

## St. Cloud State University theRepository at St. Cloud State

---

Culminating Projects in Information Assurance

Department of Information Systems

---

12-2016

# Design of a Data Encryption Test-Bed Used to Analyze Encryption Processing Overhead

Swarna Rekha Manchikatla

*St. Cloud State University*, [swarnarekha6236@gmail.com](mailto:swarnarekha6236@gmail.com)

Follow this and additional works at: [https://repository.stcloudstate.edu/msia\\_etds](https://repository.stcloudstate.edu/msia_etds)

---

### Recommended Citation

Manchikatla, Swarna Rekha, "Design of a Data Encryption Test-Bed Used to Analyze Encryption Processing Overhead" (2016).  
*Culminating Projects in Information Assurance*. 12.  
[https://repository.stcloudstate.edu/msia\\_etds/12](https://repository.stcloudstate.edu/msia_etds/12)

This Starred Paper is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact [rswexelbaum@stcloudstate.edu](mailto:rswexelbaum@stcloudstate.edu).

**Design of a Data Encryption Test-Bed Used to Analyze Encryption Processing Overhead**

by

Swarna Rekha Manchikatla

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

October, 2016

Starred Paper Committee:  
Dr. Dennis Guster, Chairperson  
Dr. Susantha Herath  
Dr. Balasubramanian Kasi

## Abstract

Data security is one of the most pressing issues faced by the organizations today. Unauthorized access to confidential information corresponding to employees/customers like SSN (Social Security numbers), financial information, health records, birth dates can be compromised both to the individual customers involved and the company withholding the data. The problem has become immense, approximately 260 million records were compromised since 2005 and companies, states and countries have reacted by mandating that industries should stringently follow the best security practices, including encryption and decryption of data. Also, the costs associated with data threats are quite increasing (Whitfield & Susan, 2007). Businesses that use strong encryption methodologies in their mobile devices, computers, cloud systems, other locations might not gain 100 % protection from dangerous hackers, but they can decrease their vulnerability to such attacks and thereby the potential of financial losses. Data encryption is the method of converting data in a computer or any communication system making it unintelligible in a way that the data can be reversed only by the authorized people accessing the original data. The primary goal is to safeguard the confidentiality of data, but integrity checks are also provided by the technique in various forms of authentication message codes. For instance, digital signature schemes are also fundamentals of encryption. The purpose of it is to ensure the authenticity of the identity of the receiver and sender. With an increasing awareness of security threats, many of the current companies are using cryptographic techniques for ensuring data security. Many of the companies like Amazon, Apple, AT&T and Comcast are using encryption techniques for securing the information. While there are a many encryption and decryption techniques available today, there is an obvious requirement for the current companies to find and choose the best reliable cryptographic techniques for securing their data. A performance test of various algorithms is needed to bring up the best technique. This research paper deals with the implementation of different cryptographic algorithms with a programming language called JAVA. It involves designing a graphical user interface (GUI) where sample input can be entered, common algorithms used to encrypt and decrypt the input can be selected. A mechanism for building a test bed for comparing the performances of the implemented algorithms is designed to calculate the encryption processing overhead.

## Table of Contents

	Page
List of Tables .....	6
List of Figures .....	7
Chapter	
I. Introduction.....	9
Introduction.....	9
Problem Statement .....	10
Significance of the Study .....	10
Objective of the Study .....	10
Acronyms .....	11
Summary .....	12
II. Background and Review of Literature .....	13
Introduction.....	13
Encryption.....	17
Decryption.....	17
Encryption Techniques .....	17
Importance of Cryptosystem.....	19
Encryption Implementation in Different Areas .....	20
Companies Encrypting Data .....	23
Summary .....	25
III. Methodology.....	26

	4
Introduction.....	26
JAVA Programming Language.....	26
Advantages of JAVA .....	26
How Does JAVA Work? .....	27
Swing in JAVA.....	29
Integrated Development Environment (IDE).....	31
Purpose of Cryptography .....	33
Cryptographic Algorithms .....	34
Other Encryption Algorithms Under Study .....	51
Design of a Test-Bed for Encryption Processing Overhead .....	51
System Design .....	52
Interface Design .....	58
System Testing.....	59
Test Results.....	63
Summary.....	63
IV. Analysis of Results .....	64
Introduction.....	64
Running the Data Encryption Test Bed Project.....	64
Hardware and Software Requirements .....	72
Summary.....	73
V. Conclusion and Future Study.....	74
Introduction.....	74

	5
Conclusion .....	74
Future Work .....	75
References.....	76
Appendix.....	82

**List of Tables**

Table	Page
1. Acronyms .....	11
2. AES State Structure .....	47
3. AES Rounds Based on Block and Key Length.....	48

## List of Figures

Figure	Page
1. Symmetric Encryption .....	18
2. Asymmetric Encryption .....	19
3. Top Companies Security Policies .....	24
4. Classification of Encryption Methods .....	25
5. Byte Code Conversion of JAVA Virtual Machine .....	29
6. JAVA Swing Class Hierarchy .....	31
7. Data Encryption Standard Functionality.....	36
8. DES Key Schedule.....	37
9. Feistel Scheme of DES .....	38
10. Functionality of Triple DES Algorithm.....	39
11. Public Key Cryptography .....	40
12. Steps of AES Encryption .....	43
13. AES Structure .....	46
14. Cipher Block Chaining (CBS) Mode Encryption and Decryption .....	49
15. Generation of Message Digest Using MD5 .....	50
16. MD5 File Transfer .....	50
17. Use Case Diagram for Design of Data Encryption Test-Bed .....	54
18. Class Diagram for Design of Data Encryption Test-Bed.....	55
19. Sequence Diagram for Design of a Data Encryption Test-Bed .....	56
20. Activity Diagram for Design of Data Encryption Test-Bed .....	57



	8
21. Collaboration Diagram for Design of Data Encryption Test-Bed .....	58
22. Running Project for Main Window.....	65
23. Main Window for Data Encryption Test-Bed Project .....	66
24. Selection Algorithm for Data Encryption and Decryption .....	67
25. Entering Plain Text to Encrypt and Decrypt.....	68
26. Displaying the Best Algorithm for the Given Plain Text .....	69
27. Encrypted and Decrypted Text .....	70
28. Encryption and Decryption Details.....	71
29. Graphical Chart Representation of Encryption/Decryption Algorithms .....	72

## Chapter I: Introduction

### Introduction

Security breaches have been intensively seen across multiple financial/commercial organizations over decades. About 42.8 million security incidents occurred in 2014, which is a 48% increase since 2013. These are just the incidents that the companies were originally aware of. Every year, organizations are putting millions and billions of dollars into data security and, every time, hackers are finding new and innovative ways to access sensitive data which has become a serious problem in the businesses today. Many businesses have looked to encryption to solve their security risks. Strong encryption is an imperative part of any business security plan. Encryption is a method used to secure data such that it can be read only by the corresponding intended recipient. Encryption uses a set of instructions called a cipher or algorithm to modify a message or any other piece of data into another form that don't at all resemble the original text. Encryption is used to ensure both confidentiality and integrity. There is an obvious need of a tool that investigates the best cryptographic algorithms amongst the available ones for the businesses or organizations involving huge clients (National Academy of Public Administration, 2015).

This paper gives a high-level overview of popular encryption and decryption algorithms. The primary goal of the paper is to develop a test-bed that compares the performance overhead of various cryptographic algorithms implemented using JAVA programming language that decides the best reliable algorithm based on the time taken to encrypt and decrypt the input message. This tool allows the organizations to choose the best algorithm for encryption of their data without requiring any additional budget for research or analysis.

**Problem Statement**

Many organizations today are facing difficulties in choosing the best encryption and decryption algorithms for incorporating data security, confidentiality, and integrity in their businesses. Organizations need huge budget for analyzing the encryption processing overhead of a variety of cryptographic algorithms that are available in the market today.

**Significance of the Study**

The proposed system develops a Graphical User Interface (GUI) that allows a user to analyze the performance of different encryption and decryption algorithms. The user can run the project for different input text messages and compare the time taken for encryption and decryption of the given text against different encryption and decryption algorithms. For the entered text and selected algorithm, the project displays the best algorithm from amongst the available algorithms based on encryption and decryption times. The user also has the provision to view the performance reports of executed encryption and decryption algorithms in a pictorial representation.

**Objective of the Study**

The objective of the study is to design a data encryption test-bed that analyzes the encryption processing overhead of different encryption algorithms and displays the best reliable algorithm from amongst the available ones. This tool also gives a graphical representation of the performance of different algorithms.

## Acronyms

Table 1

### *Acronyms Used in This Document*

<b>Acronym</b>	
<b>DES</b>	Data Encryption Standard
<b>AES</b>	Advanced Encryption Standard
<b>CBC</b>	Cipher Block Chaining
<b>SHA</b>	Secure Hash Algorithm
<b>MD5</b>	Message Digest Algorithm
<b>RC6</b>	Rivest cipher 6
<b>RSA</b>	Rivest-Shamir-Adleman
<b>IDE</b>	Integrated Development Environment
<b>JDK</b>	Java Development Tool Kit
<b>JRE</b>	Java Runtime Environment
<b>GUI</b>	Graphical User Interface
<b>HTTP</b>	Hyper Text Transport Protocol
<b>HSTS</b>	HTTP Strict Transport Security
<b>MVC</b>	Model View Controller
<b>AWT</b>	Abstract Window Toolkit
<b>SVN</b>	Subversion
<b>PKC</b>	Public Key Cryptography
<b>CPU</b>	Central Processing Unit
<b>ASCII</b>	American Standard Code for International Interexchange
<b>ECB</b>	Electronic Code Book
<b>NIST</b>	The National Institute of Standards and Technology

**Summary**

The problems faced by the organizations in the current arena are briefly described in this chapter. The significance and objective of this research paper is also discussed. The encryption/decryption process, importance/implementation of cryptosystem in the real world is described and the studies conducted by different authors in the past are illustrated in Chapter II.

## Chapter II: Background and Review of Literature

### Introduction

The modern computers are *associated with standard encryption* in one way or another. There is no coincidence when the first computers that is, semi-programmable computer called Colossus, was developed to decrypt text at the time of Second World War (Kopplin, 2002).

Arora et al. (cited in Priyanka, Arun, & Himanshu, 2012) investigated the performance of various security algorithms on single and multiple processors for various sizes of input. Arora et al. also researched about the quantitative entities such as Speedup Ratio that helps in evaluating the performance of various security algorithms like RSA, MD5 and AES that are used in the current world for encrypting/decrypting huge quantity of data. She considered three cryptographic algorithms, namely MD5 (a hashing technique), RSA (an asymmetric cryptographic algorithm), and AES (a symmetric cryptographic algorithm).

Seth et al. (cited in Shashi, & Rajan, 2011) did a depth analysis on three algorithms—DES, RSA, and AE—by considering few parameters; for example, computational time, output byte, and memory utilization. These factors form the significant issue in any Encryption Algorithm. Test results prove that the DES algorithm takes minimum encryption time, whereas AES algorithm takes minimum memory utilization but the encryption time difference in AES and DES algorithms is extremely minor. On the other hand, RSA algorithms take the longest encryption time and memory utilization but output byte is minimized.

Abdul, Kader, and Mohie (2008) made an analysis on the performance of Symmetric Encryption and Decryption Algorithms. Their research paper gave an assessment of six of the most widely recognized encryption algorithms: RC2, AES (Rijndael), Blowfish, DES, 3DES,

and RC6. A comparison at different settings was conducted for each algorithm. For example, various data block sizes, various data types, utilization of battery power, diverse key size, and speed of encryption or decryption. Experiments show following test results. There is no huge difference when the test results are shown either in hexadecimal base encoding or in encoding of base 64. In the scenario of the changing size of packet, it was discovered formerly that RC6 need less time when compared to all other algorithms except Blowfish. If there should be an occurrence of changing data type, for example, image rather than text, it was found that RC6, RC2, and Blowfish has the drawback over different other algorithms as far as time utilization is concerned. Likewise, 3DES has very low performance when compared to DES algorithm. Because of changing key size (just in RC6 and AES algorithms), it can be concluded that the higher key size prompts a clear change in the time and battery utilization.

Pavithra and Ramadevi (2012) analyzed the performance assessment of different cryptographic algorithms. Taking time as a parameter, different cryptographic algorithms are assessed on various video files. Various video files are having distinctive processing speed on which files of different sizes are processed. Time is calculated for encryption and decryption in various video file formats; for example, .vob and .DAT, having document size range from 1MB to 1100MB. Test experiments show that AES algorithm is executed in minimum processing time and high throughput level when compared with BLOWFISH and DES. Alanazi et al. (2010) did a comparison experiment of three Encryption Algorithms (AES, DES, and 3DES) using nine components, for example, Block Size, Key Lengths, Security, Cipher Type, Possible ASCII printable character keys, Possible Keys, and Time needed to verify all feasible keys at 50 billion keys every second and so on. Results demonstrated that AES is better when compared to DES

and 3DES. Mandal et al. (cited in Akash, Chandra, & Archana, 2012) worked on comparing the two most generally used symmetric encryption methods, namely Advanced encryption standard (AES) and Data encryption standard (DES) based on Avalanche effect because of a bit variation in input plaintext making the key and plaintext constant, requirement of memory and simulation time for implementation and encryption respectively. Avalanche effect is a feature of any encryption algorithm where a piece of variance either in the key or the input plaintext results in a noticeable change in the output cipher text.

Avalanche Effect = Number of flipped bits in the output cipher text/Number of bits in the output cipher text.

Avalanche effect is huge in AES when compared to DES while simulation time and memory requirement for DES is more than that of AES, which demonstrates AES is better when compared to DES. AES is perfect for encrypting messages sent between objects by means of chat channels, and is helpful for objects involving monetary transactions. Kakkar et al. (cited in Ajay, Singh, & Bansal, 2012) researched different procedures and algorithms utilized for the security of data in MN (Multinode Network). They found that the system strength relies on the management of key, number of keys, type of cryptographic key (private or public keys), number of bits utilized as a part of a key. Longer key length and information length expends more power and results in more warmth dispersal. Long key and data lengths require more power and leads to more spread of heat. All the keys rely on the mathematical properties and in accordance with time, their strength reduces. The keys having more number of bits require more processing time, which basically demonstrates that the system need more time for data encryption.



Data Encryption Algorithms were compared by Raman and Simar (2011). The experimental results demonstrated that Blowfish has performed better when compared to other common encryption algorithms. AES showed bad performance in the experiment when compared to other encryption algorithms because of the need of additional processing power. The primary analysis is made using ECB Mode. The outcomes demonstrated that the Blowfish algorithm takes less processing time on comparing with other algorithms. The results also showed that AES need additional resources as the block size of data increases. Another point can be seen here that 3DES requires constantly more time than DES because of its triple stage encryption method. Blowfish, that has a long key (448 bits), dominates other encryption algorithms. DES and 3DES are known to have worm gaps in their security method, AES and Blowfish did not have any such things noticed as of today (Simar & Raman, 2011).

Turki Al-Somani et al. (cited in Thambiraja, Ramesh, & Umarani, 2012) conducted Performance Evaluation of three Encryption / Decryption Algorithms on different Operating systems like SunOS and Linux. They implemented three symmetric block encryption and decryption algorithms with the help of Java and JCA. The principle goal was to assess the performance of these algorithms regarding CPU execution time. The estimations were performed on SunOS and Linux platforms. The CPU execution time for producing the encryption, decryption and secret key on a 10MB file is analyzed. The outcomes demonstrated that the Blowfish algorithm was the fast algorithm, which is followed by the DES algorithm and Triple-DES algorithm. The 3DES algorithm was moderate in its performance because of the additional complexity and security that it has on top of DES algorithm (Thambiraja et al., 2012).

## **Encryption**

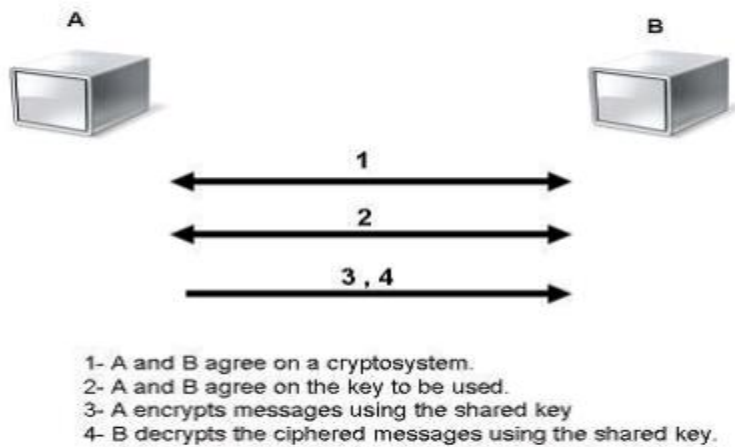
Encryption is a method of encoding information in such a way that it makes difficult to decode the information. The longer the length of the cipher/key (in bits), the more difficult it would be to break.

## **Decryption**

Decryption is a methodology of changing encrypted data back into its original form. This process requires a public or private key.

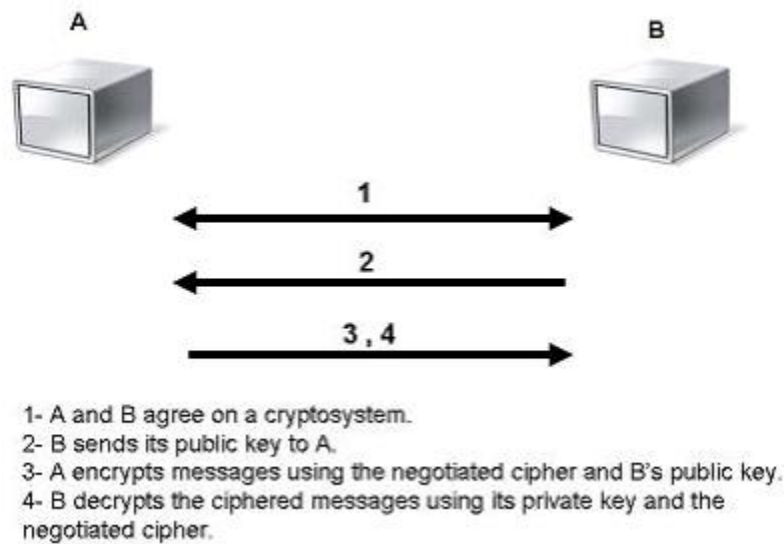
## **Encryption Techniques**

**Symmetric encryption.** Symmetric encryption uses only a single cipher/key for data encryption and decryption. In this scenario, the Sender And receiver accept to share a common secret key. Both Sender And receiver encrypt and decrypt their input messages using the same shared key. In Figure 1, the Sender A and Receiver B agree to use a common encryption and decryption technique. The sender encrypts the input text using the shared key and the encryption technique selected. On the other hand, the receiver decrypts the encrypted text using the same shared key and corresponding decryption technique. The key plays a major role in symmetric encrypt as the whole system compromises if the shared key is known by any third party for any reason (Abdel, 2006).



*Figure 1.* Symmetric Encryption (Abdel, 2006).

**Asymmetric encryption.** Asymmetric encryption is also named as public-key encryption. It uses two keys, one to encrypt and the other to decrypt data so that the data is sent more securely. This mechanism is also known as PKC—Public Key Cryptography. Here users use public key that is viewed by public, private key that is viewed only by users. Figure 2 shows that Sender A and Receiver B accept to use a common encryption and decryption technique. Node A encrypts the input message using the public key. Node B decrypts the received input message using its own private key. Managing the secret keys is a tedious task in Asymmetric encryption, but more security is incorporated in this mechanism (Abdel, 2006). Asymmetric encryption runs 1,000 times slower when compared to symmetric because of the complex mathematical processing it undergoes. To take advantage of both benefits, hybrid encryption/decryption technique is used, where exchange of secret keys is taken care by asymmetric encryption and transfer of input message from sender to receiver is taken care by the symmetric encryption (Yogesh & Munjal, 2011).



*Figure 2.* Asymmetric Encryption (Abdel, 2006).

### **Importance of Cryptosystem**

Cryptosystem refers to a suite of cryptographic algorithms that are needed to implement a specific security service, most commonly for achieving confidentiality (encryption). It includes both encryption and decryption of data.

Companies can encrypt their websites using HTTPS—Hypertext Transfer Protocol Secure—by default which means that in cases when a user connects to the website, HTTPS will automatically use a separate channel which encrypts the communications from the dedicated computer to the website. Companies need to flag all authentication cookies as secure which means that cookie communications are restricted only to encrypt transmission that aids web browsers to use the cookies through only an encrypted connection. Also, it stops network operators from stealing/logging user's identities by dealing with authentication cookies passing over insecure connections. To ensure that the communication remains secure, companies can enable HTTP Strict Transport Security (HSTS). It essentially guides on using only secure

communications, thereby preventing few attacks in situations when a network acts that the site has actually asked to insecurely communicate (Korde, 2016).

Email service providers implement STARTTLS for email transfer. STARTTLS is an encryption system that is used to encrypt communications between email servers using the standard—Simple Mail Transfer Protocol (SMTP). Whenever a user emails on another provider, for instance, Hotmail user sending email to a Gmail user, the mail message has to be delivered actually over the Internet. In cases where both the email servers understand STARTTLS, then only the communications will be encrypted. If not, they would be exposed to eavesdropping, hence it is important to many email service providers to implement the system (Lucian, 2016). Companies can use forward secrecy for encryption of keys. Forward secrecy is developed to secure previously encrypted communications, even in case one of the service provider's keys is compromised in the future. Without the forward secrecy, an attacker who has a service provider's secret key can use it instantly for going back and read previous communications and may be the ones that were recorded in the past, some months or more ago.

### **Encryption Implementation in Different Areas**

Encryption will always secure, clean data in addition to allowing it to get transferred to any other parties while being in accordance with many other security measures. Once a file is encrypted, it would be difficult for other outside parties to get in and get access to sensitive/personal or business data. Encryption is available for any device/area where data is stored, which includes internet traffic, USB and external drive, complete hard drives, passwords, and cloud storage.

**Internet traffic.** Travelling with a laptop is a convenient way to get into company files whenever you need them, but, using an unsecured Wi-Fi network in any public place may make you vulnerable to malicious attacks. Using a VPN—virtual private network—users can access third-party server, which in fact, encrypts the information (Drew, 2014).

**USB and external drive.** Portable data storage devices are convenient to use, but they have the potential risk of theft or loss. On the other hand, products like BitLocker help keep removable media encrypted in the scenarios where they might fall into the deceptive hands irrespective of type.

**Complete hard drives.** Usually, a user's login id and password to a PC that corresponds to a specific company will not be helpful if someone steals the complete hard drive. As soon as the drive is plugged into another PC, the thief can get at all of its contents. For computers with operating systems as Enterprise or Vista (or the Enterprise/Pro, Windows 8) or Ultimate, Windows 7 or Microsoft, BitLocker software is provided which comes with complete encryption. Just navigate to Control Panel, System and Security and turn BitLocker Drive Encryption on by selecting the appropriate radio button.

**Passwords.** The most vital component of encryption is a password. For the most hack-proof password, choose a long code of more than 10 characters that includes both upper and lower case letters, special characters, and numbers. Assign each system or device a separate password and store them in a secure place if it is too difficult to remember them.

**Cloud storage.** Services like Dropbox provide built-in data encryption, which provides security when your data is on their servers. Even then, they also have decryption keys that give

them access to your data under certain situations. Products such as TrueCrypt added to cloud storage locations deliver an extra layer of security.

In the early modern encryption, Cryptosystem's standards became important in the mid-1970s. The first was the DES Data Encryption Standard (DES) cryptographic algorithm using a cipher/key with 56 bits long. DES was considered strong enough before that is to be used for securing bank's Automatic Teller Machines (ATMs) but, as the processing power gradually increased, it was then replaced by triple DES that actually ran the same data using the DES algorithm for about three times for more strength (National Institute of Standards and Technology, 1977). A new encryption technique named Advanced Encryption Standard (AES) was developed in 2001. We could encrypt the cipher/key with another one, but still the problem of sending the second cipher/key securely to the receiver remains (Schneier et al., 1998). Physically handing over the key to someone is not practical in a commercial way and, hence sending the keys to others to decrypt data securely becomes quite impossible. Therefore, the second kind of encryption, namely asymmetric/public key encryption, was developed. This encryption uses two keys, one is a private one and the other is a public one. If one key is used to encrypt the data, the other will decrypt it. If Company X wants to send a text to Company Y, it uses Y's public key, which is available to the public, to encrypt the text. After it is encrypted, the only one that can decrypt the text is the Y's private key which is accessible only by B. The original development of this technology formed RSA Security that is still used today in their products and organizations.

Symmetric key encryption type is, in fact, faster than asymmetric, hence, we encrypt data using a symmetric key and again encrypt the key using the famed RSA algorithm," according to

Mike Vegara, Director of RSA product management (Abdul, Kader, Abdul, & Hadhoud, 2001, p. 1). On the other hand, AES has a small cipher/key length of 128 bits, whereas the RSA algorithm begins at 1,024 bits. However, according to Nicko van Someren, Cipher's chief technical officer, the difference is that RSA is very difficult to break. He said that 1,024-bit RSA uses 10 times as many numbers of processor cycles for computation but it takes in the order of about 30,000 times much longer to break (Abdulet al., 2001).

Similar to symmetric encryption, hashing algorithms are also coming in different flavors. One such is MD5 that is still used in many of the today's systems, but has been overtaken by SHA-1, that is developed by the National Security Agency (NSA) during the mid-1990s. Even then, the security of SHA-1 has been doubted by the community of cryptography using some alleged attacks.

### **Companies Encrypting Data**

Four companies, namely, Google, Dropbox, Spider Oak, and Sonic.net are implementing five of the best standards for encryption as illustrated in Figure 3. Also, Yahoo! has just published that it is taking several measures to increase encryption, which includes the very powerful data center encryption of links, Twitter announced that it has started the data center links encryption. By using encryption over networks and computer systems, service providers can make surveillance of backdoor much challenging (Kurt, Nate, & Parker, 2013).













	Encrypts data center links	Supports HTTPS	HTTPS Strict (HSTS)	Forward Secrecy	STARTTLS
	undetermined	limited	✗	undetermined	✗
	undetermined	✓ (iCloud)	✗	undetermined	✗ (me.com, mac.com)
	undetermined	undetermined	✗	undetermined	✗ (att.net)
	undetermined	undetermined	✗	undetermined	✗ (comcast.net)
	✓	✓	✓	✓	✓
	undetermined	✓	planned	✓	✗ (facebook.com)
	undetermined	✓	✓	undetermined	✗
	✓	✓	✓	✓	✓
	✗ contemplating	✓ planned 2014	✓ planned 2014	✓ planned 2014	✗ contemplating
	✗	✓	✗	undetermined	✗ (outlook.com)

Figure 3: Top Companies Security Policies (Kurt et al., 2013).

Shashi and Rajan (2011) did a Data Communication Comparative Analysis of Encryption Algorithms. These researchers evaluated the performance of the encryption and decryption algorithms based on memory usage, output bytes, and computational time for the algorithms illustrated in below figure. The test outcomes demonstrated the correlation of AES, DES, and RSA utilizing same input text file for five experiments. AES and DES produced the same output byte for various file sizes. The authors showed that the RSA has extremely small output byte when compared to DES and AES algorithms. Execution time of the RSA algorithm is much higher when compared with DES and AES algorithms. The authors concluded taking into account the text files and test results, that DES algorithm has less encryption time whereas AES has less memory usage. However, AES and DES have very minor encryption time differences.

RSA took longest encryption time and memory use is likewise high, however the output byte is comparatively very less in RSA algorithm (Shashi & Rajan, 2011).

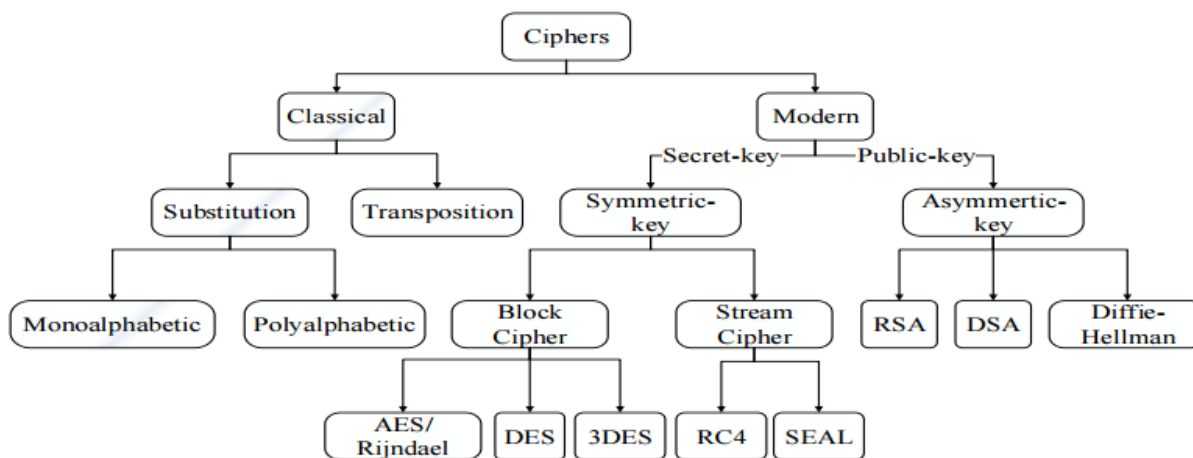


Figure 4. Classification of Encryption Methods (Gurpreet & Supriya, 2013).

Agrawal and Pradeep (2012) presented a study of the well-known symmetric key encryption algorithms; for example, DES, 3DES, AES, and Blowfish. Usually, Symmetric Key algorithms run faster when compared to Asymmetric Key algorithms; for example, RSA, etc. and the requirement of memory for symmetric algorithms is less than that of asymmetric encryption algorithms. Also, the security factor of symmetric key encryption is better than that of Asymmetric key encryption (Agrawal & Pradeep, 2012).

### Summary

Different encryption standards used by various companies and research works of various authors are discussed in this chapter. The purpose of cryptography, a high-level overview of cryptographic algorithms and the implementation details of data encryption test-bed is illustrated in Chapter III.

## **Chapter III: Methodology**

### **Introduction**

Cryptographic techniques can be implemented in a variety of ways. One such way is to use a programming language. There are many programming languages available today. Picking a strong programming language like JAVA gives an appropriate implementation of algorithms. Because of the Java features like robust, cross-platform capabilities, ease of use, and security, it has universally become a language chosen more frequently because it provides Internet solutions worldwide. It comes in different versions and this paper uses the JAVA 1.6 version.

### **JAVA Programming Language**

JAVA was released by Sun Microsystems in 1995 and is inherited from C and C++ languages. It is an open source programming language which is used worldwide today in developing web applications, websites, online games, mobile applications, and many more.

### **Advantages of JAVA**

1. Java is easy to learn: Java was developed to be easy to use and is, therefore, easy to understand, code, compile, run, debug, and finally learn when compared to other programming languages.
2. Java is object-oriented: Object oriented concept enables you to develop modular and reusable code.
3. Java is platform-independent: Platform independent is one of the most important uses of Java, as is the ability to move the code easily from one computer system to another and execute successfully. The ability to run the same Java program on different

- systems is important to software on World Wide Web, and Java achieves it 100% by being platform-independent on both the source and binary levels.
4. Java is distributed: Java is developed to make computing, distributed across different places easy with the capability of networking which is integrated inherently into it. Writing programs of networking in Java is similar to sending and receiving information to and from a file.
  5. Java is secure: Java views security at its design phase itself. The Java language, JDK –Java Runtime, Developer Kit, compiler, interpreter, and finally runtime environment is incorporated with security.
  6. Java is robust: Robust stands for reliability. Java has much emphasis on checking for possible and unexpected errors, as Java compilers can detect many issues which would first appear during the time of execution in other languages.
  7. Java is multithreaded: Multithreaded is the ability of a program to do several operations concurrently in a program. In Java, multithreaded programming is integrated into it, whereas in other languages, OS-specific functions need to be explicitly called to enable multithreading.

### **How Does JAVA Work?**

**Java Runtime Environment (JRE).** Java Runtime Environment is a runtime environment that comes with JAVA Software package. It consists of JVM (Java Virtual Machine), Java predefined classes, and Libraries. JRE is needed to run a JAVA program in the web browser.

**Java Development Kit (JDK).** Java Development Kit is a development environment that is used to develop applications or applets. It consists of JRE (Java Runtime Environment), a Java compiler (javac), the Java interpreter (Java), Java archive (jar), Java documentation generator (Javadoc) and other tools that are required for development of Java code.

**Java Virtual Environment (JVM).** The JAVA programming language is designed in such a way that a JAVA program runs in any machine independent of the platform avoiding the task of rewriting or recompiling the JAVA program for each different platform. This is accomplished by a Java Virtual machine (JVM). Initially, a Java Compiler (javac) converts the Java program into Java binary code called 'byte code.' Java Virtual Machine is present in every platform and converts the byte code to platform specific machine language which is finally processed. A JVM can interpret the byte code one instruction at a time or all instructions at a time. JVM is identified as a real processor where byte code is processed irrespective of the operating system it is running as shown in the below figure. For instance, to establish a socket connection from the source (a workstation) to a destination (a remote machine), includes an operating system call. Here, although sockets are handled differently by the two operating systems, the JVM converts the Java program code in such a way that two machines which are on different platforms should be able to connect (Simon, 2012).

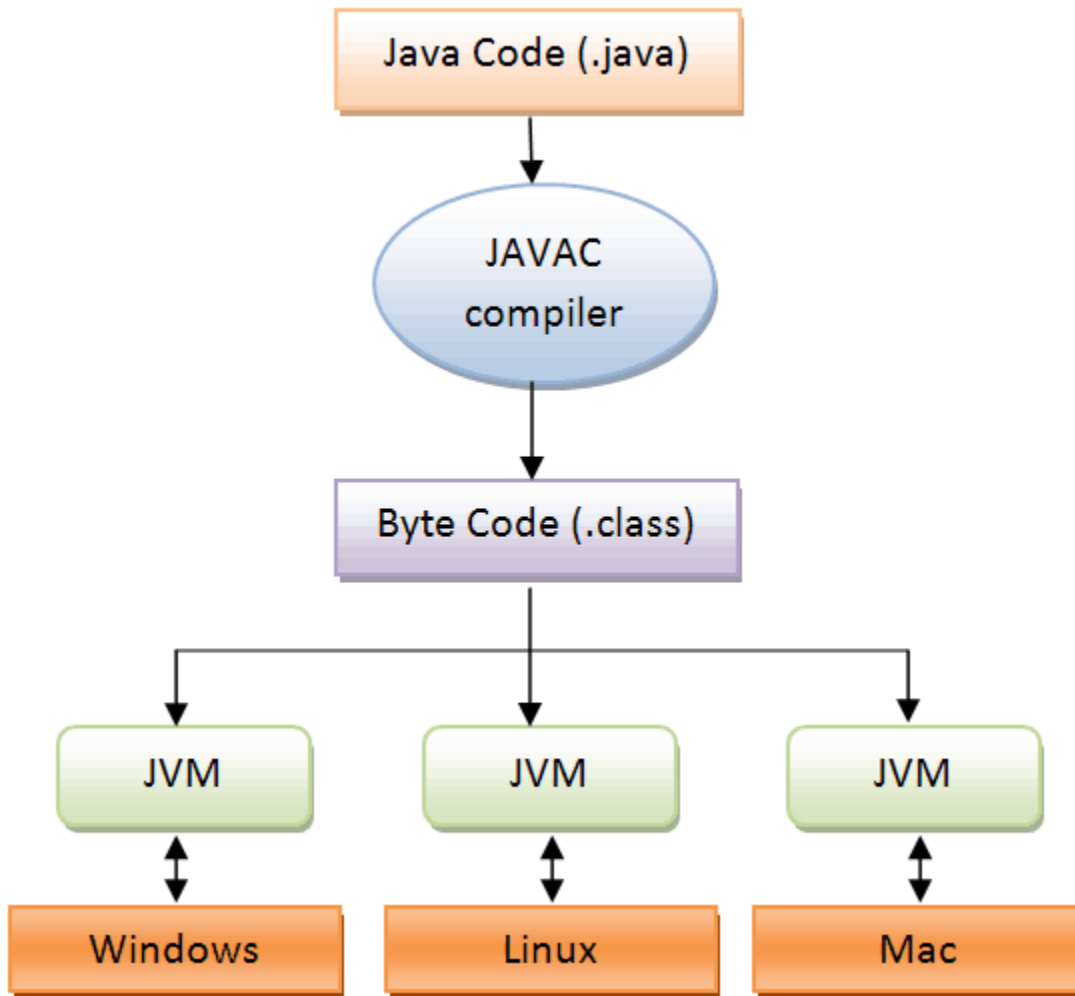


Figure 5. Byte Code Conversion of Java Virtual Machine (JVM) (Byrne, 2015).

### Swing in JAVA

Swing in JAVA is a Model-View-Controller (MVC) design that consists of a set of components providing capability of developing Graphical User Interface (GUI) components like textboxes, labels, text areas, buttons, and scrollbar's etc., which are independent of the Operating system windowing scheme. Swing has more advanced GUI components when compared to the earlier AWT (Abstract Window Toolkit) that are used to create Applets. Swing is a most powerful and flexible toolkit which provides other sophisticated components like trees, tables,

tabbed panel, scroll panes, lists etc. The hierarchy of the Swing classes are illustrated in the below figure.

Following are advantages of swing:

- **Configurable.** Swing is configurable where programmers can define their own look and feel that reflects the changes uniformly across the whole application without requiring any programmatic changes to the application code.
- **Extensible.** Users can define their own custom implementations of existing components to overriding the default ones. It is a component-based architecture, where its components are inherited from `javax.swing.JComponent`.
- **Lightweight GUI.** Swing overrides the OS's GUI framework with its own semantics. Hence it implements a lightweight GUI. For instance, every Swing component renders paint on the graphic device as a response to the `component.paint ()` method call. But unlike AWT, it delegates the task of painting to its native OS.
- **MVC.** The Swing framework makes use of Model-View-Controller design pattern. It provides a set of default view implementations, which minimizes the need to define custom implementations for the application specific components. The Model stands for the Java bean classes representing User Interface (UI) elements, View stands for the UI components that the user views, Controller stand for the request and response handler that fetches the request, processes it and sends the appropriate response to the UI.
- **Loosely coupled.** MVC in Swing provides a loosely coupled object relationship pattern, where the mapping of event listeners to a data object is performed through

programming. This feature of Swings makes the Application testing easier for programmers.

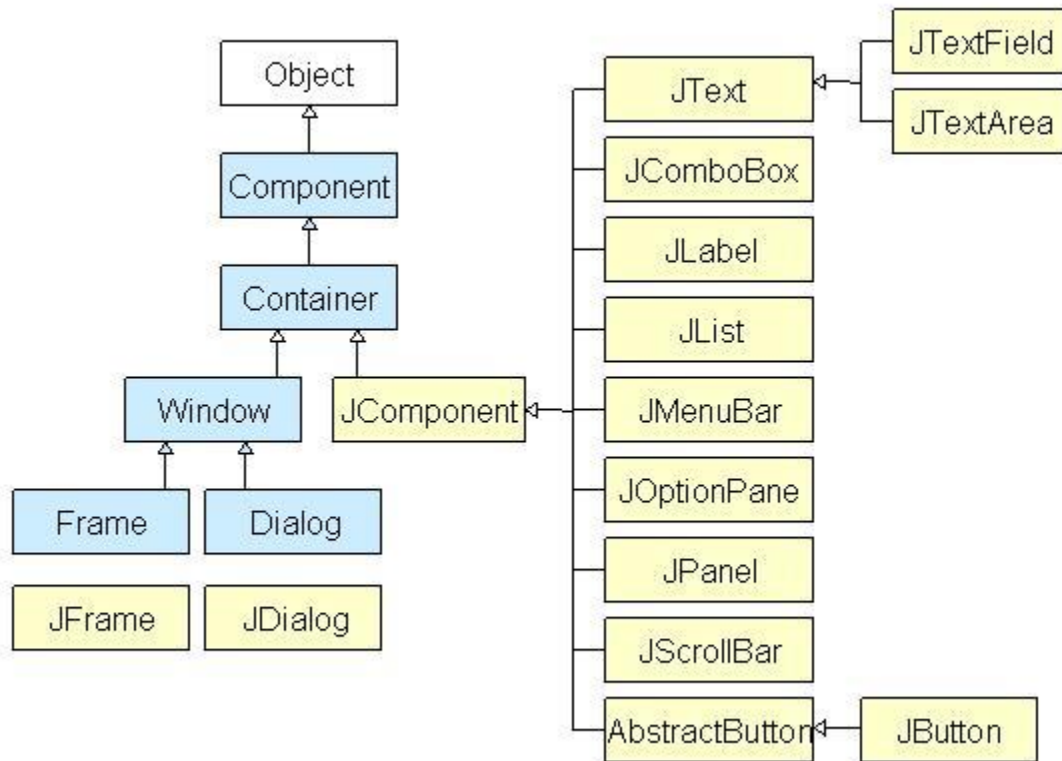


Figure 6. JAVA Swing Class Hierarchy (GURU, 2013).

### Integrated Development Environment (IDE)

There are many types of Integrated Development Environments in the market today. An IDE is a software application that enables programmers, for the development of Software. Developers use different IDEs like Eclipse, Net Beans, IntelliJ, Visual Studio, Windows PowerShell etc. for developing software projects.

**Net Beans IDE 8.2.** Net Beans is an open source Integrated Development Environment that is used for Software development by programmers. It is specifically intended for developing JAVA projects, but also supports other programming languages like C, C++, PHP, HTML5 etc.



It facilitates source code formatting, color coding, error diagnostics, debugging, reporting, easy navigation, ease of adding external jars/libraries, suggestions while coding etc. This allows programmers to compile and execute code easily and maintain the changes to code uniformly across the project. IDEs can also integrate third party libraries like Subversion (SVN), GitHub, and Apache Tomcat. It also allows to add different plugins like Check style, PMD (Programming Mistake Detector) tools for maintaining the coding standards.

Advantages include:

- Free and open source IDE: Net Beans is a free and open source IDE that is widely used across the organizations.
- Easy and Efficient Management of Projects: It is easy to use and supports efficient management of different types of projects.
- Supports a variety of programming languages: It supports different kinds of programming languages like .Net, JAVA, etc.
- Helps in writing bug free application code: It aids developers or programmers in writing error free code.
- Supports cross platform: It supports programmability across different platforms like Windows, UNIX, Linux, etc.
- Facilitates powerful tools for CSS3, HTML5 and JavaScript: It provides strong and standard tools for writing and supporting CSS3, HTML5, Java script, etc.
- An advanced set of default plugins: It provides many advanced default plugins that eases the programming.

- Supports latest technologies: It supports the latest and advanced technologies including the upgraded versions.
- Smart, fast and Easy code editing: It enables fast, easy and smart modifications of code with many short-cut keys.
- Rapid and Fast User Interface Development: It allows rapid and fast development of User Interface for any type of web application.
- Provides default Look and Feel Themes: The programmers can use a set of default themes provided by Net Beans that enhances the look and feel of the User Interface.

### **Purpose of Cryptography**

Cryptography gives various security objectives to guarantee the privacy of data, no data alteration etc. Because of the featured security advantages of cryptography, it is broadly utilized today (Denning, 1982). Following are the different objectives of cryptography:

- Confidentiality: Data in a PC is transmitted and must be accessible only to authorized people / users and not to any other individuals.
- Authentication: The data received by any system need to verify the sender's identity by checking whether the data are sent from an authorized party or not.
- Integrity: Just the authorized party is permitted to change the transmitted information. Nobody other than Sender And receiver is permitted to change the given message.
- Non Repudiation: Non Repudiation feature guarantees that neither the sender, nor the receiver of the message ought to have the ability to reject the transmission.
- Access Control: Access Control feature of cryptography ensures that just the authenticated individuals have access to the given information. Cryptography,

Encryption algorithms can be characterized into two general types - Symmetric and Asymmetric key encryption types (Thambiraja et al., 2012).

### **Cryptographic Algorithms**

**Data encryption standard.** The Data Encryption Standard (DES) is a symmetric key algorithm Developed in the early 1970s at IBM. The Data Encryption Standard is the most widely used cipher. It was designed in 1977 by IBM and it can to resist all attempts at cryptanalysis. The Data Encryption Standard is a square figure, which means a cryptographic key and calculation are connected to a piece of information all the while as opposed to one piece at once. To scramble a plain text message, DES bunches it into 64-bit pieces. Every bit is encrypted utilizing the secret key to generate a 64-bit output cipher text by a method for stage and substitution. The procedure includes 16 adjusts and can keep running in four unique modes, or scrambling pieces exclusively. Decoding is basically the opposite of encryption, taking after the same strides yet turning around the request in which the keys are connected. The quantity of conceivable keys depends on the length of the key and the plausibility—of this kind of assault. DES utilizes a key of 64-bits, out of which eight of the bits are utilized for equality checks, successfully constraining the way to 56-bits. Henceforth, it would take a great of  $2^{56}$ , or 72,057,594,037,927,936, endeavors locate the right key. It divides the data to be encrypted into a particular sequence of blocks of 64-bit and also uses a key of 56-bit to do a series of mathematical transformations to it. There are various variations of the DES algorithm, namely, cipher block chaining, in which every block of data applies XOR function with the previous block even before encryption, and on the other hand, in triple-DES, the technique of DES is applied three times in the series. The purpose of the DES algorithm is to give a standard

methodology for securing confidential and unclassified information. In this process, both encryption and decryption process use the same key.

DES algorithm consists of the following steps:

1. Encryption. DES takes an input of plaintext that has 56-bit key where 8 bits are of parity and is 64-bit long that finally generates an output of 64 bit block. The plaintext block needs to shift the bits around. The eight bits of parity are taken out of the key by applying the key to its Key Permutation.

The plaintext and key will be processed by the following steps:

- a. The key is split into two halves of 28.
- b. Every half of the key based on the round is rotated/shifted by one or two bits.
- c. The halves are recombined and then are subjected to a permutation—a compression to reduce the key right from 56 bits to 48 bits. The keys compressed are used to encrypt this plaintext block of rounds.
- d. Then the rotated/shifted key halves generated from Step b are used in the next round.
- e. After that, the data block is then divided into two halves of 32-bit as illustrated in Figure 7 and processed alternatively. This crisscrossing is called Feistel scheme.
- f. One half is subjected to a permutation-expansion to increase the size to 48 bits.
- g. Output of above step is then applied XOR with 48-bit key compressed Step c as illustrated in Figure 7. Output of above step is given as an input to the S-box that substitutes key bits and hence reduces the block of 48-bit back to 32-bits. Output of above step is subjected to a P-box to permute the given bits. The output of P-

box is applied XOR with other half of the data block. The two halves of data are then swapped to be the next round's input. This is repeated for 16 rounds (Mahajan, & Sachdeva, 2013).

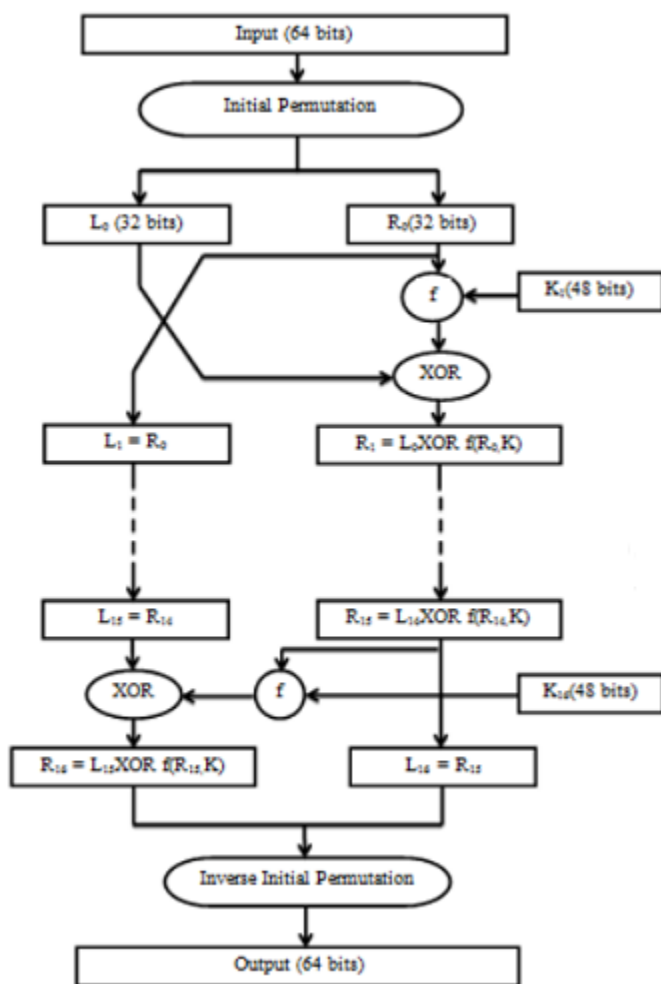


Figure 7. Data Encryption Standard Functionality (Mahajan, & Sachdeva, 2013).

1. Decryption. Decryption in DES uses the same structure as that of encryption, but the keys are used in reverse order.
2. Key schedule (n.d.). From the initial key with 64 bits, 56 bits are selected and the remaining 8 bits are used as parity check bits or discarded. The 56 bits are divided

into two halves with 28 bits each. Each half is process separately. In the next rounds, each half is rotated left by 1 or 2 bits and finally, PC2 (Permutation Choice 2) selects 48 sub key bits and 24 bits from the left and right halves respectively. The symbol '<<<<' makes sure that every sub key should use a different set of bits as illustrated in the below figure. In decryption, a similar process is used, except that the sub keys are used in reverse order.

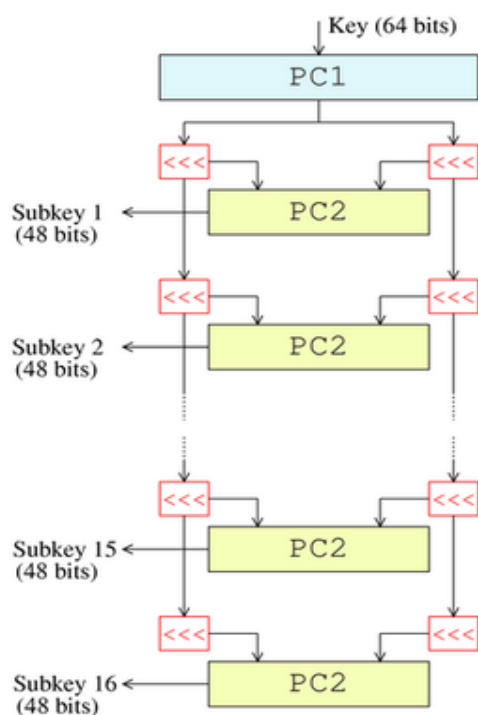


Figure 8. DES Key Schedule (Key schedule, n.d.).

**The Feistel scheme.** The crisscrossing used in the DES algorithm is called a Feistel scheme. This scheme makes sure that encryption and decryption are performed in a similar manner. The sub keys are used in a reverse order in decryption which is the only difference in the process of encryption and decryption making implementation simpler without the need of separate encryption and decryption algorithms.

The Feistel function operates in the following four stages on 32 bits at a time:

- Expansion: Using the expansion permutation represented by the E in Figure 9, the 32-bit is expanded to 48 bits by duplicating the bits. The output contains eight 6-bit pieces that are equivalent to 48 bits each containing a corresponding copy of four input bits and adjacent bit present on either side.
- Key mixing: The result of Expansion stage is applied XOR with a sub key. Sixteen sub keys of 48 bits are derived from 16 rounds of DES Key schedule process. This is termed Key Mixing.
- Substitution: The resultant block is divided into eight 6-bit pieces that are passed to Substitution boxes (S-Boxes) for processing. Each S-box replaces 6-bits with four output bits.
- Permutation: The final 32 outputs generated from S-Boxes are rearranged based on a permutation (P-Box) as illustrated in Figure 9. After permutation, the output bits are distributed in the next rounds across four S boxes.

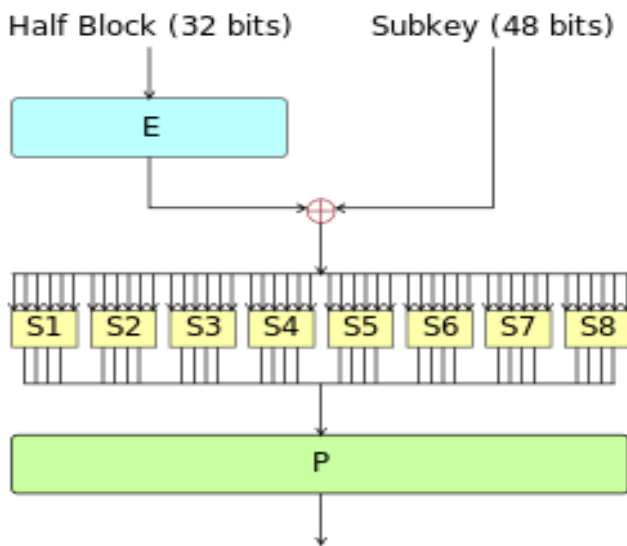


Figure 9. Feistel Scheme of DES (Wikiversity, n.d.).

**3-DES encryption with cipher block chaining.** Triple DES was designed to illustrate the flaws in previous DES without developing a whole new cryptosystem. It simply increases the size of key of DES by applying the logic three times in series with three different keys. Hence, the combined key size is 168 bits (3 x 56). It is always viewed with some doubt since the original algorithm was not developed to be used in this context, but no serious issues are seen in the 3DES design and, hence, it is the available cryptosystem today used in many Internet protocols. Triple Data Encryption Standard (DES) is a sort of automated cryptography where piece figure calculations are connected three times to every information square. The key size is expanded in Triple DES to guarantee extra security through encryption capacities. Every piece contains 64 bits of information. Three keys are alluded to as group keys with 56 bits for every key as illustrated in Figure 10. There are three entering keys in data encryption benchmarks: (a) Key 1 and Key 2 being free keys, (b) all three keys being indistinguishable, and (c) Key alternative Number 3 is named as triple DES. The key length of triple DES consists of 168 bits; however, the key security tumbles to 112 bits.

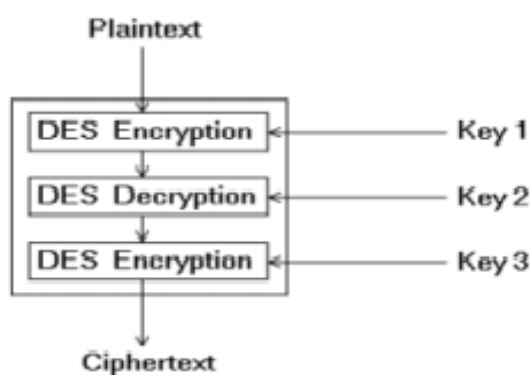


Figure 10. Functionality of Triple DES Algorithm (Singh & Bharti, 2013).



**RSA algorithm.** The RSA (Rivest-Shamir-Adleman) algorithm is the most powerful and vital public-key cryptosystem. RSA gets its security from the trouble of figuring extensive whole numbers that are the result of two vast prime numbers. Duplicating these two numbers is simple, yet deciding the first prime numbers from the aggregate figuring is viewed as infeasible because of the time it would take not withstanding utilizing today's super PCs.

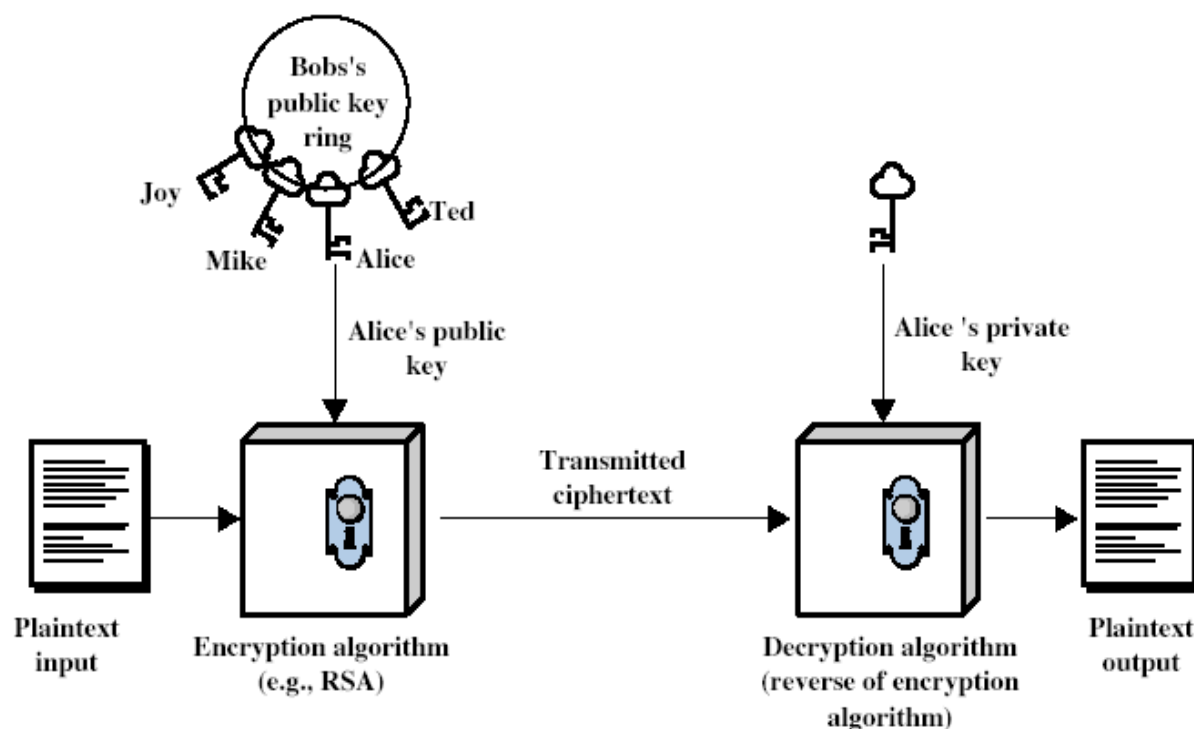


Figure 11. Public Key Cryptography (Gerasimos, 2012).

Two huge prime numbers,  $p$  and  $q$ , are produced utilizing the Rabin-Miller primality test calculation. A modulus  $n$  is figured by duplicating  $p$  and  $q$ . This number is utilized by the private keys and gives the connection between them. Its length, typically communicated in bits, is known as the key length. The key comprises of the modulus  $n$ , and an open type,  $e$ , which is ordinarily set at 65537, as it is a prime number that is not very huge. The private key comprises

of the modulus  $n$  and the private type 'd', which is ascertained utilizing the Extended Euclidean calculation to locate the multiplicative converse of  $n$ .

As illustrated in Figure 11, the RSA works like this:

1. Alice chooses two large primes  $p_A$  and  $q_A$ .
2. Alice computes  $n_A = p_A q_A$  and  $\phi(n_A) = (p_A - 1)(q_A - 1)$
3. Alice chooses an integer  $e_A$  with  $\gcd(e_A, \phi(n_A)) = 1$ , possibly at random.
4. Alice computes  $d_A \equiv e_A^{-1} \pmod{\phi(n_A)}$ .
5. Alice's public key is  $(n_A, e_A)$ . She distributes this. Her private key is  $d_A$ . She keeps this secret. Alice can discard  $p_A$ ,  $q_A$ , and  $\phi(n_A)$ .
6. If  $2^k \leq n_A < 2^{k+1}$ , Alice's function of encryption for short texts ( $k$  bits or even less, so  $M < n_A$ ) is:  $E_A(M) = M^{e_A} \pmod{n_A}$ . Anyone can compute  $E_A(M)$ . A long message is encrypted by dividing it into blocks of  $k$ -bits, and then encrypting every block separately. Note that each encrypted block has  $k+1$  bits.
7. Alice's function of decryption for small messages is:  $D_A(M) = M^{d_A} \pmod{n_A}$ , given  $0 \leq M < n_A$ . No one else other than Alice (or others who has Alice's private key) can compute this. Note:  $D_A(E_A(M)) \equiv (M^{e_A})^{d_A} \equiv M^{e_A d_A} \equiv M \pmod{n_A}$  since  $e_A d_A \equiv 1 \pmod{\phi(n_A)}$ .

Once Alice has done this, she can (a) receive messages that are encrypted from Bob (or others), and (b) send digitally-signed messages to Bob (or anyone else). If Alice needs to send encrypted texts, or receive digitally-signed texts to and from Bob respectively, Bob needs to choose his own private and public keys,  $d_B$  and  $(n_B, e_B)$ . Bob sends a short message  $M$  (at most  $k$  bits) to Alice like this:

1. Bob encrypts  $M$  as  $MeA \pmod{n}$ , and sends  $MeA$  to Alice. (Note Bob knows  $eA$  and  $n$ .)
2. Alice decrypts  $MeA$  as  $(MeA) dA \equiv M \pmod{n}$ . Thus Alice recovers  $M$ . (Make a note that Alice actually recovers the  $M$  value  $\pmod{n}$ , but this is equivalent to  $M$  satisfying  $M < n$ .) For big messages, Bob could divide the message to  $k$ -bit blocks, and encrypt separately every block. Alice would divide the message that is encrypted in  $k+1$  bit blocks, and decrypt separately every block.

**Advanced Encryption Standard (AES).** During the 1990s, the United States National Institute of Standards and Technology (NIST) conducted a competition to design a substitute for DES. The winner, Rijndael, was announced in 2001. This made the RSA algorithm the new Advanced Encryption Standard (AES). AES involves three square figures, AES-128, AES-192 and AES-256. Every figure encodes and decodes information in squares of 128 bits utilizing cryptographic keys of 128-, 192-, and 256-bits, individually. (Rijndael was intended to handle extra piece sizes and key lengths, however the usefulness was not highlighted in AES.)

Symmetric algorithms utilize the common secret key for both decryption and encryption, so both the sender and receiver should be aware and be able to utilize the same secret key. Every single key length is esteemed adequately to make sure ordered data requiring either 192-or 256-bit key lengths. There are 10 rounds of 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys (a round consists of a few base steps that include shifting, mixing, and substitution of the input plaintext and change it into the last yield of cipher text). Rijndael mixes up the SPN model by including Galios operations of the field in every round. A bit similar to arithmetic operations

of RSA modulo, the field operations of Galios produced gibberish, but can be inverted mathematically. AES has Security and, in addition, it has a relation between time and cost.

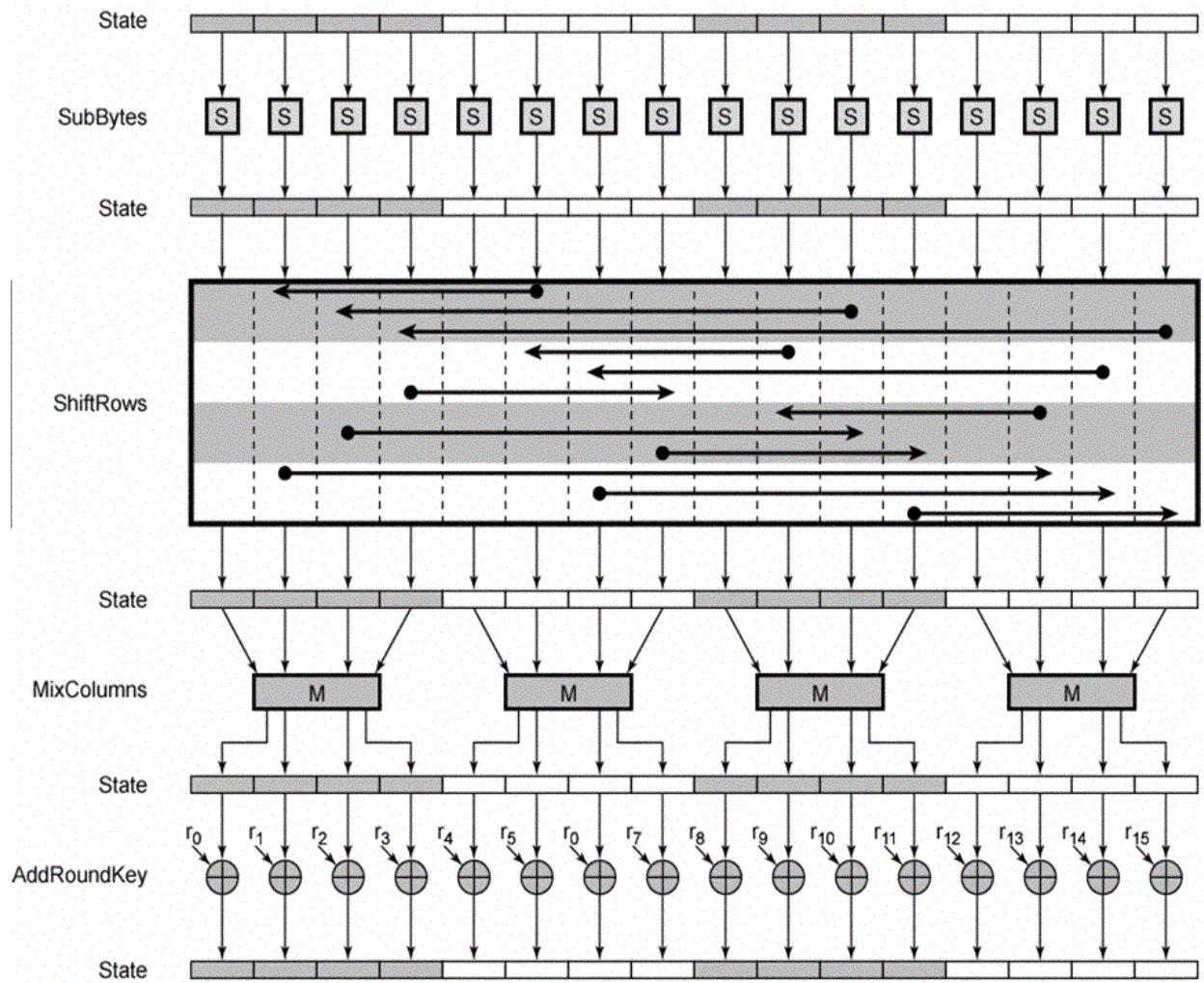


Figure 12. Steps of AES Encryption (Pranav, 2005).

**Encryption.** Following are the key steps involved in the AES algorithm as illustrated in Figure 12:

1. Key Expansion
2. Initial Round
  - Add Round Key

### 3. Rounds

- Shift Rows
- Sub Bytes
- Add Round Key
- Mix Columns

### 4. Final Round

- Sub Bytes
- Shift Rows
- Add Round Key

**Key expansion.** Key Expansion is an important step in the AES algorithm. There are multiple rounds involved in AES encryption and decryption process. The initial is manipulated and converted to generate sub keys for the subsequent rounds. The key expansion step takes a key as input that is of bytes  $4 \cdot N_k$ . Here  $N_k$  can be 4 or 6 or 8. The output of the key expansion is named as an expanded key that is of bytes  $4 \cdot N_b \cdot (N_r + 1)$ , where  $N_r$  represents the number of rounds and  $N_b$  is always a constant that equals to 4.

**Sub bytes.** In Sub Bytes step, each byte is replaced with another byte located at 8-bit substitution box (S-Box) representing a lookup table.

**Shift rows.** The Shift Row step operates on each row of the byte matrix. In each row of the state, the bytes are shifted to the left. The number of bytes shifted is based on certain offset for each row. In the first row, the bytes are left stable. In the second row, each byte is shifted left by one position. Similarly, in the fourth and fifth rows, all bytes are shifted by two and three positions left respectively. This logic remains same for any size of block either 128 or 192 bits.

**Mix columns.** In Mix Columns step, each four bytes of a column are united using a linear transformation. This step takes input of four bytes and generates output of four bytes. Here, the output bytes are affected by each input byte. In this step, each column is transformed by multiplying by a constant matrix generating new values in the column. This matrix multiplication includes both addition and multiplication. The addition implies a simple XOR function. The multiplication implies applying a polynomial order of  $x$  to the power of 7.

**Add round key.** In the Add Round Key step, the state is combined with the sub key. In each round, Rijndael's key schedule is used to derive a sub key from main key where the size of the sub key is same as that of the state. Each byte of the state is added with the sub key using bitwise XOR operation.

As illustrated in Figure 13, in the final round the step of column mixing is not performed. The round keys are derived from the key by a key schedule. For each round, we need a round key of the same size as the size of the state. This is achieved by recursive expansion of 11 the key to the size of  $(\text{number of rounds}) * (\text{size of state})$ . From this expanded key, the round keys are taken sequentially.

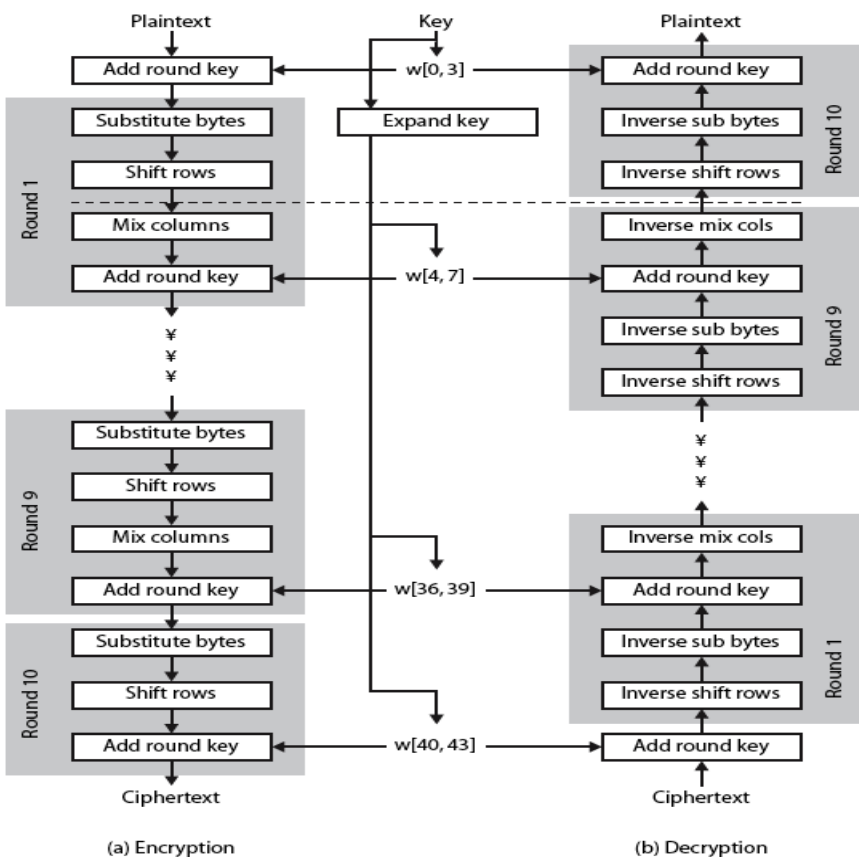


Figure 13. AES Structure (Shankar & Jenifer, 2013).

**Decryption.** The inverse of a round is given by Add Round Key, Inverted Mix Column, inverted Shift Row, and inverted Byte Sub. The inverse of a final round is got by Add Round Key, inverted Shift Row and inverted Byte Sub. After which a final Add Round Key is done.

**AES-128/256.** Advanced Encryption Standard comes with a 128-bit block size, but it also supports 128, 192, 256 bit keys. Usually huge sized keys are most likely required for building an efficient hashing function. AES takes after the custom of square ciphers. NIST gave as its explanations behind selecting Rijndael that it performs exceptionally well in all possible modes in both hardware and software over a wide nine range of environments. It has efficient setup time for key and has low requirements of memory; furthermore, its operations have the

ease to protect against timing and power attacks. In the real world, regularly utilized ciphers are block ciphers. They change a fixed size data block into another fixed size data block with the help of the function that the key chooses. In the case where the key, input, and output blocks have  $n$  bits, a one-to-one mapping of the integers of  $n$  bits to permutations of integers of  $n$  bits, is defined by the block cipher. In case the same block is twice encrypted with the same key, the subsequent cipher blocks of text are also similar. This method of encryption is named as electronic codebook (ECB). This data could be helpful to an attacker. To make identical blocks of plaintext to encrypt to a different text blocks of cipher, two modes are usually utilized as standards: AES is an iterated cipher block with a variable length of the block and a variable length of the key. The block and the key lengths respectively can be picked as 128, 192, 256 bits. The block on which the operations are performed is known as a state. A state is represented as an array of eight bits having four columns. The count of columns is equivalent to the length of the block divided by 32. Hence there are four rows and eight columns in Table 2, where  $8 \times 4 = 32$ . Each element in the table is denoted as  $A_{i,j}$ , where  $i$  represents row and  $j$  represents column as shown in the below table. Same structure is applied to the cipher key. Below is a sample state with 192 bits (Anush, 2000).

Table 2

*AES State Structure*

A0,0	A0,1	A0,2	A0,3	A0,4	A0,5	A0,6	A0,7
A1,0	A1,1	A1,2	A1,3	A1,4	A1,5	A1,6	A1,7
A2,0	A2,1	A2,2	A2,3	A2,4	A2,5	A2,6	A2,7
A3,0	A3,1	A3,2	A3,3	A3,4	A3,5	A3,6	A3,7

(Anush, 2000)



Similar to DES, AES contains a number of rounds. The number of rounds is based on the block and key lengths respectively and is represented in Table 3, where the first column represents the block length and first row represents the key length.

Table 3

*AES Rounds Based on Block and Key Length*

	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

(Anush, 2000)

**DES CBC mode with RSA-MD5.** DES CBC mode with RSA-MD5 (des-CBC-md5) is a form of DES algorithm that includes encryption and decryption of input text using keys of 56 bits, blocks of eight bytes in CBC (Cipher Block Chaining) mode and Initialization Vector (IV). The passwords generated from an application specific algorithm is used to derive keys. Un-keyed and Encrypted MD5 (Message Digest 5) hash is used to protect the Data Integrity of the Input Text. This MD5 is associated with the checksum of rsa-md5-des. This is one of the strongest encryption mechanisms that are interoperable.

**Cipher Block Chaining (CBC).** CBC is used to operate on a block cipher, where the chunk of bits is encrypted and decrypted as a single block or unit. CBC uses a single cipher key on the entire block. CBC uses Initialization Vector (IV) of a particular length and has a key feature of using chaining mechanism. This chaining mechanism makes the decryption of cipher block to depend on every preceding cipher block. Hence, the preceding blocks validity is captured in the previous immediate cipher block. As a result, any minor error in a cipher block

affects the decryption process of all the subsequent blocks. The order of cipher blocks also cannot be changed as it causes the decryption to be corrupted. In CBC, every plaintext block is applied to XOR function with the previous immediate cipher block and then they are encrypted as illustrated in Figure 14 (Rouse, 2007).

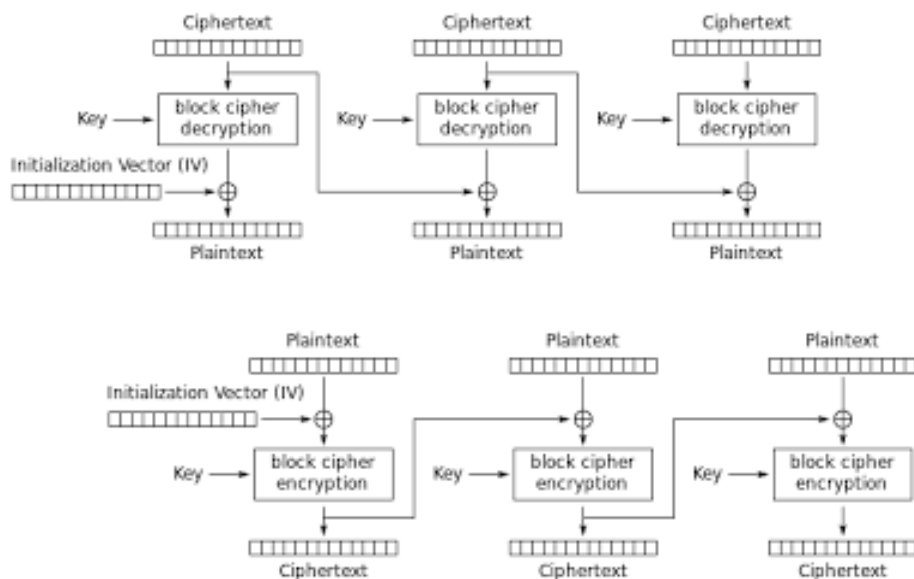


Figure 14. Cipher Block Chaining (CBC) Mode Encryption and Decryption (Murat, 2007).

**Message Digest 5 (MD5).** MD5 was developed by Ronald Rivest in 1991 (Dinesh, 2013). It is a popular hash function that produces a hash value of 128-bit or more as illustrated in the below figure. It can also be used as a checksum to validate the data integrity. MD5 is currently widely used in the software Industry to ensure that the file transferred from a source is intact with the file arrived.

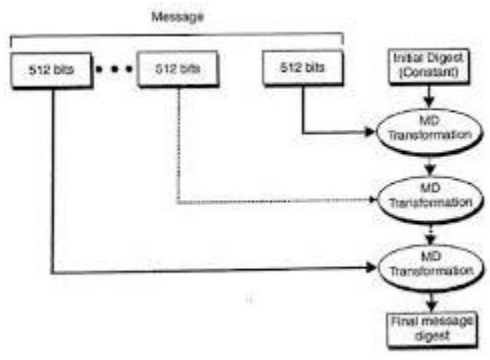


Figure 15. Generation of Message Digest Using MD5 (Dinesh, 2013).

For instance, file servers provide an MD 5 checksum that is pre-computed for the files, in such a way that the downloaded file can be compared with it by the users for data integrity as illustrated in Figure 16. Most of the UNIX based systems today use MD5 utilities in their packages.

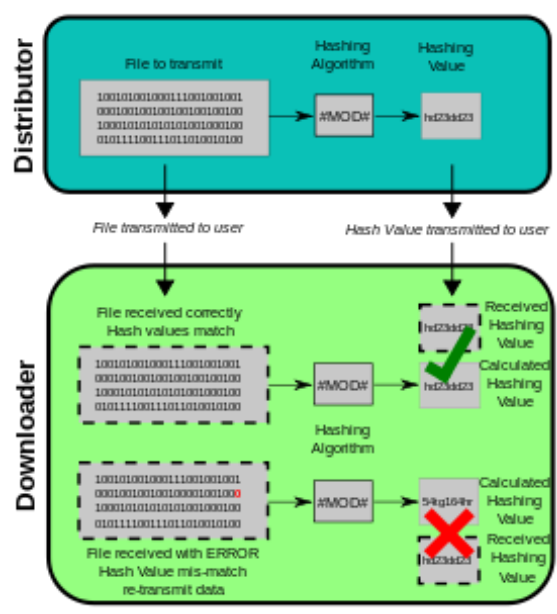


Figure 16. MD5 File Transfer (Pluke, 2012)

## **Other Encryption Algorithms Under Study**

There are many other encryption and decryption algorithms in today's world which are being used by many companies. Apart from DES, Triple DES, AES, and RSA below are the other commonly used encryption and decryption algorithms:

- RC6 - Rivest cipher
- des-cbc-crc - DES cbc mode with CRC-32
- des-cbc-md5 - DES cbc mode with RSA-MD5
- des-cbc-md4 - DES cbc mode with RSA-MD4
- des3-cbc-sha1-kd - triple DES cbc mode with HMAC/sha1
- AES-256 CTS mode with 96-bit SHA-1 HMAC
- aes256-cts-hmac-sha1-96 - aes256-cts
- des3-cbc-sha1 - des3-hmac-sha1
- des-hmac-sha1 - DES with HMAC/sha1
- aes128-cts-hmac-sha1-96
- aes128-cts - AES-128 CTS mode with 96-bit SHA-1 HMAC

## **Design of a Test-Bed for Encryption Processing Overhead**

The project for the Design of Test-Bed for encryption overhead is developed using JAVA JDK 1.7 and Net Beans 8.2 IDE (Integrated Development Environment). The following jars are included in the project:

- commons-codec-1.7.jar
- substance-lite.jar
- jfreechart-1.0.16-demo-all.jar

The most imperative part of implementing the functionality, especially for security purposes, is testing. Choosing the best reliable implementation for a given functionality can be brought out by comparing the test results of the performances of different algorithms. After implementing the encryption algorithms in the 'EncryptionAlgorithms' project using JAVA programming language, the method `System.nanoTime()` is used to calculate the time taken to encrypt and decrypt all the listed algorithms. The difference of the start time and end time of each algorithm is calculated as the time taken to run the algorithm. After the execution of all algorithms, time taken to execute all algorithms is compared, and the one with less time is displayed as the best algorithm for the given input text in the User Interface.

The algorithm performance can also be tested in LINUX environment. A powerful Linux command, namely, 'time' is used to compare a variety of timing factors in a Linux environment. Time results in giving following details of a program run:

- Real Time: Elapsed real time, usually in seconds.
- User Time: User mode—Total CPU-seconds that the process spent.
- Sys Time: Kernel mode—Total number of CPU-seconds that the process spent.

## **System Design**

**UML diagrams.** UML corresponds to Unified Modeling Language. UML is a standardized extensively helpful representation of the process flow in the field of object arranged programming structure. This standard is administered and was made by the Object Management Group.

The goal of UML is to represent a making models of things in organized PC programming. In its present structure, UML contains two imperative parts—a documentation and

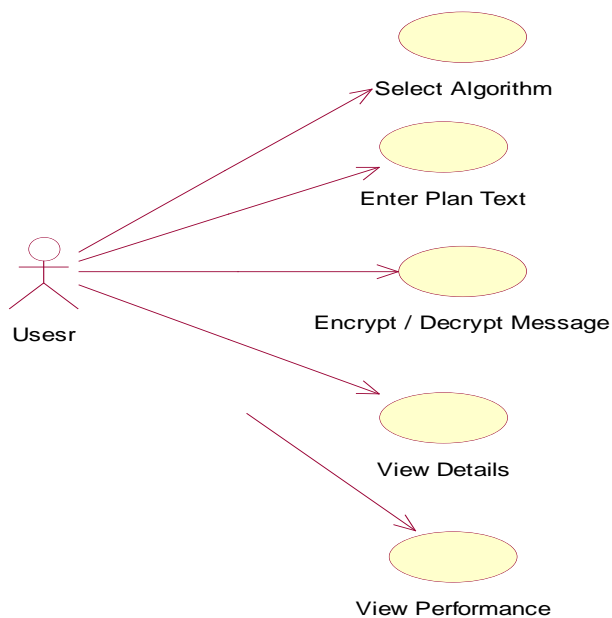
a Meta-model. Later, some sort of methodology or strategy may be added to, or associated with, UML.

The Unified Modeling Language is a standard form for demonstrating Visualization, Archiving, and Constructing the relics of programming system and, likewise, for business flow representations and other non-programming structures. The UML addresses a gathering of best designing practice that have exhibited success in the display of complex and huge structures. The UML is an imperative method of making things arranged in programming and the process flow management. The UML uses generally graphical documentations to express the diagram of programming work.

**UML Goals.** The objectives of the UML are as follows:

1. Provide clients with a well-prepared, ease-to-use, representational visual displaying Language intending that they can easily create or trade important models.
2. Provide specialization and extendibility systems to improve the key ideas.
3. Clients need not to concentrate on programming specific formats and advancement process.
4. Provide a professional representation with extended options for comprehending the displaying of process flow.
5. Encourage the enhancement of showing Object Oriented concepts.
6. Support elevated amount of improvement ideas, for instance, systems, segments, coordinated efforts, and examples.
7. Integrate the best design practices which representing the process flow.

**Use case diagram.** A Use case diagram is a form of Unified Modeling Language (UML) which gives a behavioral blueprint outlined and produced using a Use-case examination. Its primary goal is to demonstrate a graphical representation of the process flow by a framework using on-screen characters. It arranges different objects and builds relationships between them by applying conditions between those use cases. The objective of using a use case diagram is to show what objects are involved in the functionality and how they are related to on-screen characters. User feel that the diagram is useful in understanding the operations performed between entities.



*Figure 17.* Use Case Diagram for Design of Data Encryption Test-Bed.

**Class diagram.** In Software coding, a class diagram plays a major role in representing the model objects and the functions developed for operating on them. A class diagram in the Unified Modeling Language (UML) is a kind of constant structure representation that shows the structure of a framework using the system's classes, their features, operations, strategies, characteristics and the dependencies among the classes. It depicts which class contains what

data. A class diagram comes with various symbols and arrows for showing the relationships among different classes. It represents class properties like inheritance, object delegation, etc. The arrow mark shows the flow of method invocations and the numbers on arrow marks shows the type of relationship such as one-to-one, many-to-many, many-to-one, and one-to-many.

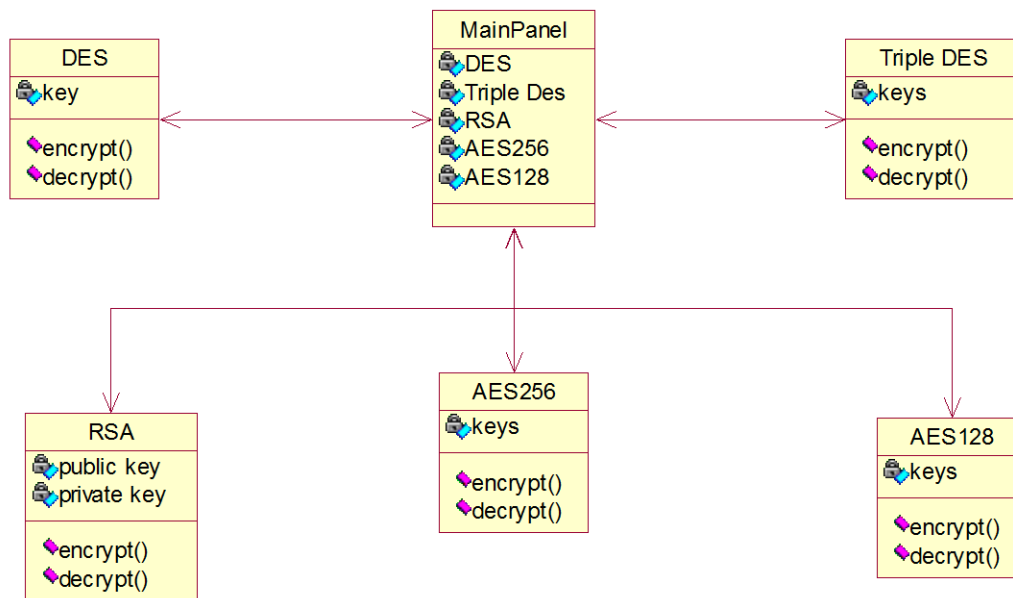


Figure 18. Class Diagram for Design of Data Encryption Test-Bed.

**Sequence diagram.** A sequence diagram is imperative in any sequence oriented methodologies. It is a form of Unified Modeling Language (UML) that represents how a request is processed and what are all the entities involved in the process. It shows the design of a Sequential Message diagram. These are also called as timing diagrams as the time span of each operation is illustrated clearly in the Sequence diagrams. The arrow marks show the flow of process between different objects. The names of operations can be clearly typed on top of the arrow mark to make users understand which operation is performed at which step and on which object. These diagrams also depict if an object or operation is an inactive or dead state.



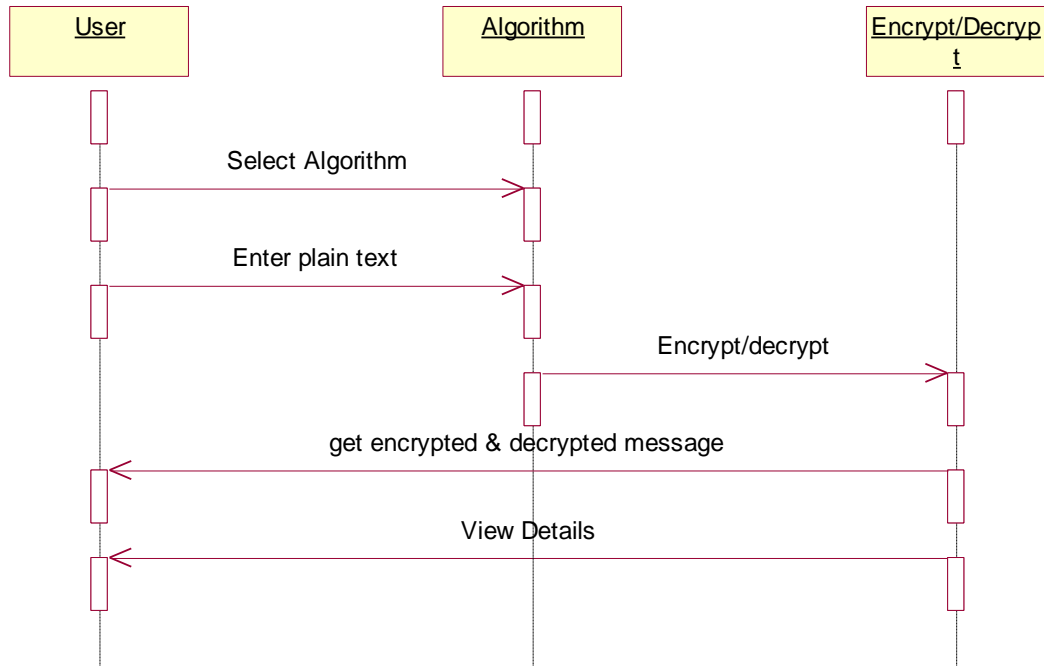
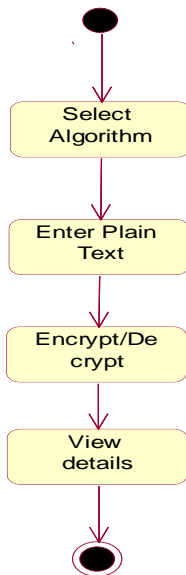


Figure 19. Sequence Diagram for Design of a Data Encryption Test-Bed.

**Activity diagram.** Activity diagrams are a form of Unified Modeling Language (UML) where graphical representations of workflow are illustrated in a step by step process and activities are represented for cycle, decision, and simultaneousness. In Activity diagrams, workflow segments can be shown in a proper system depict the business and process flow. An activity diagram represents the activities between different entities. The start point of the process flow can be shown by specific symbols and the end of flow can also be represented by using a specific end symbol. The arrows depict the stream of control between objects. Different states of objects can be represented using different structural symbols according to the project requirement.



*Figure 20.* Activity Diagram for Design of Data Encryption Test-Bed.

**Collaboration diagram.** Collaboration diagram is another form of Unified Modelling Language (UML) that is also known as interaction diagram. It shows the flow of operations among different entities. It is similar to sequence diagram, but the difference is that a collaboration diagram is used to represent a proper visualization of an organization structure and the interaction between the objects or links associated with the organization. The arrows show the flow of the process and the operations can be specified along with the arrows that eases the understanding of the work flow.

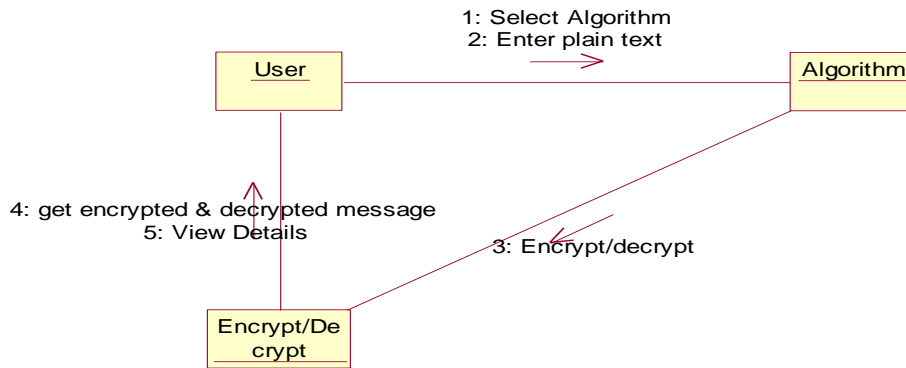


Figure 21. Collaboration Diagram for Design of Data Encryption Test-Bed.

### Interface Design

The user interface of the “Design of a data encryption test-bed to analyze encryption processing overhead” research project consists of the below options:

- After running the project, a pop up window opens which is user friendly and reusable.
- The user can select the Encryption algorithms from the list of available algorithms in the drop down, for which he would like to do the performance analysis.
- The user can enter the input text message that is needed to be encrypted and decrypted.
- There would be a button named “Analyze Processing Overhead” on the main window.
- Once the user clicks the button, a pop up displays the best algorithm for the given text message entered.
- After the user clicks on the OK button, individual performances of each algorithm get displayed.

- It clearly displays the time taken to encrypt and decrypt the input message for each executed algorithms. It also displays the best reliable encryption and decryption algorithm amongst the selected algorithms in a graphical representation.

## **System Testing**

The purpose behind testing is to discover errors. Testing is the path toward finding to locate every possible flaw or error in a work item. It gives a way to deal with checking the components, sub components in a completely or partially implemented project. Testing process ensures the reliability of the programming that satisfies the project necessities and customer requirements. There are different types of testing. Each type of testing addresses a specific testing needs.

**Types of testing.** Types of testing include Unit Testing, Integration Testing, Functional Testing, System Testing, White Box Testing, Black Box Testing, and Acceptance Testing.

- **Unit testing.** Unit testing incorporates the configuration of analyses that supports that the actual project rationale is met appropriately, and that the framework inputs generate subsequent reliable outputs. All the branches of code and internal functionalities should be acknowledged and tested. Before the application goes live, unit testing is an approach that ensures the functionalities of individual programming units of the application. This is an essential test that relies upon the data that is sent as input or parameters and the method of invoking the individual unit. Unit tests perform key tests at fragment level and test a specific business strategy, application, as well as framework setup. Unit tests ensure that each unit for a business strategy

performs precisely to the reported points of interest and contains detailed sources of information and results that are expected.

Unit testing example: Unit testing forms as a vital aspect of an integrated code snippet. This testing is ought to be performed before integration testing in the product lifecycle. For instance, when a banking application is developed, the individual components, like a savings account or checking account modules, need to unit tested.

Unit testing includes the following two important stages:

- Test strategy and approach. Field testing is usually performed physically, whereas software testing can be performed virtually.
- Test objectives. All field entries should be working legitimately. Pages should be derived from the distinguishable link connection. The actual entry screens or messages or actions should not be altered.

Features to be tested include (a) check that the sections have the right configuration, (b) ensure that no copy of sections will be allowed, and (c) make sure that all connections lead the client to the right or correct page.

- **Integration testing.** Integration tests are expected to test the programming components in conjunction with other components to make sense of whether they truly continue running as one framework. Testing is event driven and is more concerned with the crucial effects of fields on screens. Integration tests display results disregarding the way that the sections were solely satisfied, as showed up by successful unit testing, the integration of segments is correct and reliable. This testing is especially used to uncover the issues that rise up out of the integration of individual programming parts or components.

- Integration testing example. Integration testing must be performed after interfacing the application with any outside resources or interfaces. In software terminology, it can be viewed as an incremental testing of two or more programming segments that are coordinating on a solitary stage to identify the flaws caused by the integration of external interfaces. For example, in a banking application, after the individual components, like saving account and checking account, are developed, an integration testing needs to be performed for testing the combined operation of transactions.
- Test results. All the test cases identified in a typical integration testing must be passed successfully without encountering any defects.
- Functional testing. Functional testing is performed on methods or procedures of a software project. This testing shows that all the requirements are met as mentioned by the business process and specialized prerequisites, client manuals, and system documentation. Functional testing focuses on the below parameters:
  - Valid Input: Identified a set of valid input arguments or objects must be accepted.
  - Invalid Input: Identified set of invalid input arguments or objects must be rejected.
  - Functions: Identified functions need to be executed.
  - Output: Identified a set of output of the application must be generated.
  - Systems/Procedures: Procedures or interfacing systems need to be called.

The functional testing in any organization is focused mainly on requirements, key limits, or exceptional experiments. Additionally, systematic scope used to identify the information fields, Business process streams, progressive procedures and predefined forms needs to be considered while testing. Before the functional testing is completed, additional tests are identified to be performed and the issues or discrepancies of current tests are resolved.

- **System testing.** System testing ensures that the programming framework in the whole system meets the defined requirements. It builds a framework that ensures the known results and avoids unsurprising results. System testing depends on the procedure and its data streams, and does not require to have the knowledge of internal design of the logic or code. This testing is carried on a complete integrated system and falls in the category of white-box testing.
- **White Box testing.** White Box Testing is one of the most commonly used testing methodologies that is used to test the inner structures and just ensures that the application is working irrespective of the functionality. It is also termed as glass box test, clear box testing, structural testing, or transparent box testing. This testing is usually performed when the application is in the initial phase of development.
- **Black Box testing.** Black Box Testing is a type of testing, which operates exactly opposite to white box testing. This testing ensures the functionality of the application without concerning the internal structures or working of any application. This testing can be performed at any level of testing; for instance,

unit testing, system testing, integration testing, acceptance testing etc. Black Box tests can contain a source archive; for example, prerequisites report like requirements record or determination record. In this testing, the component under test deals with the discovery that depicts you cannot “see” into it. The test results give useful data on sources and responds to functionality without considering how the component works.

- **Acceptance testing.** Acceptance testing is performed after all the other testing is completed on a software component. User Acceptance Testing is an important stage of any project and needs significant end user involvement or participation. This testing ensures that the system developed meets the functional and design requirements after the application is ready to go live.

### **Test Results**

All the test cases in a typical acceptance testing should be passed successfully without encountering any defects.

### **Summary**

The implementation of different cryptographic algorithms and the test-bed for analyzing encryption processing overhead of different encryption algorithms is discussed in this chapter. The technologies, system designs and testing strategies used to develop the tool are also illustrated. The method to run the developed tool and the analysis of results is described in Chapter IV.



## Chapter IV: Analysis of Results

### Introduction

To run the project “Design of Data Encryption Test-Bed used to Analyze Encryption Processing Overhead,” one need not to have an in-depth knowledge of JAVA programming knowledge. The user interface designed in the project is quite user friendly and self-explanatory. By using few simple steps the task required can be accomplished.

### Running the Data Encryption Test Bed Project

**Step 1:** Navigate to MainPanel.java present in ‘com.encryption’ package of ‘EncryptionAlgorithm’ Java Project. Right Click on the Java file and click on “Run File” which opens the Main project window.

Once we run the project, the Java classes corresponding to the user interface display get compiled using the Java compiler and Java executer executes the Java code developed using Swing components for displaying the graphical user interface (GUI). After the projects run without any errors, the main window is displayed to the user. The steps of running the Java project in net beans 8.2 is illustrated in Figure 22.

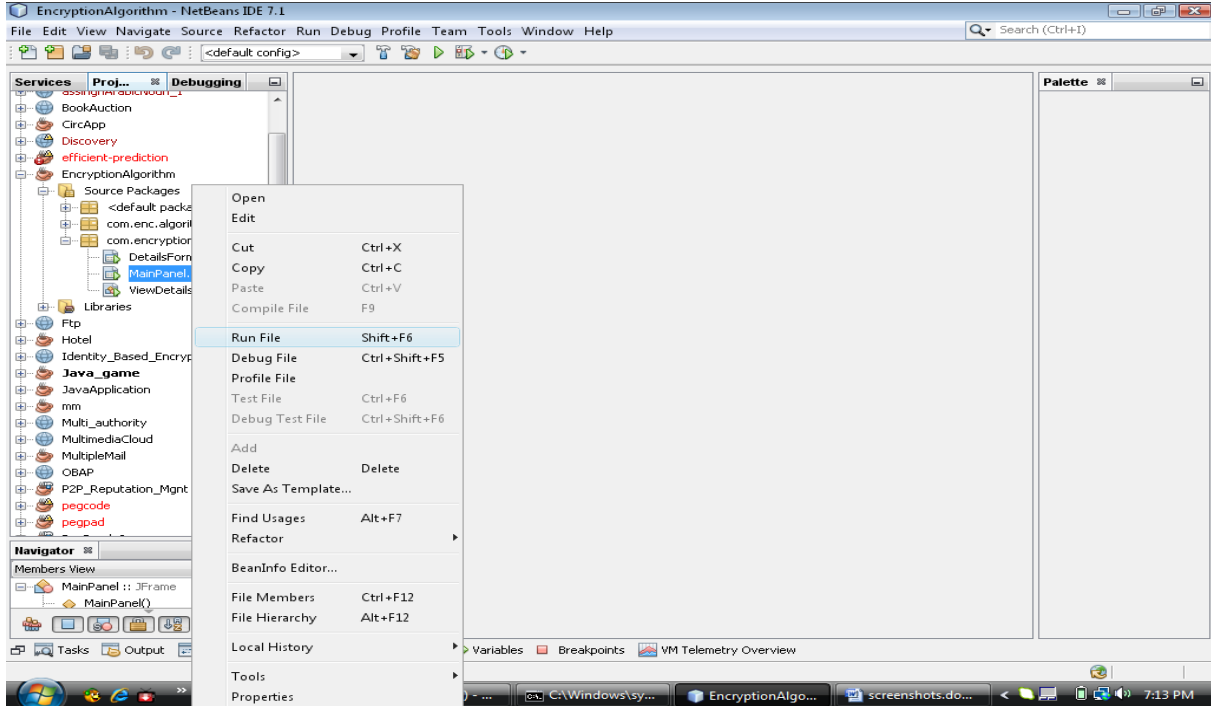


Figure 22. Running Project for Main Window.

**Step 2:** After the project is successfully executed, the Main Panel Window with header “Design of a Data Encryption Test-Bed used to Analyze Encryption Processing Overhead” is opened as illustrated in Figure 23.

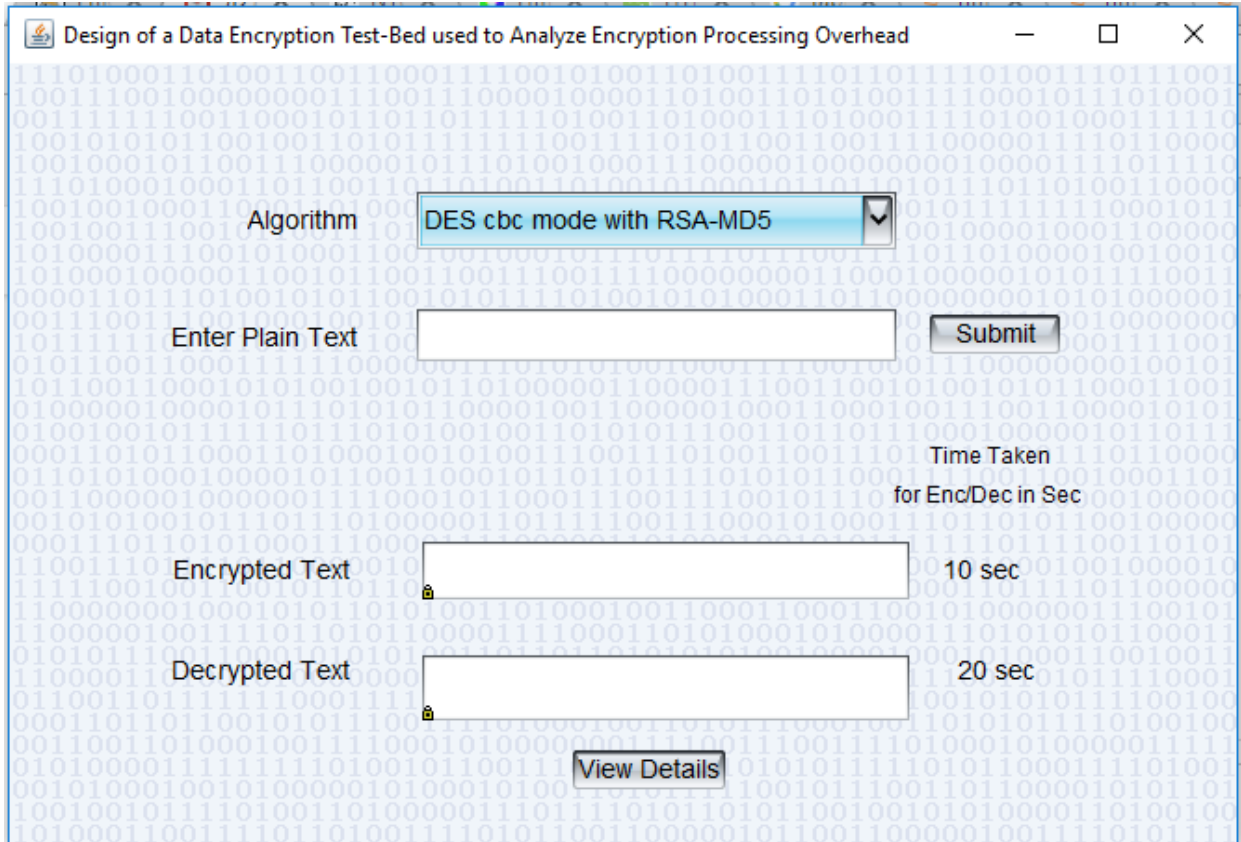


Figure 23. Main Window for Data Encryption Test-Bed project

This main user interface has different elements displayed. It has a label ‘Algorithm’ with an associated drop down displaying the list of available encryption and decryption algorithms. It has a label ‘Enter Plain Text’ with an associated textbox that facilitates the user to enter the input text. It has a Submit button. It also has two labels, namely “Encrypted Text” and “Decrypted Text” that displays the encrypted and decrypted text messages respectively in the associated text boxes. It also shows the time taken for encryption or decryption in seconds next to the textboxes.

**Step 3:** The user intending to encrypt or decrypt an input message and do a performance analysis of encryption or decryption algorithms can use the “Design of a Data Encryption Test-Bed used to Analyze Encryption Processing Overhead” Main project window.

Initially, the user need to select the encryption and decryption algorithm from the drop down associated with the label “Algorithm” as shown in Figure 24.

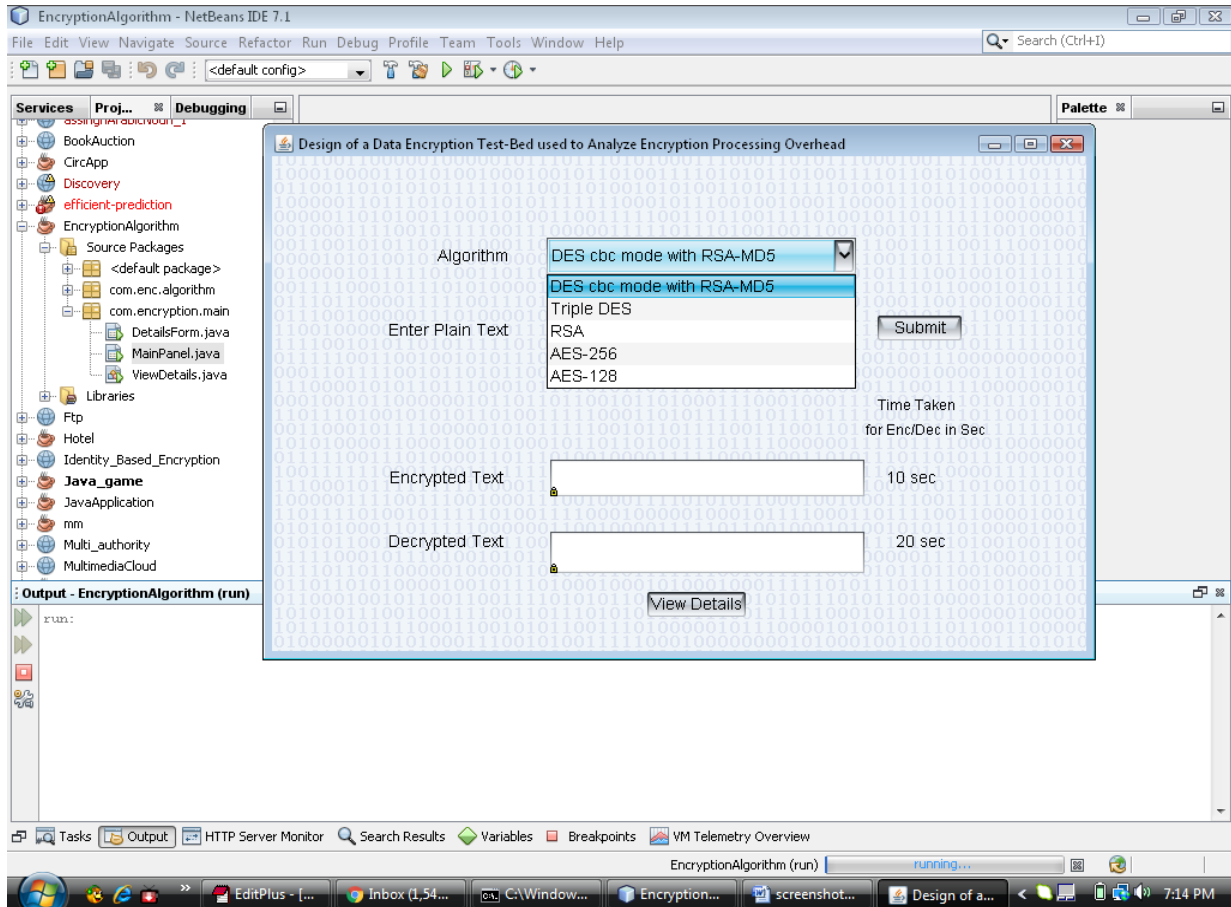


Figure 24. Selecting Algorithm for Data Encryption and Decryption.

The drop-down values on the user interface has the below values:

- DES CBC mode with RSA-MD5
- Triple DES
- RSA
- AES-256
- AES-128

**Step 4:** After selecting the encryption/decryption algorithm, he/she needs to enter the input text message that should be encrypted and decrypted in the text box associated with the label “Enter Plain Text.” To execute the selected encryption/decryption algorithm against the enter plain text, the user need to click on “Submit” button as shown in Figure 25.

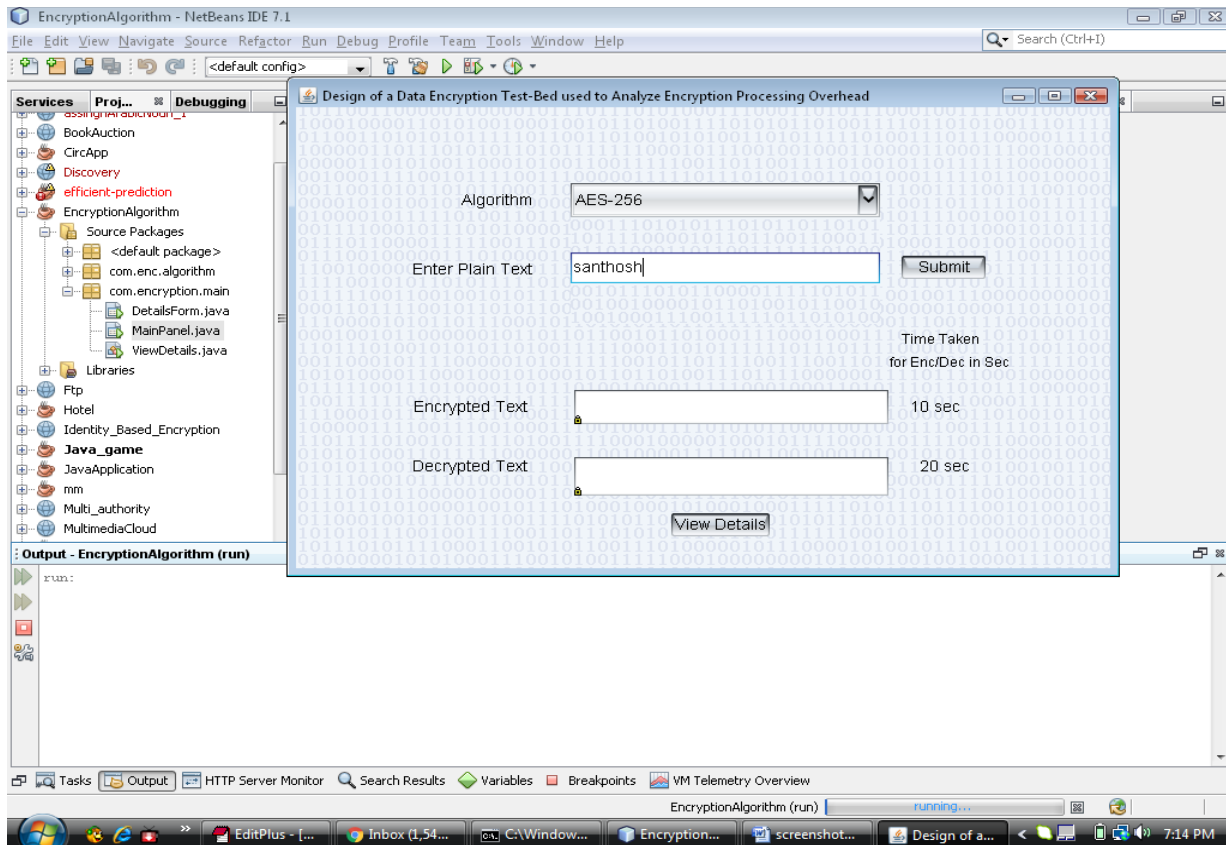


Figure 25. Entering Plain Text to Encrypt and Decrypt.

**Step 5:** After submitting, all the algorithms listed in the drop down are executed for the given plain text. The plain text is encrypted and decrypted with the respective algorithms. The time taken for encrypting and decrypting the given plain text is calculated. We will get the pop up displaying the best suitable algorithm with least time taken for encryption/decryption of the given plain text message from amongst the executed algorithms as illustrated in Figure 26.

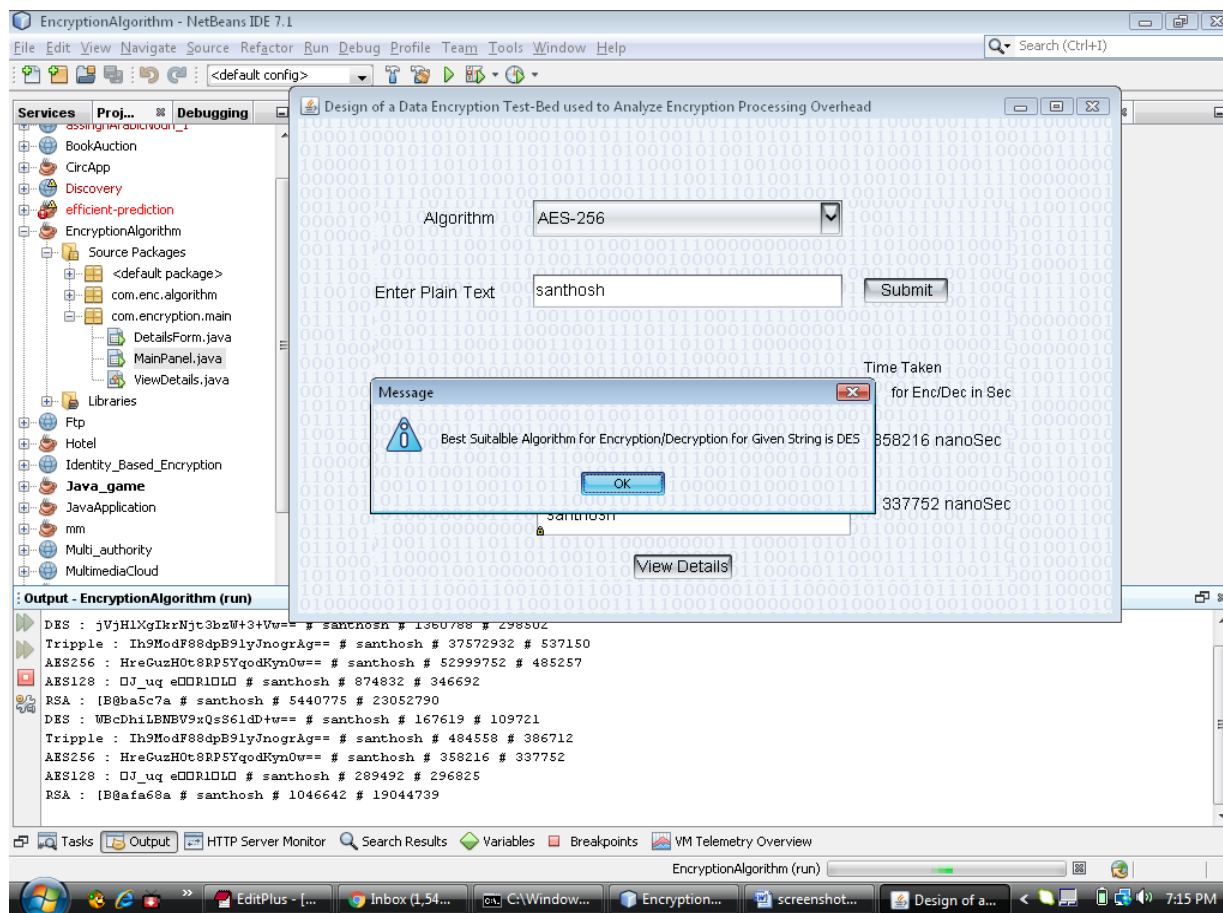


Figure 26. Displaying the Best Algorithm for the Given Plain Text.

**Step 6:** After clicking ‘Ok’ button on the pop-up, the encrypted text is displayed in the textbox “Encrypted Text” and the decrypted text is displayed in the textbox “Decrypted Text.” The time taken to encrypt and decrypt the given plain text with the selected encryption or decrypted algorithm is also displayed with a label “Time Taken for Enc/Dec in Sec” in seconds (see Figure 27).

Design of a Data Encryption Test-Bed used to Analyze Encryption Processing Overhead

Algorithm: DES cbc mode with RSA-MD5

Enter Plain Text: 0000000000000000000000000000

Submit

Time Taken for Enc/Dec in Sec

Encrypted Text: d5kNofPX7DV3mQ2h89c94mzG45ZxM= 161755 nanoSec

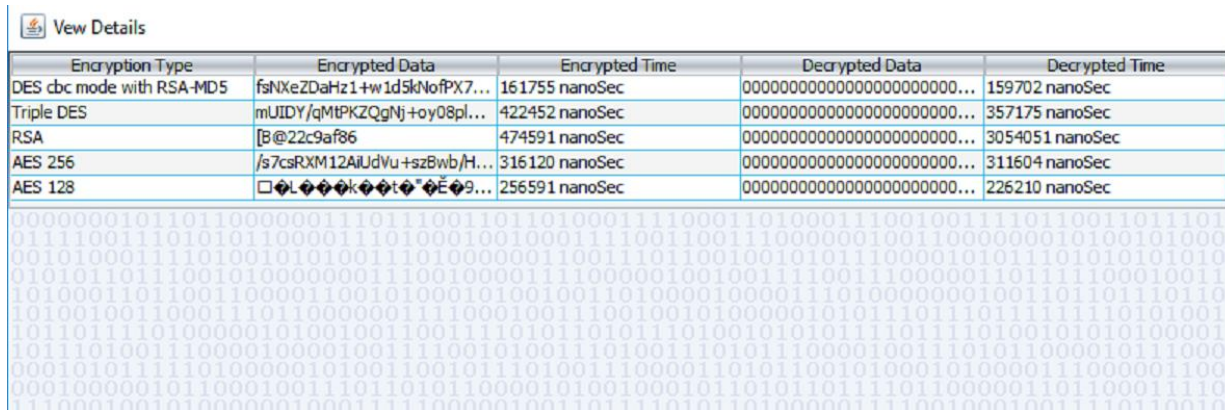
Decrypted Text: 0000000000000000000000000000 159702 nanoSec

View Details

Figure 27. Encrypted and Decrypted Text.

**Step 7:** Now if want to view the performance analysis of all the executed encryption and decryption algorithms against the given input plain text, we need to click on the button “View Details.” A new window is popped up with the header ‘View Details.’ It displays Encrypted Type, Encrypted Data, Encrypted Time, Decrypted Data, and Decrypted Time for all the encryption and decryption algorithms that ran for the given plain text. The Encrypted Type displays the encryption or decryption algorithm that is executed for the given plain text. The ‘Encrypted Data’ displays the data that is encrypted using the encryption or decryption algorithm that is displayed in Encryption Type columns. The ‘Encrypted Time’ displays the time taken to encrypt the input plain text for the encryption or decryption algorithm that is displayed in Encryption Type column in nanoseconds. Similarly, the ‘Decrypted Data’ displays the data that

is decrypted using the encryption or decryption algorithm that is displayed in Encryption Type columns. The ‘Decrypted Time’ displays the time taken to decrypt the input plain text for the encryption or decryption algorithm that is displayed in Encryption Type column in nanoseconds (see Figure 28).



Encryption Type	Encrypted Data	Encrypted Time	Decrypted Data	Decrypted Time
DES cbc mode with RSA-MD5	fsNXeZDaHz1+w1d5kNoFpX7...	161755 nanoSec	000000000000000000000000...	159702 nanoSec
Triple DES	mUIDY/qMfPKZQgNj+oy08pl...	422452 nanoSec	000000000000000000000000...	357175 nanoSec
RSA	[B@22c9af86	474591 nanoSec	000000000000000000000000...	3054051 nanoSec
AES 256	/s7csRXM12AidVu+szBwb/H...	316120 nanoSec	000000000000000000000000...	311604 nanoSec
AES 128	□L@k~*E@9...	256591 nanoSec	000000000000000000000000...	226210 nanoSec

Figure 28. Encryption and Decryption Details.

**Step 8:** The user can also view the encryption and decryption details of the given input plain text as a performance chart as well, which gives the pictorial representation of the encryption and decryption performance analysis (see Figure 29).



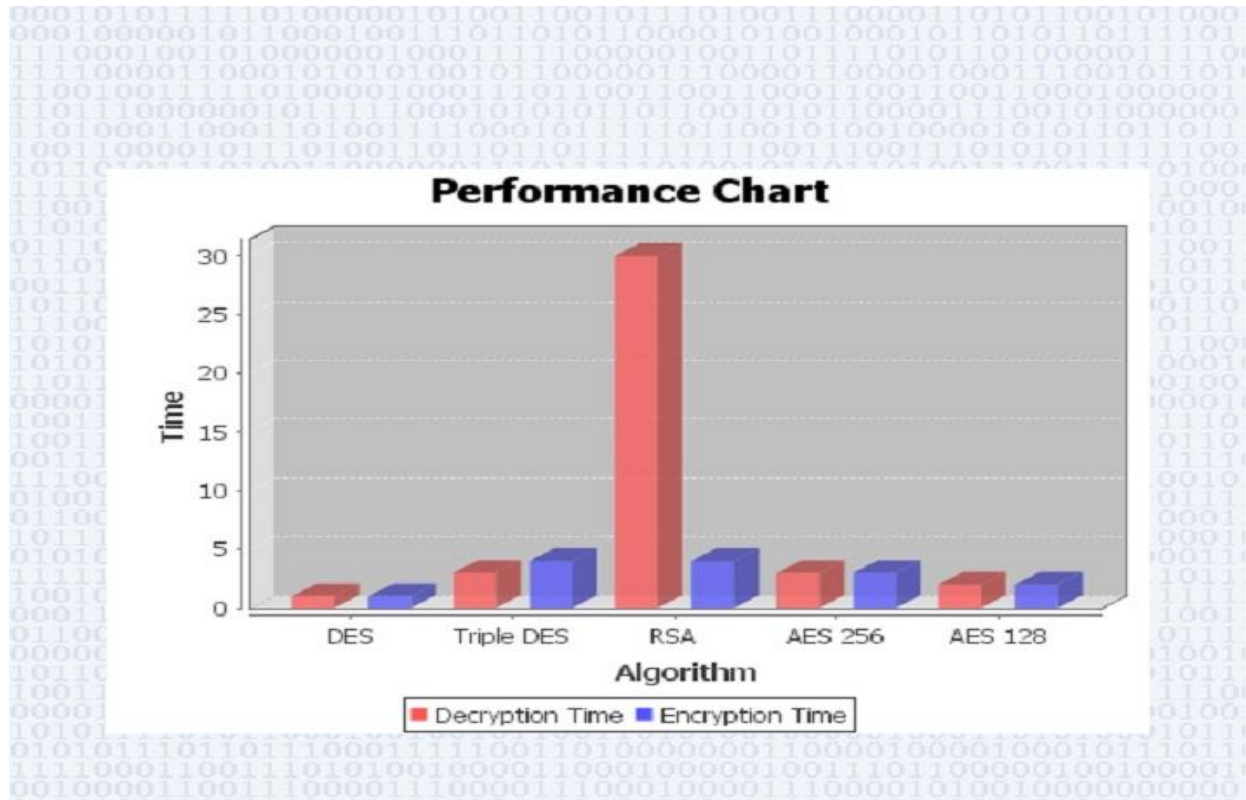


Figure 29. Graphical Chart Representation of Encryption/Decryption Algorithms.

In the above performance chart (Figure 29), the X-axis displays the encryption and decryption algorithms that are executed for the given input plain text. The Y-axis displays the time taken to execute the encryption and decryption for the given input plain text in nanoseconds. The bars with red color represent the Decryption Time and the ones with blue color represent the Encryption Time respectively.

### Hardware and Software Requirements

- Net beans IDE 8.2 for easy programming
- JAVA 1.7 JDK software installed
- Windows/Linux/Mac O.S

**Summary**

Various steps to run the Test-Bed tool are illustrated in this Chapter. The results of the execution of project are also analyzed. The possible enhancements of the project are discussed and the study is concluded in Chapter V.

## **Chapter VI: Conclusion and Future Study**

### **Introduction**

The study proposed and implemented in this paper is precisely supported in this chapter. The achievements of the study are described and conclusion of the study is made here. The possible enhancements or recommendations to the study are also discussed.

### **Conclusion**

In conclusion, this research paper helps the business organizations to choose the best reliable encryption and decryption algorithms for their organizations using the data encryption tool developed without any additional budget required for analysis.

This paper gives a high-level overview of a few of the most commonly used encryption/decryption algorithms like DES, Triple DES, RSA, AES128, and AES256 and illustrates the implementation of a test-bed for analyzing the encryption procession overhead of different cryptographic algorithms using a simple GUI (Graphical User Interface). A user-friendly interface is developed that allows any user to give any kind of input plaintext and execute it against a set of available encryption and decryption algorithms. Every minute detail starting from the requirements gathering to the implementation of the project has been presented in the paper in a theoretical and practical way. The primary focus is on clearly presenting the time taken for encrypting and decrypting the input text using which organizations can easily analyze the performance of different algorithms. For ease of understanding, the project also gives a graphical representation of the performance analysis.

**Future Work**

All encryption algorithms mentioned in this research paper are studied in the due course and completed the implementation. The study can be extended by including other encryption and decryption algorithms in the project that might gain major attention in the future. The tool can be enhanced by adding an option on the GUI like a browse button, etc. to upload a file which can hold huge amount of data.

## References

- Abdel, K. A. T. (2006). *Performance analysis of data encryption algorithms*. Retrieved from [http://www.cse.wustl.edu/~jain/cse567-06/ftp/encryption\\_perf.pdf](http://www.cse.wustl.edu/~jain/cse567-06/ftp/encryption_perf.pdf)
- Abdul, E., Kader, A., & Mohie, H. (2008). Performance evaluation of symmetric encryption algorithms. *IJCSNS International Journal of Computer Science and Network Security*, 8(12), 280-286.
- Abdul, M., Kader, D. S., Abdul, H. M., & Hadhoud, M. M. (2001). Analysis of performance for symmetric cryptography. *Communications of the IBIMA*, 8, 1943-7765.
- Agrawal, M., & Pradeep, M. (2012). *A comparative survey on symmetric key encryption techniques*. Retrieved from <http://www.enggjournals.com/ijcse/doc/IJCSE12-04-05-237.pdf>
- Ajay, K., Singh, M. L., & Bansal, P. K. (2012). Comparison of various encryption algorithms and techniques for secured data communication in multi-node network, *IJET International Journal of Engineering and Technology*, 2(1), 87-92.
- Akash, M., Chandra, P., & Archana, T. (2012). Performance evaluation of cryptographic algorithms: DES and AES. *IEEE Students' Conference on Electrical, Electronics and Computer Science*, 1-5.
- Alanazi, O., Zaidan, B. B., Zaidan, A. A., Jalab, A., Shabbir, M., & Al-Nabhani, Y. (2010). New comparative study between DES, 3DES and AES within nine factors. *Journal of Computing*, 2(3), 152-157.

- Anush, K. (2000). *Performance impact of encryption algorithms on Kerberos network Authentication protocol* (Unpublished Master's thesis). Oklahoma State University, Stillwater, OK.
- Byrne, M. (2015, October 18). *Know your language: The slow flickering Star-Death of Java (Part 1)*. Retrieved from <http://motherboard.vice.com/read/know-your-language-the-slow-flickering-star-death-of-java-part-one>
- Denning, D. E. R. (1982). *Cryptography and data security*. Boston, MA: Purdue University, Addison-Wesley Publishing Company. Retrieved from <http://faculty.nps.edu/dedennin/publications/Denning-CryptographyDataSecurity.pdf>
- Dinesh, T. (2013). *MD5–What is Message Digest 5 (MD 5)?* Retrieved from <http://ecomputernotes.com/computernetworkingnotes/security/md5>
- Drew, H. (2014). *How to encrypt your business data for optimal security*. Retrieved from <http://www.forbes.com/sites/drewhendricks/2014/09/30/how-to-encrypt-your-business-data-for-optimal-security/#2dad10db72c0>
- Gerasimos, K. (2012). *Crypto for pentesters*. Retrieved from <http://securityhorror.blogspot.com/2012/11/crypto-for-pentesters.html>
- Gurpreet, S., & Supriya. (2013). A study of encryption algorithms (RSA, DES, 3DES and AES) for Information Security, *International Journal of Computer Applications* (0975–8887), 67(19). Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.5601&rep=rep1&type=pdf>
- GURU. (2013). *Creating GUI components using java swing tutorial*. Retrieved from <http://www.guru99.com/java-swing-gui.html#1>

- Key schedule. (n.d.). *Wikipedia*. Retrieved from [https://en.wikipedia.org/wiki/Key\\_schedule](https://en.wikipedia.org/wiki/Key_schedule)
- Kopplin, J. (2002). *An illustrated history of computers, Part 3*. Retrieved from <http://www.computersciencelab.com/ComputerHistory/HistoryPt3.htm>
- Korde, V. (2016). The importance of a proper HTTP Strict transport security implementation on your web server. *Security Labs*. Retrieved from <https://blog.qualys.com/securitylabs/2016/03/28/the-importance-of-a-proper-http-strict-transport-security-implementation-on-your-web-server>
- Kurt, O., Nate, C., & Parker, H. (2013). *Which companies are encrypting your data properly?* Retrieved from <http://gizmodo.com/which-companies-are-encrypting-your-data-properly-1468088449>
- Lucian, C. (2016). Google, Microsoft, Yahoo and others publish new email Security standard. *IDS News service*. Retrieved from <http://www.pcworld.com/article/3046484/security/google-microsoft-yahoo-and-others-publish-new-email-security-standard.html>
- Mahajan, P., & Sachdeva, A. (2013). A study of encryption algorithms AES, DES and RSA for security. *Global Journal of Computer Science and Technology Network, Web and Security*, 13(16), Version 1.0. Retrieved from [https://globaljournals.org/GJCST\\_Volume13/4-A-Study-of-Encryption-Algorithms.pdf](https://globaljournals.org/GJCST_Volume13/4-A-Study-of-Encryption-Algorithms.pdf).
- Murat, K. (2007). *Modes of operation*. Retrieved from [http://www.utdallas.edu/~muratk/courses/crypto07\\_files/modes.pdf](http://www.utdallas.edu/~muratk/courses/crypto07_files/modes.pdf)

- National Academy of Public Administration. (2015). *Increasing the effectiveness of the federal role in cybersecurity education*. Retrieved from [http://napawash.org/images/reports/2015/Increasing\\_Effectiveness\\_of\\_Federal\\_Role\\_in\\_Cyber\\_Education.pdf](http://napawash.org/images/reports/2015/Increasing_Effectiveness_of_Federal_Role_in_Cyber_Education.pdf)
- National Institute of Standards and Technology. (1977). *Data encryption standard (DES)*. FIPS Publication 46. Washington, DC: Author.
- Pavithra, S., & Ramadevi, E. (2012). Performance evaluation of symmetric algorithms. *Journal of Global Research in Computer Science*, 3(8), 43- 45.
- Pluke, E. (2012). MD5. Retrieved from <https://en.wikipedia.org/wiki/MD5>
- Pranav, A. (2005). *Short tutorial on advanced encryption standard*. Retrieved from <http://pranav-mnit.tripod.com/aes.htm>
- Priyanka, A., Arun, S., & Himanshu, T. (2012). Evaluation and comparison of security issues on cloud computing environment. *World of Computer Science and Information Technology Journal (WCSIT)*, 2(5), 179-183.
- Rouse, M. (2007). *Cipher block chaining (CBC)*. Retrieved from <http://searchsecurity.techtarget.com/definition/cipher-block-chaining>
- Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., & Ferguson, N. (1998). Comparison of performance of the submissions of AES. *Counterpane Systems*. Retrieved from <http://www.counterpane.com/AESperformance.html>
- Shankar, G., & Jenifer, N. (2013). FPGA based SCA resistant AES S-Box Design. *International Journal of Scientific and Engineering Research*, 4(4), 1143-1149.



- Shashi, S., & Rajan, I. (2011). Comparative analysis of encryption algorithms for data communication. *International Journal of Computer Science and Technology*, 2(2), 192-294.
- Simar, S., & Raman, M. (2011). Comparison of data encryption algorithms. *International Journal of Computer Science and Communication*, 2(1), pp. 125-127.
- Simon, S. (2012). *What is java virtual machine and how does it work?* Retrieved from <http://www.makeuseof.com/tag/java-virtual-machine-work-makeuseof-explains/>
- Singh, L., & Bharti, R. K. (2013). Comparative analysis of cryptographic algorithms. *International Journal of Advanced Engineering Technology*. Retrieved from <http://www.technicaljournalsonline.com/ijeat/VOL%20IV/IJAET%20VOL%20IV%20IS SUE%20III%20JULY%20SEPTEMBER%202013/Vol%20IV%20Issue%20III%20Article%205.pdf%20Article%205.pdf>
- Thambiraja, E., Ramesh, G., & Umarani, R. (2012). A survey on various most common encryption techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(7), 226-233.
- Whitfield, D., & Susan, L. (2007). *Privacy on the line. The politics of wiretapping and encryption (Updated and expanded edition)*. Retrieved from [https://mitpress.mit.edu/sites/default/files/titles/free\\_download/9780262042406\\_Privacy\\_On\\_The\\_Line.pdf](https://mitpress.mit.edu/sites/default/files/titles/free_download/9780262042406_Privacy_On_The_Line.pdf)
- Wikiversity. (n.d.). *Cryptography/data encryption standard*. Retrieved from [https://en.wikiversity.org/wiki/Cryptography/Data\\_Encryption\\_Standard](https://en.wikiversity.org/wiki/Cryptography/Data_Encryption_Standard)

Yogesh, K., & Munjal, R. (2011). Comparison of symmetric and asymmetric cryptography with previous vulnerabilities. *IJCSMS International Journal of Computer Science and Management Studies*, 11(3), 60-63.

## Appendix

### The Encryption Algorithms Project

The 'Encryption Algorithms' JAVA research project includes a number of java Classes and jar files for designing a test bed to analyze encryption processing overhead.

#### DES Algorithm

```
package com.enc.algorithm;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

/**
 * DES encryption
 */

public class DataEncryptionStandardAlg {

    Cipher encryptionCipr;

    Cipher decryptionCipr;

    public DataEncryptionStandardAlg() throws Exception {

        SecretKey scrtKey = KeyGenerator.getInstance("DES").generateKey();

        encryptionCipr = Cipher.getInstance("DES");

        decryptionCipr = Cipher.getInstance("DES");

        encryptionCipr.init(Cipher.ENCRYPT_MODE, scrtKey);

        decryptionCipr.init(Cipher.DECRYPT_MODE, scrtKey);

    }

}
```

```
public String encryptTxt(String txt) throws Exception {  
    // Encode the string into bytes using utf-8  
    byte[] utf8bytes = txt.getBytes("UTF8");  
  
    // Encrypt  
    byte[] encBytes = encryptionCipr.doFinal(utf8bytes);  
  
    // Encode bytes to base64 to get a string  
    return new sun.misc.BASE64Encoder().encode(encBytes);  
}  
  
public String decryptTxt(String txt) throws Exception {  
    // Decode base64 to get bytes  
    byte[] decBytes = new sun.misc.BASE64Decoder().decodeBuffer(txt);  
  
    byte[] utf8Bytes = decryptionCipr.doFinal(decBytes);  
  
    // Decode using utf-8  
    return new String(utf8Bytes, "UTF8");  
}  
  
public static void main(String[] arguments) throws Exception {  
    DataEncryptionStandardAlg encryptAlg = new DataEncryptionStandardAlg();  
  
    String encryptedTxt = encryptAlg.encryptTxt("Don't tell anybody!");  
  
    String decryptedTxt = encryptAlg.decryptTxt(encryptedTxt);  
}  
}
```

**AES 128 Algorithm**

```
package com.enc.algorithm;

import javax.crypto.Cipher;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

/**
 * AES 128 encryption
 */

public class AES128Alg {

    static String InitializerVector = "AAAAAAAAAAAAAAAAAAAA";

    static String inputTxt = "test text 1234\0\0\0"; /* Note null padding */

    static String encrypKey = "0123456789abcdef";

    static int cal = 0;

    public static void main(String[] args) {

        try {

            byte[] cipherTxt = encryptFun(inputTxt);

            for (int i = 0; i < cipherTxt.length; i++)

                System.out.print(new Integer(cipherTxt[i]) + " ");

            String decryptedTxt = decryptFun(cipherTxt);

        } catch (Exception e) {
```

```

        e.printStackTrace();
    }
}

public static byte[] encryptFun(String inputTxt) throws Exception {
    if (inputTxt.getBytes().length % 16 != 0) {
        cal = 16 - inputTxt.getBytes().length % 16;
        for (int i = 0; i < cal; i++)
            inputTxt = inputTxt + " ";
    }

    Cipher ciprTxt = Cipher.getInstance("AES/CBC/NoPadding," "SunJCE");
    SecretKeySpec scrKey = new
SecretKeySpec(encrypKey.getBytes("UTF-
8"), "AES");
    ciprTxt.init(Cipher.ENCRYPT_MODE, scrKey, new IvParameterSpec(
    InitializerVector.getBytes("UTF-8")));
    return ciprTxt.doFinal(inputTxt.getBytes("UTF-8"));
}

public static String decryptFun(byte[] ciprTxt) throws Exception {
    Cipher ciphrTxt = Cipher.getInstance("AES/CBC/NoPadding,"
"SunJCE");
    SecretKeySpec scrKey = new
SecretKeySpec(encrypKey.getBytes("UTF-

```

```

        8”), “AES”);

        ciphrTxt.init(Cipher.DECRYPT_MODE, scrtKey, new IvParameterSpec(
        InitializerVector.getBytes(“UTF-8”)));

        return new String(ciphrTxt.doFinal(ciphrTxt), “UTF-8”).substring(0,
        new String(ciphrTxt.doFinal(ciphrTxt), “UTF-8”).length() - cal);

    }

}

```

### **AES 256 Algorithm**

```

package com.enc.algorithm;

import java.io.UnsupportedEncodingException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Arrays;

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;

/**
 * AES 256 encryption
 */

public class AES_256 {

    public static SecretKeySpec getScrtKey() {

        return scrtKey;
    }
}

```

```
}  
  
public static void setScrtKey(SecretKeySpec scrtKey) {  
    AES_256.scrtKey = scrtKey;  
}  
  
public static byte[] getByteKey() {  
    return byteKey;  
}  
  
public static void setByteKey(byte[] byteKey) {  
    AES_256.byteKey = byteKey;  
}  
  
public static String getDcryptdStr() {  
    return dcryptdStr;  
}  
  
public static void setDcryptdStr(String dcryptdStr) {  
    AES_256.dcryptdStr = dcryptdStr;  
}  
  
public static String getEncryptdStr() {  
    return encryptdStr;  
}  
  
public static void setEncryptdStr(String encryptdStr) {  
    AES_256.encryptdStr = encryptdStr;  
}
```



```
private static SecretKeySpec scrtKey;

private static byte[] byteKey;

private static String dcryptdStr;

private static String encryptdStr;

public static void setScrtKey(String myScrtKey) {
    MessageDigest mdSha = null;
    try {
        byteKey = myScrtKey.getBytes("UTF-8");
        mdSha = MessageDigest.getInstance("SHA-1");
        byteKey = mdSha.digest(byteKey);
        byteKey = Arrays.copyOf(byteKey, 16); // use only first 128 bit
        scrtKey = new SecretKeySpec(byteKey, "AES");
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

public static String encryptTxt(String txtToEncrypt) {
    try {
        Cipher ciphrTxt = Cipher.getInstance("AES/ECB/PKCS5Padding");
        ciphrTxt.init(Cipher.ENCRYPT_MODE, scrtKey);
```

```

        setEncryptedString(Base64.encodeBase64String(ciphrTxt
            .doFinal(txtToEncrypt.getBytes("UTF-8"))));
    } catch (Exception e) {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return encryptdStr;
}

public static String decryptTxt(String strToDecrypt) {
    try {
        Cipher cphr = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cphr.init(Cipher.DECRYPT_MODE, scrtKey);
        setDcryptdStr(new String(cphr.doFinal(Base64
            .decodeBase64(strToDecrypt))));
    } catch (Exception e) {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return dcryptdStr;
}

public static void main(String arguments[]) {
    final String txtToEncrypt = "My text to encrypt";
    final String pssword = "encryptor key";

```

```
        AES_256.setScrtKey(pssword);

        AES_256.encryptTxt(txtToEncrypt.trim());

        final String txtToDecrypt = AES_256.getEncryptedString();

        AES_256.decryptTxt(txtToDecrypt.trim());

    }
}
```

### **RSA Algorithm**

```
package com.enc.algorithm;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.NoSuchAlgorithmException;

import java.security.PrivateKey;

import java.security.PublicKey;

import javax.crypto.Cipher;

import javax.swing.JOptionPane;
```

```
/*RSA Algorithm*/  
  
public class RSAAlg {  
  
    public static final String RSA_ALGORITHM = "RSA";  
  
    public static final String PRVTE_KEY = "private.key";  
  
    public static final String PBLIC_KEY = "public.key";  
  
    /**  
  
    * Generate key which contains a pair of private and public key using 1024  
  
    * bytes. Store the set of keys in Prvate.key and Public.key files.  
  
    * @throws NoSuchAlgorithmException  
  
    * @throws IOException  
  
    * @throws FileNotFoundException  
  
    */  
  
    public static void keyGenratn() {  
  
        try {  
  
            final KeyPairGenerator keyGenratn = KeyPairGenerator  
                .getInstance(RSA_ALGORITHM);  
  
            keyGenratn.initialize(1024);  
  
            final KeyPair keyPr = keyGenratn.generateKeyPair();  
  
            File prvteKeyFile = new File(PRVTE_KEY);  
  
            File pblicKeyFile = new File(PBLIC_KEY);  
  
            // Create files to store public and private key  
  
            if (prvteKeyFile.getParentFile() != null) {
```

```
        prvteKeyFile.getParentFile().mkdirs();
    }
    prvteKeyFile.createNewFile();
    if (pblicKeyFile.getParentFile() != null) {
        pblicKeyFile.getParentFile().mkdirs();
    }
    pblicKeyFile.createNewFile();
    // Saving the Public key in a file
    ObjectOutputStream pblicKeyStream = new ObjectOutputStream(
        new FileOutputStream(pblicKeyFile));
    pblicKeyStream.writeObject(keyPr.getPublic());
    pblicKeyStream.close();
    // Saving the Private key in a file
    ObjectOutputStream prvteKeyStream = new ObjectOutputStream(
        new FileOutputStream(prvteKeyFile));
    prvteKeyStream.writeObject(keyPr.getPrivate());
    prvteKeyStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
/**
```

```

* The method checks if the pair of public and private key has been
* generated.
* @return flag indicating if the pair of keys were generated.
*/

```

```

public static boolean keyPresent() {
    File prvKey = new File(PRVTE_KEY);
    File pblicKey = new File(PBLIC_KEY);
    if (prvKey.exists() && pblicKey.exists()) {
        return true;
    }
    return false;
}

```

```

/**

```

```

* Encrypt the plain text using public key.
* @param txt: original plain text
* @param pblicKey: The public key
* @return Encrypted text
* @throws java.lang.Exception
*/

```

```

public static byte[] encryptTxt(String txt, PublicKey pblicKey) {
    byte[] ciprTxt = null;
    try {

```

```
        // get an RSA cipher object and print the provider
        final Cipher cphr = Cipher.getInstance(RSA_ALGORITHM);
        // encrypt the plain text using the public key
        cphr.init(Cipher.ENCRYPT_MODE, pblcKey);
        ciprTxt = cphr.doFinal(txt.getBytes());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return ciprTxt;
}

public static String decryptTxt(byte[] txt, PrivateKey prvtKey) {
    byte[] dcryptdTxt = null;
    try {
        // get an RSA cipher object and print the provider
        final Cipher ciphr = Cipher.getInstance(RSA_ALGORITHM);
        // decrypt the text using the private key
        ciphr.init(Cipher.DECRYPT_MODE, prvtKey);
        dcryptdTxt = ciphr.doFinal(txt);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return new String(dcryptdTxt);
}
```

```
}  
  
/**  
  
 * Test the EncryptionUtil  
  
 */  
  
public static void main(String[] args) {  
    try {  
        // Check if the pair of keys are present else generate those.  
        if (!keyPresent()) {  
            // Method generates a pair of keys using the RSA algorithm and  
            // stores it  
            // in their respective files  
            keyGenratn();  
        }  
        final String actualText = "Text to be encrypted ";  
        ObjectInputStream iStream = null;  
        // Encrypt the string using the public key  
        iStream = new ObjectInputStream(new  
        FileInputStream(PBLIC_KEY));  
        final PublicKey pblcKey = (PublicKey) iStream.readObject();  
        final byte[] cphrTxt = encryptTxt(actualText, pblcKey);  
        // Decrypt the cipher text using the private key.  
        iStream = new ObjectInputStream(new
```



```

        FileInputStream(PRVTE_KEY));

        final PrivateKey prvtKey = (PrivateKey) iStream.readObject();

        final String inputTxt = decryptTxt(cphrTxt, prvtKey);

        // Printing the Original, Encrypted and Decrypted Text

        System.out.println("Original Text: " + actualText);

        System.out.println("Encrypted Text: " + cphrTxt.toString());

        System.out.println("Decrypted Text: " + inputTxt);

    } catch (Exception e) {

        e.printStackTrace();

    }

}
}
}

```

### **Triple DES Algorithm**

```

package com.enc.algorithm;

import java.security.MessageDigest;

import java.util.Arrays;

import javax.crypto.Cipher;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;

/**

 * Simple TripleDES Encrypt/Decrypt Test sha1, utf-8, no padding

```

```
* uses commons-codec-1.6 javac -cp :commons-codec-1.6.jar TripleDESTest.java
* java -cp :commons-codec-1.6.jar TripleDESTest
*/
public class TripleDES {
    public static void main(String[] arguments) throws Exception {
        String txt = "textToEncrypt";
        String codedtxt = new TripleDES().encryptFunc(txt, "SecretKey");
        String decodedtxt = new TripleDES().decryptFunc(codedtxt,
            "SecretKey");
    }
    public String encryptFunc(String msg, String scrtKey) throws Exception {
        MessageDigest msgd = MessageDigest.getInstance("SHA-1");
        byte[] digestOfPass = msgd.digest(scrtKey.getBytes("utf-8"));
        byte[] kyBytes = Arrays.copyOf(digestOfPass, 24);
        SecretKey scrtK = new SecretKeySpec(kyBytes, "DESede");
        Cipher cipr = Cipher.getInstance("DESede");
        cipr.init(Cipher.ENCRYPT_MODE, scrtK);
        byte[] inputTxtBytes = msg.getBytes("utf-8");
        byte[] buffer = cipr.doFinal(inputTxtBytes);
        byte[] bse64Bytes = Base64.encodeBase64(buffer);
        String bse64EncryptString = new String(bse64Bytes);
        return bse64EncryptString;
    }
}
```

```
}  
  
public String decryptFunc(String encryptedTxt, String scrKey)  
    throws Exception {  
    byte[] msg = Base64.decodeBase64(encryptedTxt.getBytes("utf-8"));  
    MessageDigest msgd = MessageDigest.getInstance("SHA-1");  
    byte[] digestOfPaswd = msgd.digest(scrKey.getBytes("utf-8"));  
    byte[] kBytes = Arrays.copyOf(digestOfPaswd, 24);  
    SecretKey scrK = new SecretKeySpec(kBytes, "DESede");  
    Cipher dcipher = Cipher.getInstance("DESede");  
    dcipher.init(Cipher.DECRYPT_MODE, scrK);  
    byte[] inputTxt = dcipher.doFinal(msg);  
    return new String(inputTxt, "UTF-8");  
}  
}
```