AES	Advanced Encryption Standard
2. CA	Certificate Authority
3. CASE	Computer-aided software engineering
4. CBC	Cipher Block Chaining
5. DES	Data Encryption Standard
6. GSI	Grid Security Infrastructure
7. I/O	Input/Output
8. IDE	Integrated Desktop Environment
9. ISP	Internet service provider
10. JDK	Java Development Kit
11. JVM	Java Virtual Machine
12. MIT	Massachusetts Institute of Technology
13. NSA	National Security Agency
14. PGP	Pretty Good Privacy
15. PKI	Public Key Infrastructure
16. RAD	Rapid Application Development
17. RC5	Rivest Cipher
18. RSA	Ron Rivest, Adi Shamir and Len Adleman
19. SQL	Structured Query Language
20. VHDL	VHSIC Hardware Description Language
21. VM	Virtual Machine
22. XOR	**Exclusive disjunction**, a.k.a **exclusive or**

CHAPTER 1
INTRODUCTION

1.1 Background of Study

Encryption and decryption are common techniques in cryptography, the scientific discipline behind secure communications. Today, encryption has become crucial for secure electronic communication such as credit card transactions over the Internet, email privacy, etc. Can we trust these secure channels? Do they provide sufficient security or do we risk ruining our checking account? This is the important thing for us to learn which of the many encryption methods are secure and which ones we better don't rely on.

Encryption refers to algorithmic schemes that encode plain text into non-readable form or cyphertext, providing privacy. The receiver of the encrypted text uses a "key" to decrypt the message, returning it to its original plain text form. The key is the trigger mechanism to the algorithm.

Web browsers will encrypt text automatically when connected to a secure server, evidenced by an address beginning with *https*. The server decrypts the text upon its arrival, but as the information travels between computers, interception of the transmission will not be fruitful to anyone "listening in." They would only see unreadable gibberish.

There are many types of encryption and not all of it is reliable. The same computer power that yields strong encryption can be used to break weak encryption schemes. Initially, 64-bit encryption was thought to be quite strong, but today 128-bit encryption is the standard, and this will undoubtedly change again in the future.

Encryption can also be applied to an entire volume or drive. To use the drive, it is "mounted" using a special decryption key. In this state the drive can be used and read normally. When finished, the drive is dismounted and returns to an encrypted state, unreadable by interlopers, Trojan horses, spyware or snoops. Some people choose to keep financial programs or other sensitive data on encrypted drives.

Encryption schemes are categorized as being **symmetric** or **asymmetric**. Symmetric key algorithms such as Blowfish, AES and DES, work with a single, prearranged key that is shared between sender and receiver. This key both encrypts and decrypts text. In asymmetric encryption schemes, such as RSA and Diffie-Hellman, the scheme creates a "key *pair*" for the user: a public key and a private key. The public key can be published online for senders to use to encrypt text that will be sent to the owner of the public key. Once encrypted, the cyphertext cannot be decrypted except by the one who holds the private key of that key pair. This algorithm is based around the two keys working in conjunction with each other. Asymmetric encryption is considered one step more secure than symmetric encryption, because the decryption key can be kept private.

Strong encryption makes data private, but not necessarily *secure*. To be secure, the recipient of the data -- often a server -- must be positively identified as being the approved party. This is usually accomplished online using digital signatures or certificates.

As more people realize the open nature of the Internet, email and instant messaging, encryption will undoubtedly become more popular. Without encryption, information passed on the Internet is not only available for virtually anyone to snag and read, but is often stored for years on servers that can change hands or become compromised in any number of ways. For all of these reasons encryption is a goal worth pursuing.

1.1.1 RSA Algorithm

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977 [RIVE78]. The basic technique was first discovered in 1973 by Clifford Cocks, a British mathematician working for GCHQ, described an equivalent system in an internal document. Given the relatively expensive computers needed to implement it at the time it was mostly considered a curiosity and, as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1997 due to its top-secret classification.

The algorithm was patented by MIT in 1983 in the United States of America as U.S. Patent 4,405,829. It expired on 21 September 2000. Since the algorithm had been published prior to patent application, regulations in much of the rest of the world precluded patents elsewhere. Had Cocks' work been publicly known, a patent in the US would not have been possible either.

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring very large numbers, and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving them. Providing security against *partial* decryption may require the addition of a secure padding scheme.

The key length for a secure RSA transmission is typically 1024 bits. 512 bits is now no longer considered secure. For more security, use 2048 or even 4096 bits. With the faster computers available today, the time taken to encrypt and decrypt even with a 4096-bit modulus really isn't an issue anymore. In practice, it is still effectively impossible for user or to crack a message encrypted with a 512-bit key. An organization like the NSA who has the latest supercomputers can probably crack it by brute force in a reasonable time, if they choose to put their resources to work on it. The longer your information is needed to be kept secure, the longer the key you should use. If we are encrypting the plaintext with a conventional symmetrical algorithm like DES, our session key is going to be 64 bits long. Triple DES will need 192 bits, and AES will need up to 256 bits. That gives us lots of security.

1.1.2 JAVA language

Java has gained enormous popularity since it first appeared. Its rapid ascension and wide acceptance can be traced to its design and programming features, particularly in its promise that you can write a program once, and run it anywhere. Java was chosen as the programming language for this project. As stated in Java language white paper by Sun Microsystems: "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic." Below are the characteristic of JAVA that makes the language become a perfect choice for this project.

Security

Java is one of the first programming languages to consider security as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind. The compiler, interpreter, and Java-compatible browsers all contain several levels of security measures that are designed to reduce the risk of security compromise, loss of data and program integrity, and damage to system users. Considering the enormous security problems associated with executing potentially entrusted code in a secure manner and across multiple execution environments, Java's security measures are far ahead of even those developed to secure military systems. C and C++ do not have any intrinsic security capabilities.

Reliability

Security and reliability go hand in hand. Security measures cannot be implemented with any degree of assurance without a reliable framework for program execution. Java provides multiple levels of reliability measures, beginning with the Java language itself. Many of the features of C and C++ that are detrimental to program reliability, such as pointers and automatic type conversion, are avoided in Java. The Java compiler provides several levels of additional checks to identify type mismatches and other inconsistencies. The Java runtime system duplicates many of the checks performed by the compiler and performs additional checks to verify that the executable byte codes form a valid Java program.

The Virtual Machine: Java VM

This VM sits, metaphorically, between the Java program and the machine it is running on, offering the program an "abstract computer" that executes the Java code and guarantees certain behaviors regardless of the underlying hardware or software platform. Java compilers thus turn Java programs not into assembly language for a particular machine but into a platform-neutral "byte code" that the machine-specific VM interprets on the fly.

The Java VM also enforces security policies, providing a sandbox that limits what the Java program can do. A Java applet cannot, for example, peek into arbitrary files on the machine it's running on. The most recent version of Java from Sun, known as Java Development Kit (JDK) 1.1, though, provides no consistent method for an applet to request restricted system resources. This capability will be available in JDK 1.2 or later versions.

Java is Robust

Robust means reliable and no programming language can really assure reliability. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages. Java eliminates certain types of programming constructs in other languages that are prone to errors. For instance, Java does not support pointers, which eliminates the possibility of overwriting memory and corrupting data. Java has a runtime exception-handling feature to provide programming support for robustness, and can catch and respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs.

1.2 Problem Statement

1.2.1 Problem Identification

By analyzing the current situation, problem that can be identified are the implementation of the algorithm, to encrypt and decrypt messages, which we call; cryptography. In cryptography, size does matter. The larger the key, the harder it is to crack a block of encrypted data. The reason that large keys offer more protection is almost obvious; computers have made it easier to attack ciphertext by using brute force methods. Although the impact is slower in processing encrypt and decrypt data, it is guaranteed secured. Cryptography not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions. In all cases, the initial unencrypted data is referred to as *plaintext*. It is encrypted into *ciphertext*, which will in turn be decrypted into usable plaintext.

Conventional Encryption

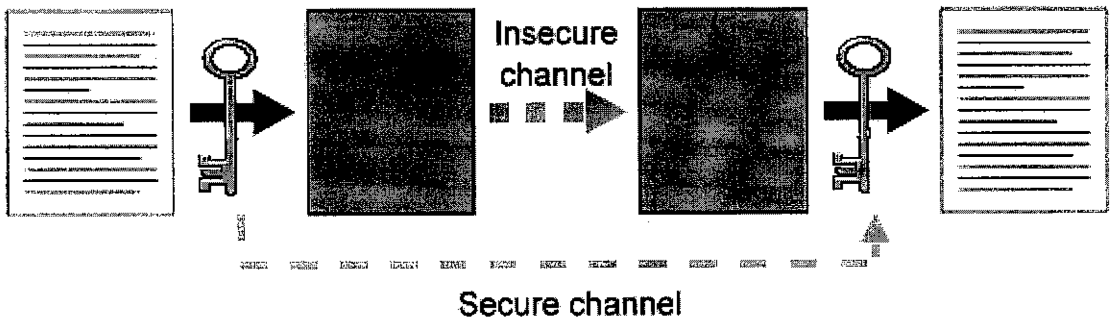


Figure 1.2.1: Conventional Encryption

- Uses a shared key
- Problem of communicating a large message in secret is reduced to communicating a small key in secret.

Public-key Encryption

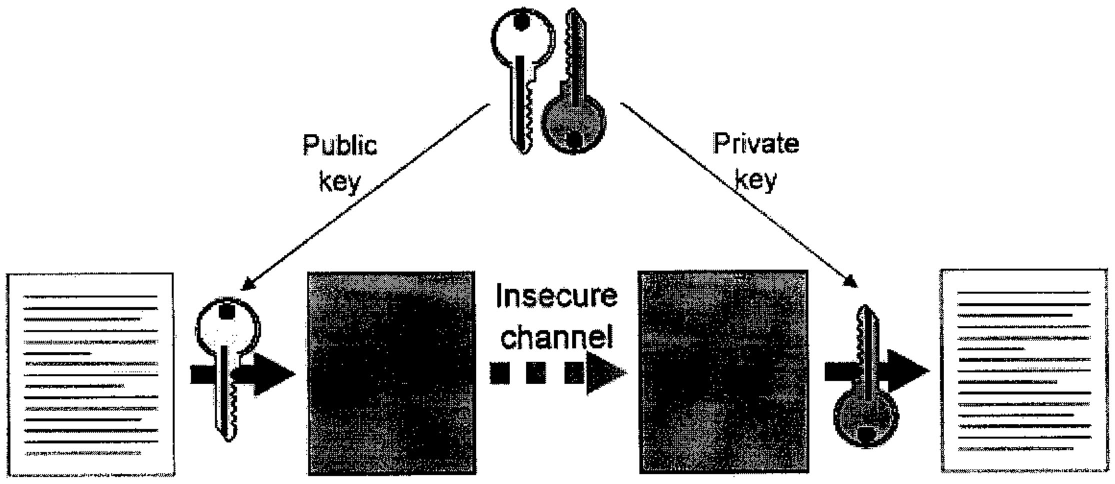


Figure 1.2.2: Public- key encryption

- Uses matched public/private key pairs

- Anyone can encrypt with the public key, only one person can decrypt with the private key

Key Agreement

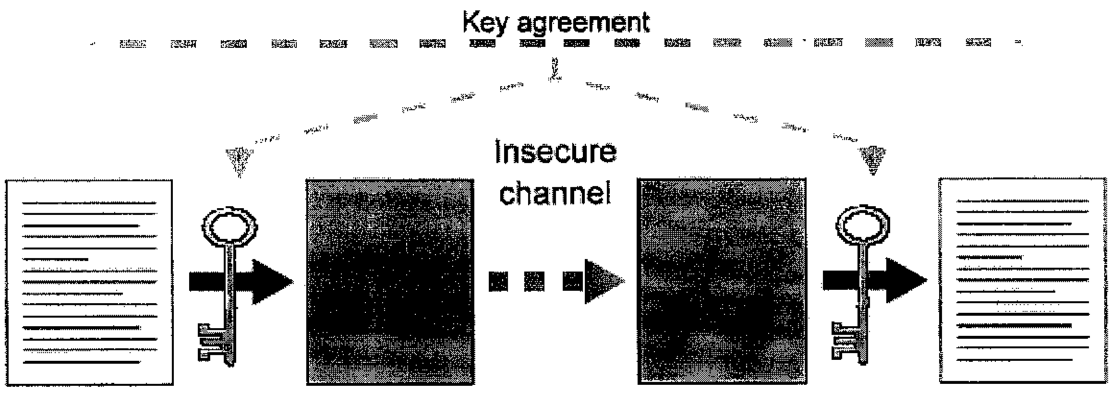


Figure 1.2.3: Key agreement

- Allows two parties to agree on a shared key
- Provides part of the required secure channel for exchanging a conventional encryption key

Hash Functions

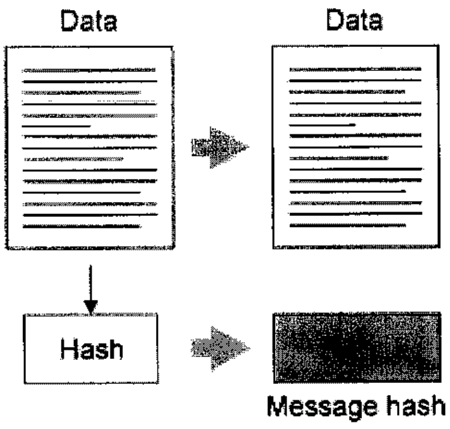


Figure 1.2.4: Hash functions

- Creates a unique "fingerprint" for a message
- Anyone can alter the data and calculate a new hash value

  -Hash has to be protected in some way

For this project, the cryptography scheme that will be used is public-key cryptography. The problem that has been identified is listed as below:

- The implementation of RSA algorithm. The algorithm involving mathematical problems; factoring large integers.

1.2.2 Significant of the Project

The significant of the project is to provide a secured and trustable system for user. With the information technology that rapidly changing nowadays, security matters should be move parallel with it in order to maintain security and privacy of users in the world of no barriers. The implementation of the algorithm (this project) is one of the ways to prevent the data being read or kept by other person.

1.3 Objective and Scope of Study

1.3.1 Objectives

The objectives of this project are:

1. Learn the most prominent classical and modern ciphers to understand how modern encryption techniques can protect our privacy.
2. To learn how RSA encryption work using Java.
3. To implement the RSA algorithm.

1.3.2 Scope of study

The scope of study for this project is to implement a system that can encrypt and decrypt message using a certain key which is, in this case, RSA. As what been stated earlier, encryption is one of the issues nowadays becoming a crucial and important concern to protect your data over the internet.

Throughout the research done for this project is how the RSA algorithm works using Java language. The algorithm needs to be understood in order to implement the codes for encrypt and decrypt the messages.

RSA, as asymmetric encryption uses a separate key for encryption and decryption. The decryption key is very hard to derive from the encryption key. The encryption key is public so that anyone can encrypt a message. However, the decryption key is private, so that only the receiver is able to decrypt the message. It is common to set up "key-pairs" within a network so that each user has a public and private key. The public key is made available to everyone so that they can send messages, but the private key is only made available to the person it belongs to.

As what can be concluded, the scope of study for this project is more to research and findings of how to implement the algorithm and to understand the algorithm itself. As the output for the project, a working code of the encryption using RSA keys will be implemented and will be showed.

CHAPTER 2
LITERATURE REVIEW AND/OR THEORY

"Whether you realize it or not, someone is watching every email and transmission you send on the Internet. If you don't believe me, I would encourage you to read up on the Echelon project, http://www.heise.de/tp/english/inhalt/te/6929/1.html and on Carnivore, http://commons.somewhere.com/rre/2000/RRE.Public.Demo.of.Carni.html. The Echelon is an international project run by the National Security Agency that is supposedly capable of intercepting all communications around the globe. Additionally, it was recently disclosed that the Carnivore system is being installed strategic locations at ISP data centers. The Carnivore is a box that's capable in sorting through Internet traffic to capture the traffic of 'suspects'.

If you still don't believe me, try sending a stream of threatening messages to a friend via email with keywords like "nuclear bomb" and "assassination" or "chemical and biological weapons" and see what happens. In any case, the need for encryption is becoming very important. There's nothing to keep your ISP from reading all of your email and watching you surf the Web. So my advice is, when sending anything over the Internet that may have sensitive information, encrypt it". **(Jonathan Eisenzopf, 2000)[1]**

"Adding that using RC5-64 cipher with longer key sizes such as 128 bits makes it far more difficult to find a secret key. With a group of other cryptographers, suggested that users employ keys of at least 90 bits for symmetric cryptosystems such as RC5. Adding one bit to the length of a key doubles the number of possible keys". **(Ron Rivest)**

"The cracking of DES is of critical importance for ecommerce, the Internet, and the World Wide Web. DES is the accepted cryptographic standard currently used by government and commercial financial institutions to protect important financial data and information, for example, routine currency transfers between national commercial banks. Encryption is the key to so much of the new Net age, and it is imperative that the forces holding it in check be defeated, the same way DES was cracked, by brute force, if necessary". **(BusinessTech Editorial)**

"In contrast to the cooperative preparations required for setting up private key encryption, such as secret-sharing and close coordination between sender and receiver, you can act entirely on your own to create and publish two numbers that enable anyone, using the RSA encryption formula, to send a private message to you through a public channel. The message becomes "First Class" e-mail, so to speak, as if sealed in an envelope. Using the two numbers you have published, anyone can scramble a message and send it to you. You are the only one who can unscramble it--not even the sender of the message can decrypt the ciphertext". **(Jack Dennon)**

"The best way to understand asymmetric encryption is to think of a box that has two kinds of keys: one key locks it and the other unlocks it. Anybody who has a copy of the locking key (aka public key) can put a secret in the box. This is different from symmetric key encryption, in which the same key is used for locking and unlocking. The real complications arise when you ask such questions as 'How do I generate an RSA key pair?' or 'How large do the numbers need to be for security?' The answers to these questions complicate RSA implementations a hundred times over". **(James Tandon)**

"A part of the security aspect is encryption. Often people think that security is "just" something you plug in afterwards – it is definitely not! A few rules of thumb when encryption is going to be included in the final product can be summarized into the following basics: (1) Do not base the encryption on the algorithm itself. (2) Make the algorithm public and the key private. The RSA encryption is typically using CBC mode (Cipher Block Chaining mode) when encrypting. This means the text that is being encrypted is divided into blocks. Each block is chained together, using the XOR operator, and then encrypted". **(Jessn)**

“But when today someone mentions asymmetric cryptography, the RSA-standard is usually meant. With RSA, each user has a pair of keys. The public key can be exchanged openly because it is worthless without the private key which each user keeps for decryption. To make this system work, there has to be a mathematical relationship between the two keys. This relation is a rather complex one. In the case of RSA, it is based on multiplying very large prime numbers. Still, the known nature of this relationship and the public key offer some clues for a hacker”. **(Tech Spotlights)**

“The DES algorithm uses a 56-bit encryption key, meaning that there are 72,057,594,037,927,936 possible keys. The DES Key Search Project developed specially designed hardware and software to search 90 billion keys per second, determining the key and winning the $10,000 RSA DES Challenge after searching for 56 hours”. **(Paul Kocher).**

“RSA Public-Key Cryptography needs large integers for reasonable security. The 32-bit or 64-bit integers available on most machines just aren't big enough. Therefore, the RSA Public-Key Cryptography package uses another package, called the Multiple-Precision Unsigned Integer Arithmetic, to do its arithmetic. In this package, the number of bits can be any multiple of 16. A 512-bit key is considered at least moderately secure; 1024 bits are preferred. The package will, in theory at least, handle any key size which is an even multiple of 16, up to the point where the computer runs out of memory. However, the computations for keys more than 1024 bits long are very slow, even on today's fastest computers”. **(Philip J. Erdelsky)**

“Represented by the equation "$c = m^e \bmod n$," the RSA algorithm is widely considered the standard for encryption and the core technology that secures the vast majority of the e-business conducted on the Internet. The U.S. patent for the RSA algorithm (# 4,405,829, "Cryptographic Communications System And Method") was issued to the Massachusetts Institute of Technology (MIT) on September 20, 1983, licensed exclusively to RSA Security and expires on September 20, 2000”. **(HIPAAdvisory.com)**

"So much misinformation has been spread recently regarding the expiration of the RSA algorithm patent that we wanted to create an opportunity to state the facts. RSA Security's commercialization of the RSA patent helped create an entire industry of highly secure, interoperable products that are the foundation of the worldwide online economy. Releasing the RSA algorithm into the public domain now is a symbolic next step in the evolution of this market, as we believe it will cement the position of RSA encryption as the standard in all categories of wired and wireless applications and devices. RSA Security intends to continue to offer the world's premier implementation of the RSA algorithm and all other relevant encryption technologies in our RSA BSAFE® software solutions and we remain confident in our leadership in the encryption market". **(Art Coviello, chief executive officer of RSA Security)**

"An asymmetric algorithm, is a *trap door one-way* function. A one-way function is easy to perform in one direction, but difficult or impossible to reverse. A trap door one-way function, is one that is easy to reverse if you have information about the trap door, but difficult or impossible to reverse if you lack that information. In symmetric cryptography, the same key is used for both encryption and decryption. This approach is simpler but less secure since the key must be communicated to and known at both sender and receiver locations"**. (Diffie-Hellman)**

"Compared with native code, Java VMs are excruciatingly slow. ... Java still cannot compete with natively compiled C++ code." (PC Magazine, April 7, 1998, 104). The difference in speed between C++ and Java is very important. Even with all of Java's benefits, Java will not be widely accepted if it can not perform adequately. C++ has been widely adopted by developers and they will not be willing to change languages if the applications they develop with Java do not measure up to their personal and their clients standards. However, if the speed difference is negligible, developers may be willing to learn and program in Java because of the significant advantages the language offers. Before developers can make this decision, they need an accurate picture of what the speed tradeoffs between the two languages are. Smallest collection of available development tools (although this is changing). Language is still immature compared to alternatives". **(PC Magazine)**

"Java, being an interpreted system, is currently an order of magnitude slower than C. Unlike natively compiled code, which is a series of instructions that correlate directly to a microprocessors instruction set, an interpreter must first translate the Java binary code into the equivalent microprocessor instruction. Obviously, this translation takes some amount of time and, no matter how small a length of time this is, it is inherently slower than performing the same operation in machine code". **(Just Java, 302).**

"**Phi** (upper case Φ or Φ; lower case φ, ϕ, φ or φ) is the 21st letter of the Greek alphabet. In Modern Greek it is pronounced *fee*, but a common anglicized pronunciation is *fie*. In Modern Greek, it represents [f], a voiceless labiodental fricative. In Ancient Greek it represented [p^h], an aspirated voiceless bilabial plosive. In the system of Greek numerals it has a value of 500.

The lower-case letter φ (or often its variant, ϕ) is used as a symbol for:

- The golden ratio 1.618... in mathematics, art, and architecture.
- Euler's totient function in number theory. Also called Euler's phi function, φ(n)
- The argument of a complex number in mathematics.
- The value of a plane angle in physics and mathematics.
- Electric potential in physics.
- The work function in electronics.
- The phase of a wave in signal processing.
- In spherical coordinates phi is usually used to represent the angle to the z axis.
- Any function in mathematics.

The upper-case letter Φ is used as a symbol for;

- In engineering, the diameter symbol ∅is often referred to as "phi". This symbol is used to indicate the diameter of a circular section, for example ∅14 means the diameter of the circle is 14 units.
- In structural engineering, Φ is notation for a strength (or resistance) reduction factor, used to account for statistical variabilities in materials and construction methods.
- The magnetic flux in physics.
- The Cumulative Normal Distribution function in statistics.
- It is also used as a symbol/icon for philosophy". **[2]**

The applications of encryption:

1) Alchemi

While the performance of enterprise grid symmetric key cryptography that was implemented using Alchemi shows an increase over the single processor version of the symmetric key cryptography, the performance improvement is limited by the I/O and communication overhead. The use of high performance networks can enhance performance. Another way increase performance to transfer the data directly between the user host and executors. However, it violates the current Alchemi security model and requires enhancement of Alchemi security to support rights delegation.

Alchemi is a .NET based grid computing framework developed at the University of Melbourne. It is an open source project which provides middleware for creating an enterprise grid computing environment by harnessing Windows machines. Alchemi supports multithreaded parallel operation in a manner similar to threading in Java or C#, but with their execution on distributed resources. The parallelism is realized at thread level and the programmer has to identify functions to be parallelized and implement them in the form of threads. Currently, inter-thread communication is not supported, so threads must be independent.

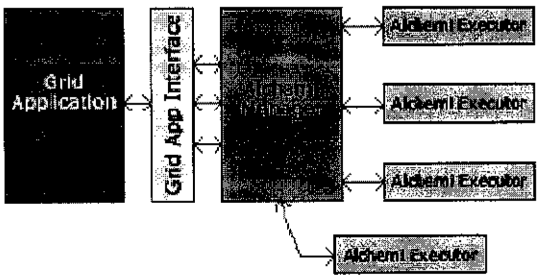


Figure 2.1: Alchemi's main components

A deployment scheme for Alchemi is shown in Figure 4.2.1. Its main components are manager and executor that support a master-worker parallel model. Alchemi has a number of features that ease the process of setting up of a grid environment in an enterprise. The executors can be setup in dedicated or non dedicated mode on employees' desktop computers. In non-dedicated mode, Alchemi has no impact on the workstation as far as the user is concerned. The Alchemi manager also requires a Microsoft SQL Server instance, which is available in most companies. [3]

2) DocLock

DocLock was released in 2005. DocLock stores your sensitive information on your phone, encrypted with password protection. The application is free, but the developer requests that you pay to support development.

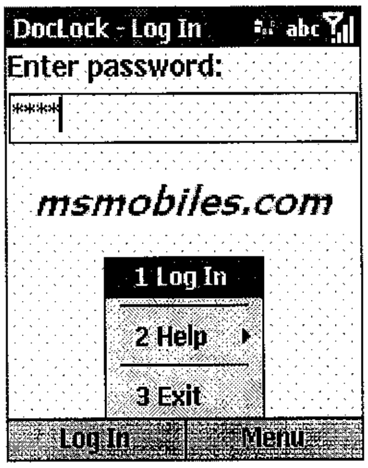


Figure 2.2: DocLock interface

From the developer: DocLock stores your sensitive information in a safe place – always at hand - protected by a single password. After entering and confirming your password, you will be able to add, edit and remove pieces of sensitive information, organized in folders, which will be stored using strong 192 bit TripleDES encryption.

Advanced features include monitoring failed "; Log In"; attempts and "Application Lock Out" whenever too many unsuccessful attempts have occurred. These settings are fully customizable within the application.

From the security perspective, TripleDES keys are using random byte padding for added security of your password. Further, your password is stored using irreversible MD5 hashing algorithm, meaning not even the makers of DocLock is capable of getting your password. Finally 2 minute inactivity Auto Log Out timer makes sure DocLock does not just "keep running" in the background of your phone. [4]

3) PGP Whole Disk Encryption

The PGP Whole Disk Encryption product line provides transparent full disk, volume, and archive encryption as a centrally managed solution or a stand-alone client.

Mobile computers are quickly emerging as the industry standard for increasing user productivity and efficiency. The portable nature of these devices also increases the possibility of loss or theft. Operating system login authentication alone cannot protect sensitive data on disks. If a system is ever stolen or lost, an enterprise may be exposed to significant risk of financial loss, legal penalties, and brand damage.

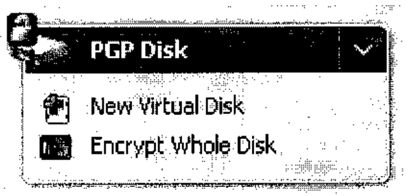


Figure 2.3: PGP Disk Encryption interface

PGP Whole Disk Encryption for Enterprises locks down the entire contents of a laptop, desktop, external drive, or USB flash drive, including boot sectors, system files, and swap files. Encryption runs as a background process that is transparent to the user, automatically protecting valuable data without requiring the user to take additional steps. [5]

CHAPTER 3
METHODOLOGY/PROJECT WORK

3.1 Procedure Identification

After researches, studies, and some considerations had been performed, the most suitable methodology for this project is Rapid Application Development (RAD). In general, the methodology is defined as a software development process that allows usable systems to be built in as little as 60-90 days, often with some compromises.

The methodology is an increment software development process model that emphasizes an extremely short development cycle. The RAD model is "high speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods, as mentioned above – 60 to 90 days.

RAD usually embraces object-oriented programming methodology, which inherently fosters software re-use. The most popular object-oriented programming languages, C++ and Java, are offered in visual programming packages often described as providing rapid application development.

As mentioned above, Rapid Application Development has two primary advantages: increased speed and increased quality. The speed increases are due to the use of CASE tools, the goal of which is to capture requirements and turn them into usable code as quickly as possible. Quality, as defined by RAD, is defined as both the degree to which a delivered application meets the expected objectives as well as the degree to which a delivered system has low maintenance costs.

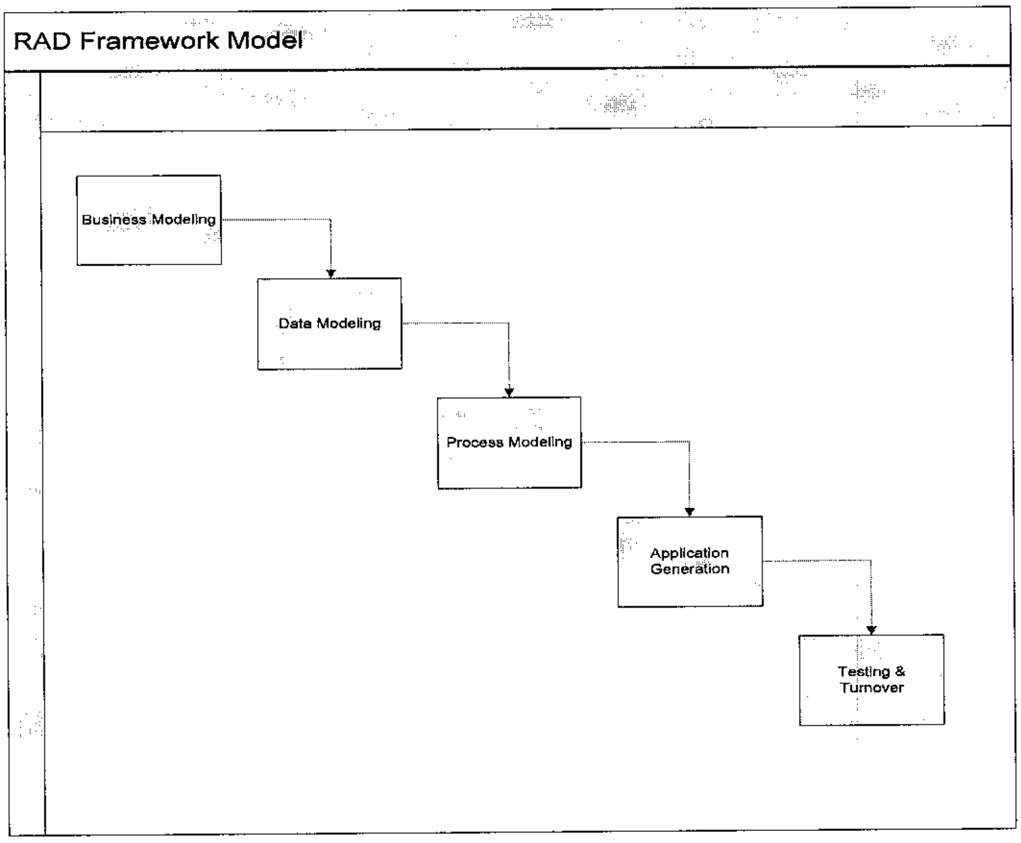


Figure 3.1.1: Rapid Application Development Framework Model

RAD (Rapid Application Development) as depicted in the above Figure 3.1.1 is a concept that products can be developed faster and of higher quality through the process flows specified:

I. Business Modeling

The information flow among business functions is modeled in a way that answers the following question: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

It is the first stage in the Rapid Application Design (RAD) Methodology Life-cycle. During this stage an outline of the system area and definition of the system scope are developed.

Those identified outlines or specifications for Encryption & Decryption using Java's project are:

1) The mechanisms to implement the encryption and decryption system, using public-key encryption mechanism. Using both keys; public key and private key.

2) The RSA algorithm itself. What differentiate RSA from other algorithm is the security that can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

3) Generate secret keys, encrypt/decrypt message that transferred over the internet. This will assist business personnel to interpret the results or outcomes secured, hence come out with appropriate and trusted security of data integrity.

II. Data Modeling and Process Modeling

To describe the processes involved in RAD that meets the specification in implementing RSA Encryption/Decryption using Java, the framework is as attached at the next page:

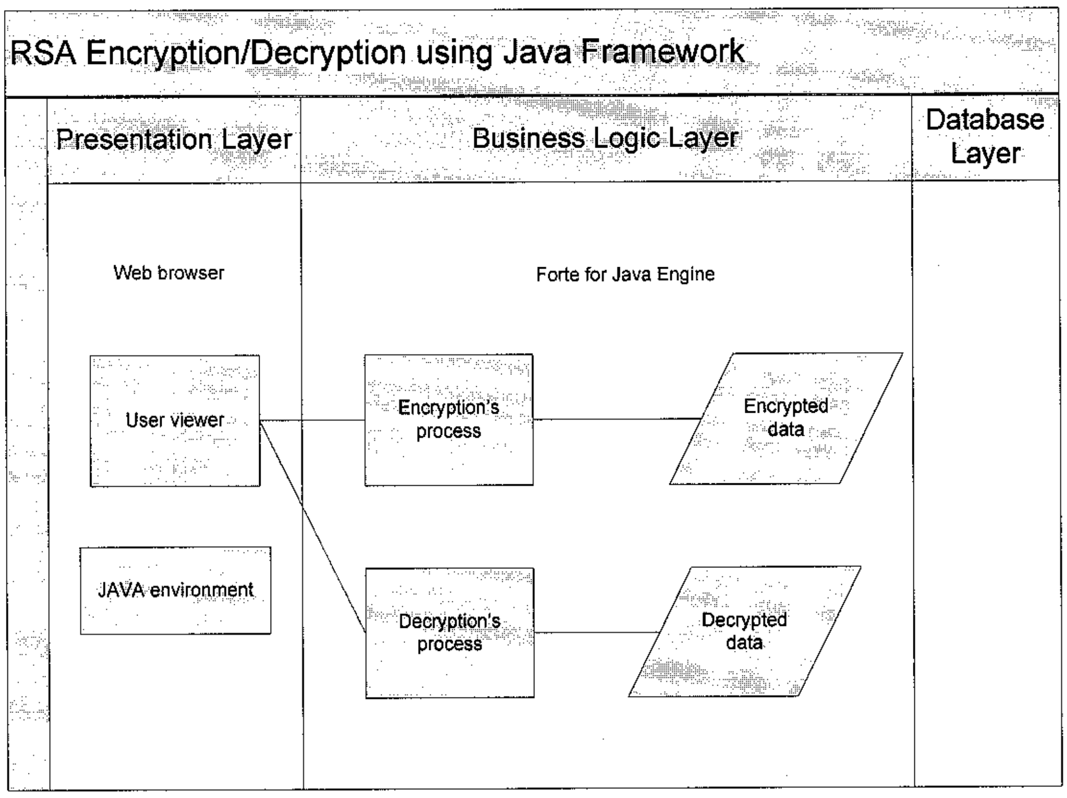


Figure 3.1.2: RSA Encryption/Decryption Back End Framework

At this stage, the framework has been divided into 3 layers which are:

Presentation

This layer will be the front-end of the framework where users can key in the prime number that have been selected by them and encrypt the message. As for the recipient, the interface will help them to decrypt the ciphertext after the secret key has been put by them.

Business Logic

For the middle part of the framework, Forte for Java is the engine for encrypt and decrypt process. It encrypts the key, unscramble the message to the number (ciphertext) and provide a secret key which only given and kept to the recipient.

During this business logic layer, the process of decryption also happened. It unscrambles back the number to the message that can be read by the recipient.

Database

This project doesn't seriously involve with any database.

III. Application Generation

RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages, the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

The construction or implementation of the project is referred to the framework model established at the earlier stage, which is during the framework design stage.

Recommendable, the project should be executed by developing the system in grid computing system. The system would be in form of web-based and Linux operating system. It applicable for user who wants to send data over the networking of a group of computers, and they have to encrypt the message first before sending it on the network. Only authenticate users or the recipient will be allowed to read the message. It is simply because; they have the private key to decrypt the message. Other users using the same network only can read unrecognized text. But due to time constraint and hardware matters, the project cannot be implemented.

IV. Testing and Turnover

Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

The transition stage will be executed after those three main stages have been successfully conducted. The encrypt/decrypt codes will be deployed and some testing will be performed to test on the features and the functionalities of the project completed.

Testing will be done at various stages throughout the development of the project; in fact, it is this testing that will prove or disproved the feasibility of the encryption scheme. Initial tests will be performed to analyze the key size requirements and the online-attack time that a potential hacker would have. Multiple tests will then be run on the brute-force key acquisition model, in order to get a mean time for breaking the encryption. Finally, extensive testing (for example, run on the VHDL implementation) to ensure that all encryption and decryption works correctly.

3.2 Tools Required

3.2.1 Hardware

1. Desktop computers (Pentium 4 2.4 GHz, 128MB RAM, 40Gb hard disk space) Internet connection.

3.2.2 Software

1. Forte for Java

The installation of Forte for Java is using Java Development Kit and Sun One Studio/Forte. And below are the characteristics of Forte for Java:

- A powerful, extensible, integrated development environment (IDE) for developing Java programs.
- Based on NetBeans technology
- Open source, modular IDE written in the Java language
- Can run on any platform with a Java Virtual Machine.

2. Java Virtual Machine

A Java virtual machine (JVM), an implementation of the Java Virtual Machine Specification, interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions. Java was designed to allow application programs to be built that could be run on any platform without having to be rewritten or recompiled by the programmer for each separate platform. A Java virtual machine makes this possible because it is aware of the specific instruction lengths and other particularities of the platform.

The Java Virtual Machine Specification defines an abstract rather than a real machine or processor. The Specification specifies an instruction set, a set of registers, a stack, a "garbage heap," and a method area. Once a Java virtual machine has been implemented for a given platform, any Java program (which, after compilation, is called bytecode) can run on that platform. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to a real processor instruction) or the bytecode can be compiled further for the real processor using what is called a just-in-time compiler.

CHAPTER 4
RESULTS AND DISCUSSION

The result of the project will be discussed generally in the first half chapter which consists of less detail regarding the system and the interfaces. While for another half of the chapter, the discussion is more on implementation of the codes. This is where the discussion be more details on the algorithm and the coding using Java language. The full interface of 'RSA Encryption & Decryption using Java' system has been attached under appendices section.

4.1 Results

Below is a Java applet that allows a user to create RSA keys and encrypt and decrypt text or numbers. It is for educational purposes only and is not industrial strength for several reasons. It also uses basic psuedo-random number generation. The applet can be used for three tasks - key generation, encryption, and decryption.

4.1.1 Key generation

To use the RSA cipher for public key cryptography, two sets of keys are required. RSA uses two different but related keys for encryption and decryption.

There is a public key which consists of an encryption value **'e'** and a **modulus 'n'.** There is also a private key made up of **'d'**, decryption value and the same **'n'** modulus. The algorithm to generate the key pair is as follows:

- Choose two large prime numbers **p** and **q.**

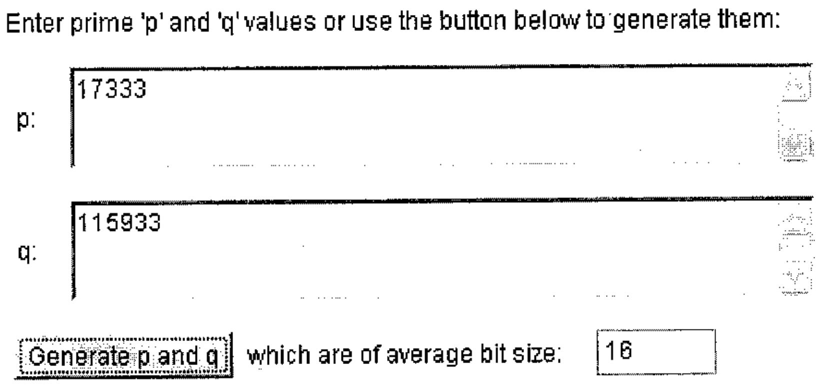
- Compute **n,** the product of **p** and **q. n** is known as the **modulus.**
- Choose a number **e**, relatively prime to **(p-1)(q-1)** and less than **n.**
- Compute a number **d** such that **ed = 1 mod (p-1)(q-1).**

The number **e** is called the **public exponent** and the number **d** is called the **private exponent.**

The **public key** is the pair **(n,e)** and the **private key** is the pair **(n,d).**

Given the public key it is possible to derive private key, but to do this we need to factorize **n** to find **p** and **q**, and this is believed to be an intractable problem for sufficiently large **n.** A quick method of factorizing large numbers would undermine the security of RSA.

To start, first enter two prime numbers into the **'p'** and **'q'** text fields. Or, click on the **"Generate p and q"** button and the applet will create two prime numbers that are any size you select greater than 3 bits long. Bigger numbers are, in general, more secure (512 will create an 'n' of size 1024, which is standard) but numbers too large will take a while to generate.


Enter prime 'p' and 'q' values or use the button below to generate them:

p: 17333

q: 115933

which are of average bit size: 16

Figure 4.1.1: Interface 1; Generate 'p' and 'q'

Next, click on the **"Generate n"** button. It multiplies **'p*q'** and will give you 'n.'

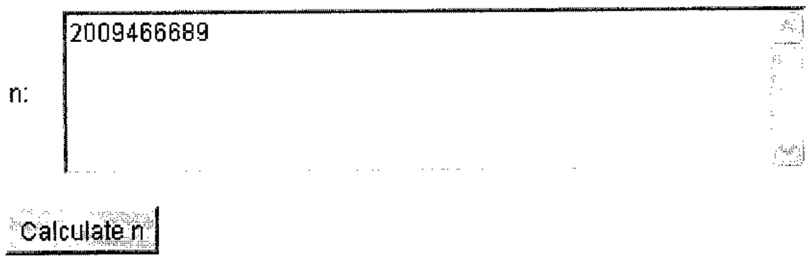
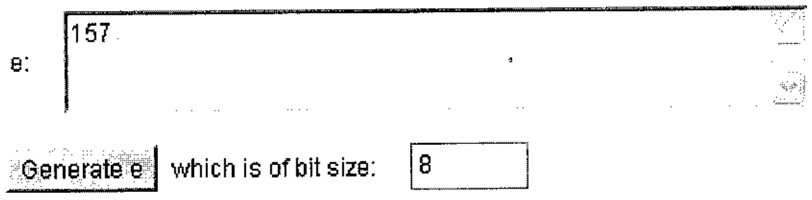

n: 2009466689

Figure 4.1.2: Interface 2; Generate **'n'**

Now click on **"Generate"** to come up with a **'e'** value that is whatever size you want (32 bits usually okay). And finally click on **"Calculate d"** to get **'d'** value.


e: 157

which is of bit size: 8

Figure 4.1.3: Interface 3; Generate **'e'**

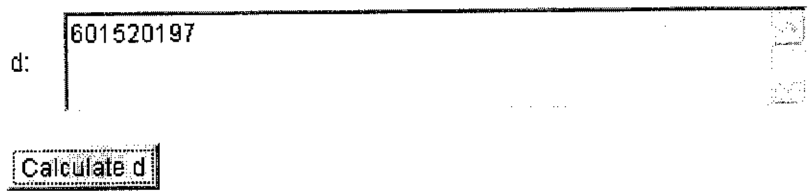

d: 601520197

Figure 4.1.4: Interface 4; Calculate **'d'**

Now, to use these values to do public key cryptography make '**e**' and '**n**' values available to anyone that wants to send an encrypted message. Keep a copy of the '**n**' value and keep the '**d**' value secret.

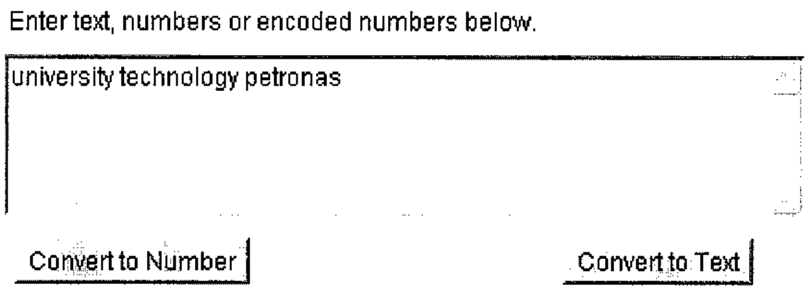
4.1.2 Encryption

To encrypt a message m, simply perform a modular exponentiation to give the ciphertext **c** thus;

$$\mathbf{c = m^e \bmod n}$$

Notice that **m** must be less than **n.** also notice that this procedure is pretty useless if $\mathbf{m^e}$ turns out to be less than **n.** For that reason it is usual to ensure that **m** contain sufficient padding to ensure that $\mathbf{m^e}$ greater than **n.** Choose padding some of which is fixed and some random.

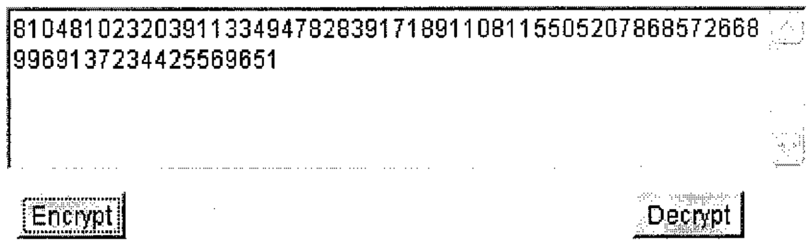
In the project implementation, using the interface, with the '**n**' and '**e**' values entered, type a text message in the "**Plain text message**" area.


Enter text, numbers or encoded numbers below.

university technology petronas

Figure 4.1.5: Interface 5; enter a message to be encrypted.

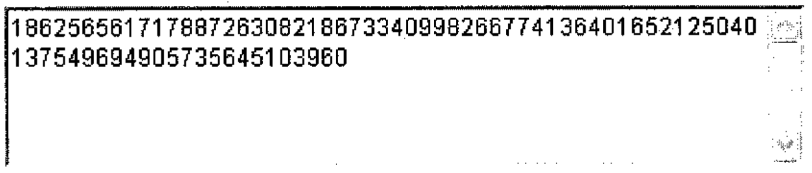
Since RSA only encrypts numbers, the converting of the message is to number with the **"Convert to number"** button. Encryption is done with buttons on the left side and flows downward; decryption is done with buttons on the right and flows upward. The last step is encrypting the message by clicking on the **"Encrypt"** button.



```
81048102320391133494782839171891108115505207868572668
99691372344255696 51
```

Figure 4.1.6: Interface 6; a message being encrypted, convert to bytes using ASCII codes.

The encrypted message is left in numerical form because if it was converted back to text (ASCII), some characters would not show up and could not be copied properly.



```
18625656171788726308218673340998266774136401652125040
137549694905735645103960
```

Figure 4.1.7: Interface 7; an encrypted message in numerical form.

4.1.3 Decryption

The plaintext **m** is recovered from the ciphertext **c** by using d instead of **e** in the modular exponentiation;

$$\mathbf{m = c^{d} \bmod n}$$

Note that if padding is used as recommended for encryption, this can be checked to determine whether the decryption has work properly.

Anyone with the public key can encrypt, but only the owner of the private key can decrypt.

Using the interface, after making sure the proper **'d'** and **'n'** values are entered, place the encrypted numeric message into the bottom text box labeled **"Encoded numerical message."** Use the **"Decrypt"** button to decode the message, and then click on the **"Convert to text"** button.

4.2 The implementation of the codes.

As what have been mentioned earlier, the implementation of the codes for Encryption & Decryption using Java project has been divided into three tasks; key generation, encryption, and decryption. For the first task is where all the buttons and boxes being labeled or can be called as declaration part. For the second task, is the part of encryption code's implementation. Using RSA algorithm, encrypted text will be converted to numbers. And for the third task; is where the implementation of decryption's process. It decrypts back the numbers to the text and can be read by the recipient. Below are the codes for the system.

4.2.1 Key generation.

Key generation is the process of generating keys for cryptography. A key is used to encrypt and decrypt whatever data is being encrypted and decrypted.

For the codes implementation, each labels, buttons, and boxes are being labeled in the same order as they appear on the interface

Below is the code to implement or generate the prime numbers. This is the certainty that the bigInteger class will generate a prime number. It's currently set at 20 which means the odds of being a prime is 1-2^20 or about one in a million of it being non-prime. If this number is increased the time to generate a prime goes up.

```
int prime_certainty = 20;


public void actionPerformed(ActionEvent event) {

        if (event.getSource() == generate_pqButton) {

        int pq_size = new Integer(generate_pq_sizeField.getText()).intValue();


        if (pq_size >= 4) {

        qTextArea.setText(new BigInteger(pq_size + 1, prime_certainty, new        Random()).toString());

        pTextArea.setText(new BigInteger(pq_size - 1, prime_certainty, new        Random()).toString());

                }

                else {

                        pTextArea.setText("Enter larger p and q size.");

                }
                }
```

If the "generate pq" button is pushed then get the desired size of **'p'** and **'q'** then let the BigInteger class generate the prime. The size of **'p'** and **'q'** is offset so that we can guarantee that **'p'** and **'q'** will not be too close to each other. This makes guessing **'p'** and **'q'** by searching values next to the square root of **'n'** more difficult.

And next, is to send the data from **'p'** and **'q'** to calculate **'n'** function.

```
                else if (event.getSource() == calculate_nButton) {

                 nTextArea.setText(calculate_n(newBigInteger(pTextArea.getText()),

                 new BigInteger(qTextArea.getText())).toString());

                }
```

Then, send the data from **'p'**, **'q'** and the size of **'e'** to the generate_e function;

```
                else if (event.getSource() == generate_eButton) {

                          eTextArea.setText(generate_e(new
                BigInteger(pTextArea.getText()),

                          new BigInteger(qTextArea.getText()),

                          new Integer(generate_e_sizeField.getText()).intValue())

                          .toString());


                          if (new BigInteger(eTextArea.getText()).compareTo(new
                BigInteger("0")) == 0) {

                                    eTextArea.setText ("Error, no valid e could be found.                    Try a
            different n value or e size");

                          }
            if (new BigInteger(eTextArea.getText()).compareTo(new
BigInteger(nTextArea.getText())) >= 0) {

                        eTextArea.setText ("Error, e must be less than n.  Try a
smaller e, or larger n size");

            }

}
```

Next, send data from **'p'**, **'q'** and **'e'** to the calculate_d function

```
else if (event.getSource() == calculate_dButton) {

            dTextArea.setText(

                        calculate_d(new BigInteger(pTextArea.getText()),

                                    new BigInteger(qTextArea.getText()),

                                    new BigInteger(eTextArea.getText()))

                                    .toString());

}
```

This converts the plain text string into a number by reading the string in, converting to bytes (ascii), then converting these bytes into a BigInteger. The opposite happens in the next function.

```
else if (event.getSource() == convert_to_numberButton) {

            mTextArea.setText(new BigInteger(plainTextArea.getText().getBytes()).toString());

}


else if (event.getSource() == convert_to_textButton) {

            plainTextArea.setText(new String(new BigInteger(mTextArea.getText()).toByteArray()));
```

```
        }
```

And next is to send the data to be encrypted and decrypted.

```
        else if (event.getSource() == encryptButton) {

                cTextArea.setText(encrypt(new
        BigInteger(mTextArea.getText()),

                new BigInteger(eTextArea.getText()),

                new BigInteger(nTextArea.getText())).toString());

        }


        else if (event.getSource() == decryptButton) {

                mTextArea.setText(decrypt(new
        BigInteger(cTextArea.getText()),

                new BigInteger(dTextArea.getText()),

                new BigInteger(nTextArea.getText())).toString());

        }

    }
```

To generate **'e'** first phi(pq) (which is equal to phi(n)) is calculated. This is equal to (p-1)*(q-1). Then the loop searches for pseudo-randomly generated **'e'** of a specified size until one is found that is relatively prime to phi(pq) (gdc(e,phi_pq) = 1). The generate random prime function was used, because it guarantees a specific bit size of the number it returns. The regular method of generating a pseudo-random number only guarantees the number is between 0 and 2^n-1. Another way would be just adding a one to the front of a random number, but it is set the primarily certainty to 0 because **'e'** inconsiderable as prime. The last line is to ensure it does not go into an infinite loop if **'e'** cannot be found for that bit size.

```
BigInteger generate_e(BigInteger p, BigInteger q, int bitsize) {

        BigInteger e, phi_pq;


        e = new BigInteger("0");

        phi_pq = q.subtract(new BigInteger("1"));

        phi_pq = phi_pq.multiply(p.subtract(new BigInteger("1")));


        int i = 0;


        do {

                e = (new BigInteger(bitsize, 0, new Random())).setBit(0);

                i = i + 1;

        } while( i<100 && (e.gcd(phi_pq).compareTo(new BigInteger("1")) != 0));
```

If no valid **'e'** is found return an error (originally return an invalid e).

```
        if (e.gcd(phi_pq).compareTo(new BigInteger("1")) != 0) {

                e = new BigInteger("0");

        }


        return e;

}
```

Once again the BigInteger class saves the programmer a lot of work. Calculate **phi_pq ((p-1)*(q-1))** and then let the modInverse function do the hard part of finding **e^(-1) mod phi_pq.**

```
BigInteger calculate_d(BigInteger p, BigInteger q, BigInteger e) {

          BigInteger d, phi_pq;


          phi_pq = q.subtract(new BigInteger("1"));

          phi_pq = phi_pq.multiply(p.subtract(new BigInteger("1")));


          d = e.modInverse(phi_pq);

          return d;

}
```

Returns **n=p*q**

```
BigInteger calculate_n(BigInteger p, BigInteger q) {

          return p.multiply(q);

}
```

4.1.2 Encryption

Encryption is done using the modPow function provide by the BigInt class.

```
BigInteger encrypt(BigInteger m, BigInteger e, BigInteger n) {

          BigInteger c, bitmask;

          c = new BigInteger("0");

          int i = 0;

          bitmask = (new BigInteger("2")).pow(n.bitLength()-1).subtract(new BigInteger("1"));
```

```
        while (m.compareTo(bitmask) == 1) {

                c = m.and(bitmask).modPow(e,n).shiftLeft(i*n.bitLength()).or(c);

                m = m.shiftRight(n.bitLength()-1);

                i = i+1;

        }

        c = m.modPow(e,n).shiftLeft(i*n.bitLength()).or(c);

        return c;

}
```

4.2.3 Decryption

Decryption is done just as encryption above, only now the data is read in chunks the same size as **'n'**, and the result, if correct, will be one bit less than the size of **'n'** (because that was the original chuck size).

```
BigInteger decrypt(BigInteger c, BigInteger d, BigInteger n) {

        BigInteger m, bitmask;

        m = new BigInteger("0");

        int i = 0;

        bitmask = (new BigInteger("2")).pow(n.bitLength()).subtract(new BigInteger("1"));

        while (c.compareTo(bitmask) == 1) {

                m = c.and(bitmask).modPow(d,n).shiftLeft(i*(n.bitLength()-1)).or(m);

                c = c.shiftRight(n.bitLength());

                i = i+1;

        }

        m = c.modPow(d,n).shiftLeft(i*(n.bitLength()-1)).or(m);

        return m;

}
```

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Encryption can play a very important role in day-to-day computing and communicating. With the implementation RSA Encryption & Decryption using Java's project, it helps user to protect their privacy of data from being seen by others. It simple and easy to be used, even the process of encryption/decryption of RSA looks complicated to be understood at the early stage. However, encryption gives many advantages as stated below;

- Encryption can protect information stored on your computer from unauthorized access - even from people who otherwise have access to your computer system.
- Encryption can protect information while it is in transit from one computer system to another.
- Encryption can be used to deter and detect accidental or intentional alterations in your data.
- Encryption can be used to verify whether or not the author of a document is really who you think it is.

Despite these advantages, this project has some weaknesses. Encryption provides protection to your data but encryption also has its limits:

- Encryption can't prevent an attacker from deleting your data altogether.
- An attacker can compromise the encryption program itself. The attacker might modify the program to use a key different from the one you provide, or might record all of the encryption keys in a special file for later retrieval.

- An attacker might find a previously unknown and relatively easy way to decode messages encrypted with the algorithm you are using.
- An attacker could access your file before it is encrypted or after it is decrypted.

For all these reasons, encryption should be viewed as a part of our overall computer security strategy, but not as a substitute for other measures such as proper access controls.

5.2 Future recommendation

5.2.1 Implement the algorithm within the grid computing.

Grid computing is gaining a lot of attention within the IT industry. Although it has been used within the academic and scientific community for some time, standards, enabling technologies, toolkits, and products are becoming available that allow businesses to use and reap the advantages of Grid computing. As with many emerging technologies, you will find almost as many definitions of Grid computing as people you ask.

Because a grid may be large, dispersed, and heterogeneous, designing a grid application can present a challenge. While a non-grid application runs in a relatively stable, well-defined, and often dedicated environment, a grid-enabled application runs in a dynamic, sometimes loosely defined, and heavily networked environment

The grid application needs to handle any and all requirements for authentication, access control, data integrity, confidentiality of data, and public and private key management. The Globus Toolkit provides a Certificate Authority (CA) to use in establishing the identity of each member of a grid. The Public Key Infrastructure (PKI) and the Grid Security Infrastructure (GSI) are helpful. In addition, the grid application must handle both symmetric and asymmetric encryption schemes.

5.2.2 The combination of symmetric and asymmetric key

If we want the benefits of both types of encryption algorithms, the general idea is to create a random symmetric key to encrypt the data, and then encrypt that key asymmetrically. Once the key is asymmetrically encrypted, we add it to the encrypted message. The receiver gets the key, decrypts it with their private key, and uses it to decrypt the message.

Cryptography is a very robust field. In the current state of cryptography, the keys are the most important tools in keeping data secure. Keeping the private keys secure and large enough will make it very difficult to crack an encryption system.

REFERENCES & APPENDICES

References :

1) Jonathan Eisenzopf. **RSA Encryption in Perl:** ***Encryption overview.*** http://www.webreference.com/perl/tutorial/16/. **November 9, 2000.**
2) http://en.wikipedia.org/wiki/Phi
3) Agus Setiawan, David Adiutama, Julius Liman, Akshay Luther and Rajkumar Buyya. ***GridCrypt: High Performance Symmetric Key Cryptography using Enterprise Grids.*** http://www.gridbus.org/papers/gridcrypt.pdf. 2005
4) Adam. **A** ***free pass code encryption application for Windows Mobile Smartphone.*** http://msmobiles.com/news.php/4013.html. July 08, 2005
5) PGP Corporation. ***PGP Whole Disk Encryption.*** http://www.pgp.com/products/wholediskencryption/index.html. 2002-2006
6) Mao, W. ***Modern Cryptography: Theory & Practice.*** Upper Saddle River (NJ): Prentice Hall Professional Technical Reference, 2004.
7) Schneier, B. ***Applied Cryptography***, 2nd Ed. New York: John Wiley & Sons, 1996.
8) Denning, D.E. ***Cryptography and Data Security***. Reading (MA): Addison-Wesley, 1982.
9) Chey Cobb, ***Cryptography for Dummies***, CISSP: Wiley Publishing, Inc., 2004.
10) Kenneth W. Dam and Herbert S. Lin: ***Cryptography's Role In Securing the Information Society:*** National Research Council, 1996.
11) Gary C. Kessler: ***An Overview of Cryptography***: *Handbook on Local Area Networks*: Auerbach, 1998.
12) Encryption Issues: www.thecomputershow.com

13) **Combining Symmetric and Asymmetric Encryption:** http://www.codeproject.com/dotnet

14) Nadia Nedjah, Luiza de Macedo Mourelle; ***Efficient and secure cryptographic systems based on addition chains:*** Hardware design vs. software/hardware co-design; Dept. of Electronic Engineering and Telecommunications, State Uni. Of Rio de Janeiro, Brazil.

15) Peter Gutmann. ***Encryption and Security Tutorial.*** University of Auckland. http://www.cs.auckland.ac.nz/~pgut001.

16) Paul Kocher, President & Chief Scient. ***How to Think Like a Cryptographer.*** Cryptography Research, Inc. RSA 2004 – February 24, 2004

17) Sun Microsystems, Inc. ***Forte™ for Java™ 4, Community Edition: Getting Started Guide.*** http://www.sun.com/patents. Copyright © 2002

18) Whitfield Diffie and Martin E. Hellman. ***New Directions in Cryptography.*** www-ee.stanford.edu/~hellman/publications/24.pdf

Other related web resources:

Free Encryption / Cryptographic Software	http://www.thefreecountry.com/
Protected your document	http://www.codecomments.com/
Encryption and Decryption using PHP and GnuPG	http://www.zend.com
PGP Encryption for Beginners	http://library.linux360.ro/tutorials
Combining Symmetric and Asymmetric Encryption	http://www.codeproject.com
Encryption using the Win32 Crypto API	http://www.codeproject.com
Cryptography Tutorial, Implementation and Starter Kit	http://www.cryptography-tutorial.com/
Crypto tutorial	http://www.antilles.k12.vi.us
DES Encryption Package	http://efgh.com/software/des.htm
RSA Public-Key Cryptography	http://efgh.com/software/rsa.htm
Advanced Encryption Package 2006 Professional (Cryptography)	http://www.secureaction.com/
RSA & DES Demonstration Programs.	http://www.privacycrypt.com/
RSA demo	http://euler.slu.edu/
New Directions in Cryptography	http://www.cs.rutgers.edu
RSA Algorithm	http://www.di-mgt.com.au
Computers and Society Applications of Encryption	http://www.cs.usfca.edu
RSA Security Releases RSA Encryption Algorithm into Public Domain	http://www.hipaadvisory.com/

APPENDICES

Appendix A: RSA example

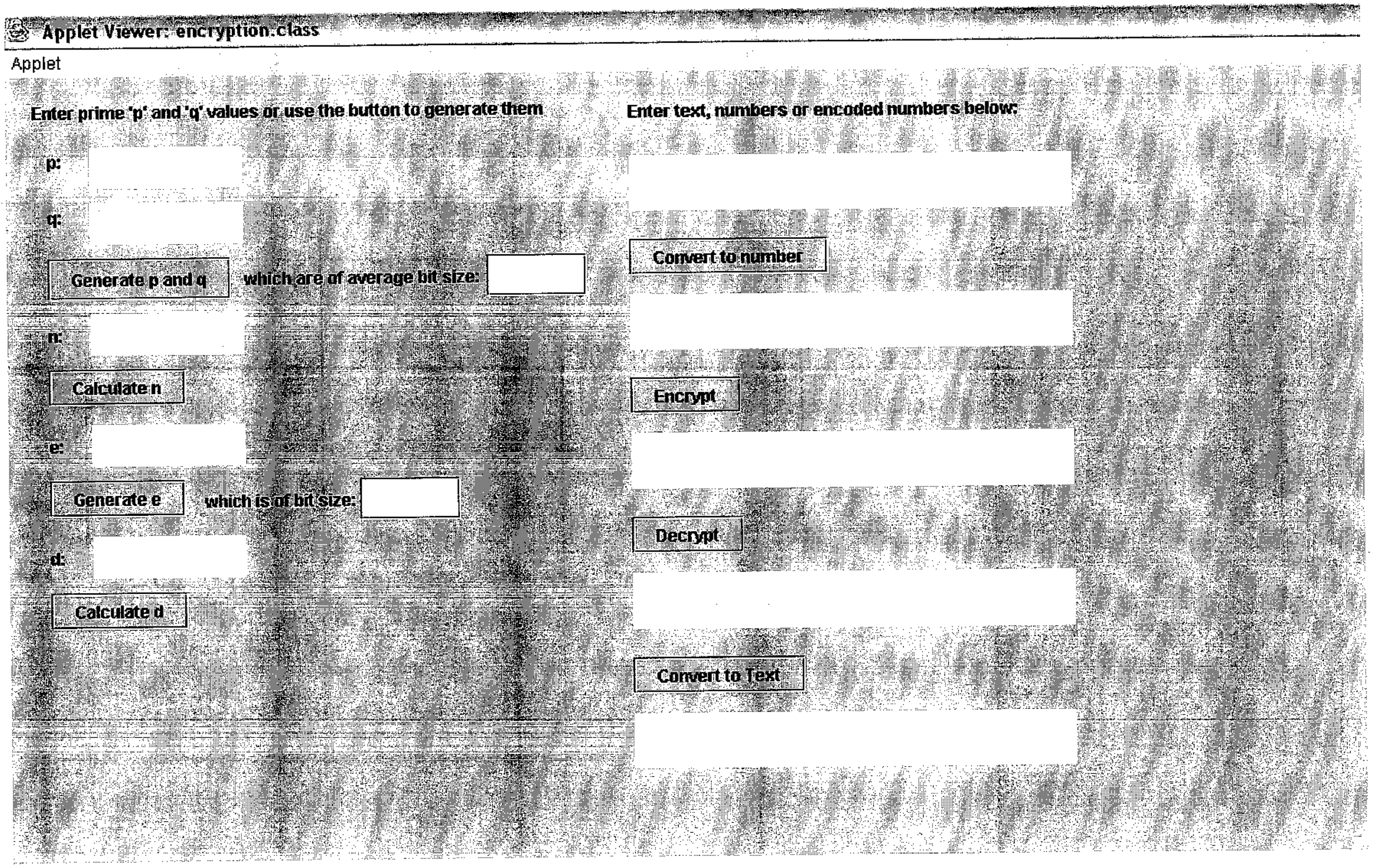
Figure 4- 1 RSA example

Step: Preparation	a) Choose two primes **p** and **q** so that their product **n=p*q** is greater than the used alphabet length M (i.e. here M=26). b) Compute φ(n).	a) Say **p=3** and **q=11**, then **n=33** b) **φ(33)** = (3-1)*(11-1) = **20**

2. Step: Encryption uses the public key (n,e)	a) Choose a public encoding key **e** that has to be relative prime to **φ(n)**.
	
	b) Now encrypt each plain letter **P** by computing
	
	C=P^{e} MOD **n**.
		a) Here, possible values for **e** are 3, 7, 9, 11, 13, 17, 19. Let's pick **e=3**.
		b) Encrypt as follows:
		
		S =18: 18^3 = **24** MOD 33
		A = 0: 0^3 = **0** MOD 33
		F = 5: 5^3 = **26** MOD 33
		E = 4: 4^3 = **31** MOD 33

3. Step: Decryption uses the private key (d,n)	a) The private decoding key **d** is chosen as the inverse of **e** MOD φ(n): **e** * **d** = 1 MOD **φ(n)** Mathematically, find integers d and k that fulfill: **e** * **d** = 1 + k * **φ(n)** via the Extended Euclidean Algorithm. b) Decrypt by computing **P=C^{d}** MOD **n**	a) **d**=7 since 3*7 = 1 MOD 20. b) **24**^{7} = 18 MOD 33, **18=S.** **0**^{7} = 0 MOD 33, **0=A.** **26**^{7} = 5 MOD 33, **5=F.** **31**^{7} = 4 MOD 33, **4=E.**

Appendix B: The full interface of 'RSA Encryption & Decryption using Java' system.



Enter prime 'p' and 'q' values or use the button to generate them

p:

q:

which are of average bit size:

n:

e:

which is of bit size:

d:

Enter text, numbers or encoded numbers below: