A PROJECT REPORT ON

**Online Railway Parking System**

Made By

*Discipline*

CSE

IT

CSE

*Roll No.*

*06414802715*

*06414802715*

*06414802715*

*Name*

Nitigya Sharma

Mukul Taneja

Vansh Mittal

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE

**Industrial Training**

**(ETCS - 359)**

Developed At

**Centre for Railway Information Systems**   
(An Organization of the Ministry of Railways, Govt. of India)   
Chanakya Puri, New Delhi - 110021  
Phone: 24104525, 24106717   
 Fax: 26877893

Mr Baloo

(Project Supervisor)

**ACKNOWLEDGEMENT**

In completing my project, we are very thankful to many individuals and we must place on record our sincere thanks to all of them.

First of all, we would like to express our deep sense of gratitude to our supervisor Mr Baloo, Senior Project Manager who gave us his invaluable guidance glowing with his words of encouragement and inspiration, criticisms and discussions throughout the problem designing.

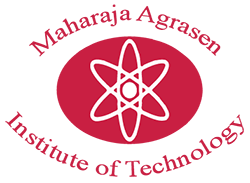
We are very much grateful to Dr. Namita Gupta (Professor & H.O.D. (CSE)) for his valuable support and cooperation in conceptualizing the project/research work and to all those outstanding individuals with whom we have worked, who helped us in understanding the concept.

We are highly thankful to our family members for their all-time support in initiating us and bringing a spark in us to pursue the work.

**Maharaja Agarsen Institute of Technology, Delhi**

Rohini, Delhi – 110086

**Department of Computer Engineering and Information Technology**



**CERTIFICATE**

*This is to certify that, this Project Report entitled*

**“Online Railway Parking System”**

*Is a bonafide record of Project work carried out by*

**Nitigya Sharma (06414802715)**

**Mukul Taneja (06414802715)**

**Vansh Mittal (06414802715)**

*As per partial fulfillment of the award of post graduate degree in*

*B.Tech. (Computer Engineering) during the academic year 2017-2018*

Date:

Place: Delhi

(Dr. Namita Gupta)

**Project Coordinator Head of Department**

**Declaration**

Certified that this project report **“**Online Railway Parking System**”** is the bonafide work of **Nitigya sharma (06414802715), Mukul Taneja()** and **Vansh Mittal ()** who carried out the project in collaboration with **Centre for Railways information System (CRIS)**, embodies the work done by him under the guidance of **Mr. Baloo**, **Senior Android Doveloper And Project Manager At CRIS** towards partial fulfilment of the requirements for the Degree of Bachelor of Technology in Computer Science and Engineering from **Maharaja Agrasen Institute of Technology, Delhi**. They have fulfilled all the requirements needed as per the rules of the University, for the completion of Project. This work is original and has not been submitted in part or in full to any other University or Institution other than mentioned obove.

**Nitigya Sharma**

**(Pre-final year Student)**

Computer Science and Engineering Department,

Maharaja Agrasen Institute of Technology, Rohini

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
|  | **Acknowledgement** | **i.** |
|  | **Certificate** | **ii.** |
|  | **Declaration** | **iii.** |
|  | **Preface** | **iv.** |
|  | **Organisation Introduction** | **v.** |
| **1.** | **CHAPTER 1: Introduction(app kisliye hai aur kya)** | 1 |
| 1.1 | Introduction | 2 |
| 1.1.1 | Advanced Encryption Standard (AES) |  |
| 1.1.2 | Data Encryption Standard (DES) |  |
| 1.1.3 | Cipher Block Chaining (CBC) |  |
| 1.1.4 | Cryptography |  |
| 1.2 | Objective | 2 |
|  |  |  |
| **2.** | **CHAPTER 2: Project Designing (algoritham and tech details)** | 6 |
| 2.1 | Steps of implementation |  |
| 2.2 | Testing | 10 |
| 2.3 | Encryption Testing |  |
|  |  |  |
| **3.** | **CHAPTER 3: Designing & Coding (codes and output)** | 15 |
| 3.1 | DES Coding | 18 |
| 3.1.1 | Output |  |
| 3.2 | AES Coding | 20 |
| 3.2.1 | Output |  |
|  |  |  |
| **4.** | **CHAPTER 4: Conclusion(teeno part ne kiya kya)** | 25 |
| 4.1 | Analysis | 28 |
| 4.1.1 | AES Analysis |  |
| 4.1.2 | DES Analysis |  |
| 4.1.3 | CBC Analysis |  |
|  | **References** | **vi.** |

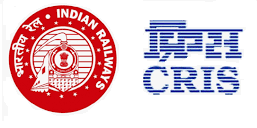
**Preface**

India is facing a new problem nowadays – lack of sufficient parking space. With families getting smaller and the total number of motor vehicles exceeding the total number of heads per family, the parking scenario is woefully falling short of the current requirements in the country. The situation is such that on any given working day approximately 40% of the roads in urban India are taken up for just parking the cars. The problem has been further exacerbated by the fact that nowadays even people from low income group are able to own cars. The number of families with cars has become much more than what the country is able to manage.

As it is, the cities in India are highly congested and on top of that the parked cars claim a lot of space that could otherwise be used in a better way. Thanks to poor, and at times zero, navigability, Indian cities are regarded as some of the worst options for living. One can also add the issue of pollution to this mix and understand the enormity of the crisis. In this context it needs to be understood that the Indian cities, with the possible exception of Chandigarh, were never planned in such a way so as to accommodate a deluge of cars as is the situation now. The apathy of present day urban planners has only made the situation worse.

That’s why there comes a need of efficient parking system that can accommodate cars and manage their existence at a particular work place where vehicle has to come and leave regularly

**Centre for Railways information System**

****

The Centre for Railway Information Systems (CRIS) is an Autonomous Organization under the Ministry of Railways. It develops and manages the Information Technology applications of the Indian Railways. CRIS also provides IT applications for non-Railway Government and Public Sector organizations.  
Our approach focuses on new ways of conducting business by combining IT innovation and adoption, while leveraging an organization's current IT assets. We work with the Indian Railways to conceive prudent technology strategies and build new services in today's dynamic digital environment.  
Our large pool of skilled personnel enables us to develop complex information systems, and our presence across the country gives us the reach to support large-scale rollouts.

**Introduction**

**PHP’s Hypertext Preprocessor (PHP)**

The Advanced Encryption Standard, or AES, is a symmetric [block cipher](http://searchsecurity.techtarget.com/definition/block-cipher) chosen by the U.S. government to protect classified information and is implemented in software and hardware throughout the world to encrypt sensitive data.

The [National Institute of Standards and Technology (NIST)](http://searchsoftwarequality.techtarget.com/definition/NIST) started development of AES in 1997 when it announced the need for a successor algorithm for the [Data Encryption Standard (DES)](http://searchsecurity.techtarget.com/definition/Data-Encryption-Standard), which was starting to become vulnerable to [brute-force attacks](http://searchsecurity.techtarget.com/definition/brute-force-cracking).

This new, advanced encryption [algorithm](http://whatis.techtarget.com/definition/algorithm) would be unclassified and had to be "capable of protecting sensitive government information well into the next century," according to the NIST announcement of the process for development of an advanced [encryption](http://searchsecurity.techtarget.com/definition/encryption) standard algorithm. It was intended to be easy to implement in hardware and software, as well as in restricted environments (for example, in a [smart card](http://searchsecurity.techtarget.com/definition/smart-card)) and offer good defenses against various attack techniques.

### **AES features -**

The selection process for this new [symmetric key algorithm](http://searchsecurity.techtarget.com/definition/secret-key-algorithm) was fully open to public scrutiny and comment; this ensured a thorough, transparent analysis of the designs submitted.

NIST specified the new advanced encryption standard algorithm must be a block cipher capable of handling 128-bit blocks, using keys sized at 128, 192, and 256 bits; other criteria for being chosen as the next advanced encryption standard algorithm include:

* **Security:** Competing algorithms were to be judged on their ability to resist attack, as compared to other submitted ciphers, though security strength was to be considered the most important factor in the competition.
* **Cost:** Intended to be released on a global, non-exclusive and royalty-free basis, the candidate algorithms were to be evaluated on computational and memory efficiency.
* **Implementation:** Algorithm and implementation characteristics to be evaluated included the flexibility of the algorithm; suitability of the algorithm to be implemented in hardware or software; and overall, the relative simplicity of implementation.

# Data Encryption Standard (DES)

The Data Encryption Standard (DES) is an outdated symmetric-key method of data [encryption](http://searchsecurity.techtarget.com/definition/encryption).

DES works by using the same [key](http://searchsecurity.techtarget.com/definition/key) to encrypt and decrypt a message, so both the sender and the receiver must know and use the same [private key](http://searchsecurity.techtarget.com/definition/private-key). Once the go-to, symmetric-key algorithm for the encryption of electronic data, DES has been superseded by the more secure [Advanced Encryption Standard](http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard) (AES) algorithm.

Originally designed by researchers at [IBM](http://searchitchannel.techtarget.com/definition/IBM-International-Business-Machines) in the early 1970s, DES was adopted by the U.S. government as an official Federal Information Processing Standard ([FIPS](http://whatis.techtarget.com/definition/FIPS-Federal-Information-Processing-Standards)) in 1977 for the encryption of commercial and sensitive yet unclassified government computer data. It was the first encryption [algorithm](http://whatis.techtarget.com/definition/algorithm) approved by the U.S. government for public disclosure. This ensured that DES was quickly adopted by industries such as financial services, where the need for strong encryption is high. The simplicity of DES also saw it used in a wide variety of embedded systems, [smart cards](http://searchsecurity.techtarget.com/definition/smart-card), [SIM cards](http://searchmobilecomputing.techtarget.com/definition/SIM-card) and network devices requiring encryption like [modems](http://searchmobilecomputing.techtarget.com/definition/modem), [set-top boxes](http://searchnetworking.techtarget.com/definition/set-top-box), and [routers](http://searchnetworking.techtarget.com/definition/router).

# Cipher Block Chaining (CBC)

Cipher block chaining (CBC) is a mode of operation for a [block cipher](http://searchsecurity.techtarget.com/definition/block-cipher) (one in which a sequence of bits are encrypted as a single unit or block with a [cipher](http://searchsecurity.techtarget.com/definition/cipher) [key](http://searchsecurity.techtarget.com/definition/key) applied to the entire block). Cipher block chaining uses what is known as an initialization vector (IV) of a certain length. One of its key characteristics is that it uses a chaining mechanism that causes the [decryption](http://searchsecurity.techtarget.com/definition/encryption) of a block of [ciphertext](http://searchcio-midmarket.techtarget.com/definition/ciphertext) to depend on all the preceding ciphertext blocks. As a result, the entire validity of all preceding blocks is contained in the immediately previous ciphertext block. A single bit error in a ciphertext block affects the decryption of all subsequent blocks. Rearrangement of the order of the ciphertext blocks causes decryption to become corrupted. Basically, in cipher block chaining, each plaintext block is XORed (see [XOR](http://whatis.techtarget.com/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR)) with the immediately previous ciphertext block and then encrypted.

Identical ciphertext blocks can only result if the same plaintext block is encrypted using both the same key and the initialization vector, and if the ciphertext block order is not changed. It has the advantage over the [Electronic Code Book](http://searchsecurity.techtarget.com/definition/Electronic-Code-Book) mode in that the XOR'ing process hides plaintext patterns.

Ideally, the initialization vector should be different for any two messages encrypted with the same key. Though the initialization vector need not be secret, some applications may find this desirable.

# Cryptography

## **What is Cryptography?**

Cryptography is the science to encrypt and decrypt data that enables the users to store sensitive information or transmit it across insecure networks so that it can be read only by the intended recipient.

Data which can be read and understood without any special measures is called plaintext while the method of disguising plaintext in in order to hide its substance is called encryption.

The encrypted plain text is known as ciphertext and process of reverting the encrypted data back to plain text is known as decryption.

* The science of analyzing and breaking secure communication is known as cryptanalysis. The people who perform the same also known as attackers.
* Cryptography can be either strong or weak and the strength is measured by the time and resources it would require recovering the actual plaintext.
* Hence appropriate decoding tool is required to decipher strongly encrypted messages.
* There are some cryptographic techniques available with which even a billion computers doing a billion checks a second, it is not possible to decipher the text. As the power of computing increases day by day, one has to make their encryption algorithm very strong in order to protect it from the attackers.

## **How Encryption Works**

A cryptographic algorithm works in combination with a key(can be a word, number, or phrase) to encrypt the plaintext and the same plaintext encrypts to different ciphertext with different keys.

Hence, the encrypted data is completely dependent couple of parameters viz- the strength of the cryptographic algorithm and the secrecy of the key.

## **Cryptography Techniques**

**Symmetric Encryption** - conventional cryptography, also known as Conventional encryption in which one key is used both for encryption and decryption. Eg: DES, Triple DES algorithms, MARS by IBM, RC2, RC4, RC5,RC6.

**Asymmetric Encryption** - It is Public key cryptography that uses a pair of keys for encryption: a public key, which encrypts data, and a private key used for decryption. The public key is published to the people while keeping the private key secret. Eg: RSA, Digital Signature Algorithm (DSA), Elgamal

**Hashing** - Hashing is ONE- way encryption, which the scrambled output that cannot be reversed or at least cannot be reversed easily that is used to validate the integrity of information. Eg: MD5 algorithm. It is used to create Digital Certificates, Digital signatures, Storage of passwords, Verification of communications.

# Objectives

Cryptographic primitives can be divided into two large classes. First, secret-key cryptography (or symmetric cryptography) uses the same key for both decryption and encryption. This key is a shared secret between two or more parties that can be used to maintain a private information link. In secret-key cryptography. The most studied and widely used class of symmetric cryptographic algorithms are block ciphers. Block ciphers are mathematical functions that take as input a plaintext and a secret key. By performing highly complex scrambling operations, a block cipher outputs an encrypted version of the input text, known as ciphertext.

On the other hand, public-key cryptography (or asymmetric cryptography); in that case, the key used to encrypt a message differs from the key used to decrypt it. The private key is kept secret, while the public key is widely distributed. For this class of algorithms, RSA (Rivest, Shamir, and Adleman) is the most widely known and deployed. RSA uses exponentiation modulo a product of two large primes to encrypt and decrypt, performing both public key encryption and public key digital signature. Its security is related to the difficulty of factoring large integers, a problem for which there is no known efficient general technique. A relaxation of this factorization problem arises when some additional information is provided. The problem is then to factor large integers knowing some bits of the prime factors.

# Encryption/Decryption Modules

The Encryption/decryption module provides encryption/decryption functions :-

One can differentiate between symmetric and asymmetric algorithms; the symmetric ones are mostly used for message confidentiality and the asymmetric ones for key exchange and message integrity. Some symmetric algorithms provide different block cipher modes, mainly Electronic Code Book (ECB) which is used for short (64-bit) messages and Cipher Block Chaining (CBC) which provides the structure needed for longer messages. In addition, the Cipher Feedback Mode (CFB-128) stream cipher mode, Counter mode (CTR) and Galois Counter Mode (GCM) are implemented for specific algorithms.

All symmetric encryption algorithms are accessible via the generic cipher layer (see [mbedtls\_cipher\_setup()](https://tls.mbed.org/api/cipher_8h.html#a009056b59d69abba5843ce78cd9aae1c)).

The asymmetric encryption algorithms are accessible via the generic public key layer (see [mbedtls\_pk\_init()](https://tls.mbed.org/api/pk_8h.html#a999d1160bb30c03d0c4382c3a9b0aa89)).

The following algorithms are provided:

* **Symmetric –**
  + AES (see mbedtls\_aes\_crypt\_ecb(), mbedtls\_aes\_crypt\_cbc(), mbedtls\_aes\_crypt\_cfb128() and mbedtls\_aes\_crypt\_ctr()).
  + ARCFOUR (see mbedtls\_arc4\_crypt()).
  + Blowfish / BF (see mbedtls\_blowfish\_crypt\_ecb(), mbedtls\_blowfish\_crypt\_cbc(), mbedtls\_blowfish\_crypt\_cfb64() and mbedtls\_blowfish\_crypt\_ctr())
  + Camellia (see mbedtls\_camellia\_crypt\_ecb(), mbedtls\_camellia\_crypt\_cbc(), mbedtls\_camellia\_crypt\_cfb128() and mbedtls\_camellia\_crypt\_ctr()).
  + DES/3DES (see mbedtls\_des\_crypt\_ecb(), mbedtls\_des\_crypt\_cbc(), mbedtls\_des3\_crypt\_ecb() and mbedtls\_des3\_crypt\_cbc()).
  + GCM (AES-GCM and CAMELLIA-GCM) (see [mbedtls\_gcm\_init()](https://tls.mbed.org/api/gcm_8h.html#ac3f60a663c6b01ef6d977ac06aac57df))
  + XTEA (see mbedtls\_xtea\_crypt\_ecb()).
* **Asymmetric –** 
  + Diffie-Hellman-Merkle (see [mbedtls\_dhm\_read\_public()](https://tls.mbed.org/api/dhm_8h.html#a1fe799e776fd0f27eb4080248e547f3d), [mbedtls\_dhm\_make\_public()](https://tls.mbed.org/api/dhm_8h.html#af49bee438130355e24540f7565e7465f) and [mbedtls\_dhm\_calc\_secret()](https://tls.mbed.org/api/dhm_8h.html#ac3985de01420d018ed91daec9e7d7969)).
  + RSA (see [mbedtls\_rsa\_public()](https://tls.mbed.org/api/rsa_8h.html#a9db0f76aff9a6ce179b6fbe329bf5569) and [mbedtls\_rsa\_private()](https://tls.mbed.org/api/rsa_8h.html#a3e016741ddaa354916ade4d48e9a0965)).
  + Elliptic Curves over GF(p) (see [mbedtls\_ecp\_point\_init()](https://tls.mbed.org/api/ecp_8h.html#ae069f80bc2f9cf2215c34430a9ccd924)).
  + Elliptic Curve Digital Signature Algorithm (ECDSA) (see [mbedtls\_ecdsa\_init()](https://tls.mbed.org/api/ecdsa_8h.html#aca644ee02921388fdc42eb06377f4b62)).
  + Elliptic Curve Diffie Hellman (ECDH) (see [mbedtls\_ecdh\_init()](https://tls.mbed.org/api/ecdh_8h.html#a97e787a12f7aae180ab9828303199dca)).

This module provides encryption/decryption which can be used to provide secrecy.

It also provides asymmetric key functions which can be used for confidentiality, integrity, authentication and non-repudiation.

# Program Designing

# General steps of implementation –

# 1.Here are the general steps to encrypt/decrypt a file in Java:

# Create a key from a given byte array for a given algorithm.

# Get an instance of Cipher class for a given algorithm transformation.

# See document of the Cipher class for more information regarding supported algorithms and transformations.

# Initialize the Cipher with an appropriate mode (encrypt or decrypt) and the given Key.

# Invoke doFinal(input\_bytes) method of the Cipher class to perform encryption or decryption on the input\_bytes, which returns an encrypted or decrypted byte array.

# Read an input file to a byte array and write the encrypted/decrypted byte array to an output file accordingly.

# 2. Note about key size -

# The AES algorithm requires that the key size must be 16 bytes (or 128 bit).

# So if you provide a key whose size is not equal to 16 bytes, a java.security.InvalidKeyException will be thrown.

# In case your key is longer, you should consider using a padding mechanism that transforms the key into a form in which its size is multiples of 16 bytes.

# See the Cipher class Javadoc for more details.

# To demonstrate how to encrypt and decrypt in Java using the Java Cryptography Extension (JCE)

# Symmetric key and asymmetric key are the two basic types of cryptographic systems. They are also called as “secret key” and “public key” cryptography.

# One of the success factors to Java is attributed to the strong security it provides to the platform and applications. This is the first part of a series of tutorials on Java security I am about to write. In this tutorial, we will see a simple example on using the “secret key” cryptography with JCE.

# Kryptos.

# Symmetric Key Java Encryption Decryption Example

# The following example uses a symmetric key for encryption and decryption. “Data Encryption Standard (DES)” was a popular symmetric key algorithm. Presently DES is outdated. DESede is a triple DES and a stronger variant of DES.

# Add the Security Provider. We are using the SunJCE Provider that is available with the JDK.

# Generate Secret Key. Use KeyGenerator and an algorithm to generate a secret key. We are using DESede. There are other algorithms like blowfish.

# Encode Text. For consistency across platform encode the plaintext as byte using UTF-8 encoding.

# Encrypt Text. Instantiate Cipher with ENCRYPT\_MODE, use the secret key and encrypt the bytes.

# Decrypt Text. Instantiate Cipher with DECRYPT\_MODE, use the same secret key and decrypt the bytes.

# AES is the latest encryption standard over the DES. You can refer the

# encryption-decryption with AES symmetric algorithm using JCE tutorial.

# All the above-given steps and concept are same, we just replace the DES with AES.

# Symmetric Key Java Encryption Decryption Example

# The following example uses symmetric key for encryption and decryption. “Data Encryption Standard (DES)” was a popular symmetric key algorithm. Presently DES is outdated. DESede is a triple DES and a stronger variant of DES.

# Add the Security Provider. We are using the SunJCE Provider that is available with the JDK.

# Generate Secret Key. Use KeyGenerator and an algorithm to generate a secret key. We are using DESede. There are other algorithms like blowfish.

# Encode Text. For consistency across platform encode the plaintext as byte using UTF-8 encoding.

# Encrypt Text. Instantiate Cipher with ENCRYPT\_MODE, use the secret key and encrypt the bytes.

# Decrypt Text. Instantiate Cipher with DECRYPT\_MODE, use the same secret key and decrypt the bytes.

# AES is the latest encryption standard over the DES. You can refer the encryption decryption with AES symmetric algorithm using JCE tutorial. All the above-given steps and concept are same, we just replace the DES with AES.

# Testing

**Encryption Testing -**

Introduction to Encryption :-

Encryption is basically the method of disguising plain or clear text in such a way as to hide its contents from anyone for whom it is not intended.

A cryptographic algorithm, or cipher, is a mathematical function used for encryption and decryption and works in combination with a specific key. Different keys will encrypt the same plaintext to a different cipher text.

**Public Key Encryption**

Both PGP and GPG use public-key cryptography. In a public-key system, each user has a pair of keys consisting of a private key and a public key. The private key is to be kept secret to the user and should never be revealed and the public key can be given to anyone with whom the user wants to communicate. The public key can be thought of as an open safe. When you encrypt a document using a public key it is like putting it in the safe, shutting it and spinning the combination lock several times. The corresponding private key is the combination to open the safe. Therefore only the person who holds the private key can recover the document encrypted using the associated public key.

**GPG vs PGP**

PGP (short for Pretty Good Privacy) is a public key encryption program designed to deliver automated encryption services to a number integrated applications. GPG is basically a free tool that provides the same functionality as PGP. Keys created in one can be used in the other and similarly files encrypted using one can be decrypted using the other. GPG is, by default, a command line tool. There are graphical interfaces available for it but it is not necessary to install them for this test. For the purpose of this test we are going to use PGP and GPG separately.

**Passphrases**

In order to unlock your private key you must create a passphrase. There is no limit on the length of a passphrase, and it should be carefully chosen. From the perspective of

security, the passphrase to unlock the private key is one of the weakest points in public-key encryption since it is the only protection you have if another individual gets your private key. Ideally, the passphrase should not use words from a dictionary and should mix the case of alphabetic characters as well as use non-alphabetic characters. A good passphrase is crucial to the secure use of public-key encryption.

**PGP**

1)Go to www.pgp.com and download the 30 day trial version of PGP Desktop (you should get the documentation with the download).

2)Install the software and activate using your trial serial number

3)Start up the program but quit the setup assistant .

4)Generate a PGP keypair by going to File -> New -> PGP Key but don't publish your public key to the PGP Global Directory if asked.

5)Add yourself to the revokers list on your keypair – see instructions in the user guide.

6)Encrypt a file (any file will do) with your own key.

7)Export your public key by going to File -> Export

8)Send your public key, and the file you encrypted in (5) to jonathan.ashton@oucs.ox.ac.uk with the subject line: “PGP TEST”.

9)Follow the instruction in the encrypted file that I send you back before moving on to test GPG.

**GPG**

1)Go to www.gnupg.org and download the appropriate version for your Operating System

2)From the user guide in the documentation section of the website, follow the instructions for

1.Generating a new keypair (use the default key size – 1024 bits)

2.Generating a revocation certificate.

3.Exchanging keys – exporting a public key.

3)Encrypt a file ( as described in encrypting and decrypting documents) using your

own key.

4)Send the encrypted file to the email address given above with the subject line : “GPG TEST”

5)Follow the instruction in the encrypted file that I send you back.

**Designing & Coding**

**DES Coding**

import javax.swing.\*;

import java.security.SecureRandom;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Random ;

class DES {

byte[ ] skey = new byte[1000];

String skeyString;

static byte[ ] raw;

String inputMessage,encryptedData,decryptedMessage;

public DES( ) {

try {

generateSymmetricKey( );

inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");

byte[ ] ibyte = inputMessage.getBytes( );

byte[ ] ebyte=encrypt(raw, ibyte);

String encryptedData = new String(ebyte);

System.out.println("Encrypted message "+encryptedData);

JOptionPane.showMessageDialog(null,"Encrypted Data "+"\n"+encryptedData);

byte[ ] dbyte= decrypt(raw,ebyte);

String decryptedMessage = new String(dbyte);

System.out.println("Decrypted message "+decryptedMessage);

JOptionPane.showMessageDialog(null,"Decrypted Data "+"\n"+decryptedMessage);

}

catch(Exception e) {

System.out.println(e);

}

}

void generateSymmetricKey() {

try {

Random r = new Random();

int num = r.nextInt(10000);

String knum = String.valueOf(num);

byte[] knumb = knum.getBytes();

skey=getRawKey(knumb);

skeyString = new String(skey);

System.out.println("DES Symmetric key = "+skeyString);

}

catch(Exception e) {

System.out.println(e);

}

}

private static byte[ ] getRawKey(byte[ ] seed) throws Exception {

KeyGenerator kgen = KeyGenerator.getInstance("DES");

SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");

sr.setSeed(seed);

kgen.init(56, sr);

SecretKey skey = kgen.generateKey( );

raw = skey.getEncoded( );

return raw;

}

private static byte[ ] encrypt(byte[ ] raw, byte[ ] clear) throws Exception {

SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");

Cipher cipher = Cipher.getInstance("DES");

cipher.init(Cipher.ENCRYPT\_MODE, skeySpec);

byte[ ] encrypted = cipher.doFinal(clear);

return encrypted;

}

private static byte[ ] decrypt(byte[ ] raw, byte[ ] encrypted) throws Exception {

SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");

Cipher cipher = Cipher.getInstance("DES");

cipher.init(Cipher.DECRYPT\_MODE, skeySpec);

byte[ ] decrypted = cipher.doFinal(encrypted);

return decrypted;

}

public static void main ( String args [ ] ) {

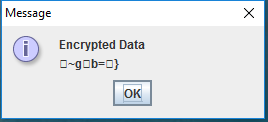
DES des = new DES ( );

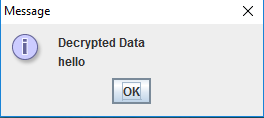
}

}

**OUTPUT**

****

****

****

**AES CODING**

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.xml.bind.DatatypeConverter;

/\*\*

\* This example program shows how AES encryption and decryption can be done in Java.

\* Please note that secret key and encrypted text is unreadable binary and hence

\* in the following program we display it in hexadecimal format of the underlying bytes.

\* @author Shubham

\*/

public class AESEncryption {

/\*\*

\* 1. Generate a plain text for encryption

\* 2. Get a secret key (printed in hexadecimal form). In actual use this must

\* by encrypted and kept safe. The same key is required for decryption.

\* 3.

\*/

public static void main(String[] args) throws Exception {

String plainText = "Hello World";

SecretKey secKey = getSecretEncryptionKey();

byte[] cipherText = encryptText(plainText, secKey);

String decryptedText = decryptText(cipherText, secKey);

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

System.out.println("AES Key (Hex Form):"+bytesToHex(secKey.getEncoded()));

System.out.println("Encrypted Text (Hex Form):"+bytesToHex(cipherText));

System.out.println("Descrypted Text:"+decryptedText);

}

/\*\*

\* gets the AES encryption key. In your actual programs, this should be safely

\* stored.

\* @return

\* @throws Exception

\*/

public static SecretKey getSecretEncryptionKey() throws Exception{

KeyGenerator generator = KeyGenerator.getInstance("AES");

generator.init(128); // The AES key size in number of bits

SecretKey secKey = generator.generateKey();

return secKey;

}

/\*\*

\* Encrypts plainText in AES using the secret key

\* @param plainText

\* @param secKey

\* @return

\* @throws Exception

\*/

public static byte[] encryptText(String plainText,SecretKey secKey) throws Exception{

// AES defaults to AES/ECB/PKCS5Padding in Java 7

Cipher aesCipher = Cipher.getInstance("AES");

aesCipher.init(Cipher.ENCRYPT\_MODE, secKey);

byte[] byteCipherText = aesCipher.doFinal(plainText.getBytes());

return byteCipherText;

}

/\*\*

\* Decrypts encrypted byte array using the key used for encryption.

\* @param byteCipherText

\* @param secKey

\* @return

\* @throws Exception

\*/

public static String decryptText(byte[] byteCipherText, SecretKey secKey) throws Exception {

// AES defaults to AES/ECB/PKCS5Padding in Java 7

Cipher aesCipher = Cipher.getInstance("AES");

aesCipher.init(Cipher.DECRYPT\_MODE, secKey);

byte[] bytePlainText = aesCipher.doFinal(byteCipherText);

return new String(bytePlainText);

}

/\*\*

\* Convert a binary byte array into readable hex form

\* @param hash

\* @return

\*/

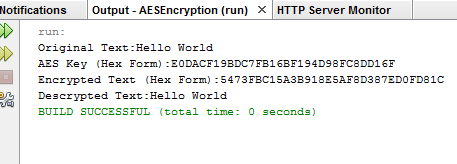
private static String bytesToHex(byte[] hash) {

return DatatypeConverter.printHexBinary(hash);

}

}

**OUTPUT**

****

**Conclusion**

## **OPRS Server Analysis**

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has the built-in flexibility of key length, which allows a degree of ‘future-proofing’ against progress in the ability to perform exhaustive key searches.

However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

## **OPRS Terminal Analysis**

The DES satisfies both the desired properties of the block cipher. These two properties make cipher very strong.

* **Avalanche effect**  − A small change in plaintext results in the very great change in the ciphertext.
* **Completeness** − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis has found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

### **OPRS application Analysis**

In CBC mode, the current plaintext block is added to the previous ciphertext block, and then the result is encrypted with the key. Decryption is thus the reverse process, which involves decrypting the current ciphertext and then adding the previous ciphertext block to the result.

The advantage of CBC over ECB is that changing IV results in different ciphertext for the identical message. On the drawback side, the error in transmission gets propagated to few further block during decryption due to chaining effect.

It is worth mentioning that CBC mode forms the basis for a well-known data origin authentication mechanism. Thus, it has an advantage for those applications that require both symmetric encryption and data origin authentication.

**REFERENCES**

[**www.google.com**](http://www.google.com)

[**www.wikipedia.com**](http://www.wikipedia.com)

[**www.webopedia.com**](http://www.webopedia.com)

[**www.tutorialspoint.com**](http://www.tutorialspoint.com)