**1. INTRODUCTION**

**T**HERE exist two mainstream approaches to steganography in empirical covers, such as digital media objects: steganography designed to preserve a chosen cover model and steganography minimizing a heuristically-defined embedding distortion. The strong argument for the former strategy is that provable undetectability can be achieved w.r.t. a specific model. The disadvantage is that an adversary can usually rather easily identify statistical quantities that go beyond the chosen model that allow reliable detection of embedding changes. The latter strategy is more pragmatic—it abandons modeling the cover source and instead tells the steganographer to embed payload while minimizing a distortion function. In doing so, it gives up any ambitions for perfect security. Although this may seem as a costly sacrifice, it is not, as empirical covers have been argued to be incognizable F , which prevents model-preserving approaches from being perfectly secure as well. While we admit that the relationship between distortion and steganographic security is far from clear, embedding while minimizing a distortion function is an easier problem than embedding with a steganographic constraint (preserving the distribution of covers). It is also more flexible, allowing the results obtained from experiments with blind steganalyzers to drive the design of the distortion function. In fact, today’s least detectable steganographic schemes for digital images – were designed using this principle. Moreover, when the distortion is defined as a norm between feature vectors extracted from cover and stego objects, minimizing distortion becomes tightly connected with model preservation insofar the features can be considered as a low-dimensional model of covers. This line of reasoning already appeared in and and was further developed in . With the exception of , steganographers work with additive distortion functions obtained as a sum of single-letter distortions. A well-known example is matrix embedding where the sender minimizes the total number of embedding changes. Near-optimal coding schemes for this problem appeared in and , together with other clever constructions and extensions . When the single-letter distortions vary across the cover elements, reflecting thus different costs of individual embedding changes, current coding methods are highly suboptimal . This paper provides a general methodology for embedding while minimizing an arbitrary additive distortion function with a performance near the theoretical bound. We present a complete methodology for solving both the payload-limited and the distortion-limited sender. The implementation described in this paper uses standard signal processing tools—convolutional codes with a trellis quantizer—and adapts them to our problem by working with their dual representation. These codes, which we call the syndrome-trellis codes (STCs), can directly improve the security of many existing steganographic schemes, allowing them to communicate larger payloads at the same embedding distortion or to decrease the distortion for a given payload. In addition, this work allows an iterative design of to the distortion function to minimize detectability measured using blind steganalyzers on real cover sources. This paper is organized as follows. In the next section, we introduce the central notion of a distortion function. The problem of embedding while minimizing distortion is formulated in Section III, where we introduce theoretical performance bounds as well as quantities for evaluating the performance of practical algorithms with respect to each other and the bounds. The syndrome coding method for steganographic communication is reviewed in Section IV. By pointing out the limitations of previous approaches, we motivate our contribution, which starts in Section V, where we introduce a class of syndrome-trellis codes for binary embedding operations. We describe the construction and optimization of the codes and provide extensive experimental results on different distortion profiles including the wet paper channel. In Section VI, we show how to decompose the problem of embedding using nonbinary embedding operations to a series of binary problems using a multilayered approach so that practical algorithms can be realized using binary STCs. The application and merit of the proposed coding construction is demonstrated experimentally in Section VII on covers formed by digital images in raster and transform (JPEG) domains. Both the binary and nonbinary versions of payload and distortion-limited senders are tested by blind steganalysis. Finally, the paper is concluded in Section VIII. This paper is a journal version of and , where the STCs and the multilayered construction were introduced. This paper unifies these methods into a complete and self-contained framework. Novel performance results and comparisons are included.

* 1. **Existing System:**

The existing system is an Image steganogarphy but this has a limitation of amount of data that could be stored particularly if the size of the data after encryption is enormous.

**Disadvantages:**

* Size of data to hide is a limitation
* Image is to be large in size
* More number of image bits are to be used for steganographic hiding using algorithms like LSB.
  1. **Proposed System:**

The proposed system is more robust and can hide enormous data in it as it is a video file that hides the actual data unlike images.

**Advantages:**

* Large data could be hidden in a video file
* Video file is usually large and hence no suspicion about the file would arise
* Only rounding off certain parts of the video frames is done by DCT and thus enormous encrypted data could be stored in a video.

**2.LITERATURE SURVEY**

**2.1 PROJECT LITERATURE**

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy n company strength. Once these things r satisfied, ten next step is to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration r taken into account for developing the proposed system.

Forbidden Zone Data Hiding (FZDH) is introduced in The method depends on the Forbidden Zone (FZ) concept, which is defined as the host signal range where no alteration is allowed during data hiding process. FZDH makes use of FZ to adjust the robustness-invisibility trade-off

The mapping function in (2) states that the host signal is modified by adding an additional term, which is a scaled version of the quantization difference. In 1-D, this additional term is scalar, whereas in N-D host signal is moved along the quantization difference vector and towards the reconstruction point of the quantizer. Hence, embedding distortion is reduced and became smaller than the quantization error.

In order to fulfill the requirement of mutual exclusion, the reconstruction points of the quantizers that are indexed by different *m* should be non-overlapping, which can be achieved by using a base quantizer and shifting its reconstruction points depending on *m*, similar to Dither Modulation . A typical embedding function that uses a uniform quantizer.

**2.2. INTRODUCTION TO JAVA**

## 2.2.1 Java Technology

Java technology is both a programming language and a platform.

### The Java Programming Language

### The Java programming language is a high-level language that can be characterized by all the following buzzwords:

* + - Simple
    - Architecture neutral
    - Object oriented
    - Portable
    - Distributed
    - High performance
    - Interpreted
    - Multithreaded
    - Robust
    - Dynamic
    - Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



Fig. 2.1Compiling and Interpreting stages

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



Fig 2.2 Platform Independent nature

### 2.2.2 The Java Platform

A platform is the hardware or software environment in which a program runs. We’ve already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

* The Java Virtual Machine (Java VM)
* The Java Application Programming Interface (Java API)

You’ve already been introduced to the Java VM. It’s the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that’s running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Fig 2.3The Java API and the virtual machine insulate the program from the hardware.

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

## What Can Java Technology Do?

The most common types of programs written in the Java programming language are applets and applications. If you’ve surfed the Web, you’re probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

* **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
* **Applets**: The set of conventions used by applets.
* **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data Gram Protocol) sockets, and IP (Internet Protocol) addresses.
* **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
* **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
* **Software components**: Known as JavaBeans, can plug into existing component architectures.
* **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
* **Java Database Connectivity (JDBCTM)**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



Fig 2.4Java 2 SDK

## How Will Java Technology Change My Life?

We can’t promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

* **Get started quickly**: Although the Java programming language is a powerful object-oriented language, it’s easy to learn, especially for programmers already familiar with C or C++.
* **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
* **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people’s tested code and introduce fewer bugs.
* **Develop programs more quickly**: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
* **Avoid platform dependencies with 100% Pure Java**: You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure JavaTMProduct Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
* **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.
* **Distribute software more easily**: You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

## 2.3. NETWORKING

### TCP/IP STACK

The TCP/IP stack is shorter than the OSI one:



Fig 2.5 TCP/IP stack

TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

### IP datagram’s

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

### UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

### TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

### Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

### Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16-bit network addressing. Class C uses 24-bit network addressing and class D uses all 32.

### Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

### Host address

8 bits are finally used for host addresses within our subnet. This place a limit of 256 machines that can be on the subnet.

### Total address



Fig 2.6 Total Address

The 32-bit address is usually written as 4 integers separated by dots.

### Port addresses

A service exists on a host, and is identified by its port. This is a 16-bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

### Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

#include <sys/types.h>

#include <sys/socket.h>

int socket(int family, int type, int protocol);

Here "family" will be AF\_INET for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

What is a Java Web Application?

A Java web application generates interactive web pages containing various types of markup language (HTML, XML, and so on) and dynamic content. It is typically comprised of web components such as Java Server Pages (JSP), servlets and JavaBeans to modify and temporarily store data, interact with databases and web services, and render content in response to client requests.

Because many of the tasks involved in web application development can be repetitive or require a surplus of boilerplate code, web frameworks can be applied to alleviate the overhead associated with common activities. For example, many frameworks, such as Java Server Faces, provide libraries for templating pages and session management, and often promote code reuse.

What is Java EE?

Java EE (Enterprise Edition) is a widely-used platform containing a set of coordinated technologies that significantly reduce the cost and complexity of developing, deploying, and managing multi-tier, server-centric applications. Java EE builds upon the Java SE platform and provides a set of APIs (application programming interfaces) for developing and running portable, robust, scalable, reliable and secure server-side applications.

Some of the fundamental components of Java EE include:

Enterprise JavaBeans (EJB): a managed, server-side component architecture used to encapsulate the business logic of an application. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.

Java Persistence API (JPA): a framework that allows developers to manage data using object-relational mapping (ORM) in applications built on the Java Platform.

JavaScript and Ajax Development

JavaScript is an object-oriented scripting language primarily used in client-side interfaces for web applications. Ajax (Asynchronous JavaScript and XML) is a Web 2.0 technique that allows changes to occur in a web page without the need to perform a page refresh. JavaScript toolkits can be leveraged to implement Ajax-enabled components and functionality in web pages.

Web Server and Client

Web Server is a software that can process the client request and send the response back to the client. For example, Apache is one of the most widely used web server. Web Server runs on some physical machine and listens to client request on specific port.

A web client is a software that helps in communicating with the server. Some of the most widely used web clients are Firefox, Google Chrome, Safari etc. When we request something from server (through URL), web client takes care of creating a request and sending it to server and then parsing the server response and present it to the user.

HTML and HTTP

Web Server and Web Client are two separate software’s, so there should be some common language for communication. HTML is the common language between server and client and stands for Hypertext Markup Language.

Web server and client needs a common communication protocol, HTTP (Hypertext Transfer Protocol) is the communication protocol between server and client. HTTP runs on top of TCP/IP communication protocol.

Some of the important parts of HTTP Request are:

HTTP Method – action to be performed, usually GET, POST, PUT etc.

URL – Page to access

Form Parameters – similar to arguments in a java method, for example user, password details from login page.

Sample HTTP Request:

|  |  |
| --- | --- |
| 1  2  3 | GET /FirstServletProject/jsps/hello.jsp HTTP/1.1  Host: localhost:8080  Cache-Control: no-cache |

Some of the important parts of HTTP Response are:

Status Code – an integer to indicate whether the request was success or not. Some of the well-known status codes are 200 for success, 404 for Not Found and 403 for Access Forbidden.

Content Type – text, html, image, pdf etc. Also known as MIME type

Content – actual data that is rendered by client and shown to user.

**MIME Type or Content Type**: If you see above sample HTTP response header, it contains tag “Content-Type”. It’s also called MIME type and server sends it to client to let them know the kind of data it’s sending. It helps client in rendering the data for user. Some of the mostly used mime types are text/html, text/xml, application/xml etc.

**Understanding URL**

URL is acronym of Universal Resource Locator and it’s used to locate the server and resource. Every resource on the web has it’s own unique address. Let’s see parts of URL with an example.

**http://localhost:8080/FirstServletProject/jsps/hello.jsp**

**http://** – This is the first part of URL and provides the communication protocol to be used in server-client communication.

**localhost** – The unique address of the server, most of the times it’s the hostname of the server that maps to unique IP address. Sometimes multiple hostnames point to same IP addresses and web server virtual host takes care of sending request to the particular server instance.

**8080** – This is the port on which server is listening, it’s optional and if we don’t provide it in URL then request goes to the default port of the protocol. Port numbers 0 to 1023 are reserved ports for well known services, for example 80 for HTTP, 443 for HTTPS, 21 for FTP etc.

**FirstServletProject/jsps/hello.jsp** – Resource requested from server. It can be static html, pdf, JSP, servlets, PHP etc.

**Why we need Servlet and JSPs?**

Web servers are good for static contents HTML pages but they don’t know how to generate dynamic content or how to save data into databases, so we need another tool that we can use to generate dynamic content. There are several programming languages for dynamic content like PHP, Python, Ruby on Rails, Java Servlets and JSPs.

Java Servlet and JSPs are server side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.

**Web Container**

Tomcat is a web container, when a request is made from Client to web server, it passes the request to web container and its web container job to find the correct resource to handle the request (servlet or JSP) and then use the response from the resource to generate the response and provide it to web server. Then web server sends the response back to the client.

When web container gets the request and if it’s for servlet then container creates two Objects HTTPServletRequest and HTTPServletResponse. Then it finds the correct servlet based on the URL and creates a thread for the request. Then it invokes the servlet service() method and based on the HTTP method service() method invokes doGet() or doPost() methods. Servlet methods generate the dynamic page and write it to response. Once servlet thread is complete, container converts the response to HTTP response and send it back to client.

Some of the important work done by web container are:

**Communication Support** – Container provides easy way of communication between web server and the servlets and JSPs. Because of container, we don’t need to build a server socket to listen for any request from web server, parse the request and generate response. All these important and complex tasks are done by container and all we need to focus is on our business logic for our applications.

**Lifecycle and Resource Management** – Container takes care of managing the life cycle of servlet. Container takes care of loading the servlets into memory, initializing servlets, invoking servlet methods and destroying them. Container also provides utility like JNDI for resource pooling and management.

**Multithreading Support** – Container creates new thread for every request to the servlet and when it’s processed the thread dies. So servlets are not initialized for each request and saves time and memory.

**JSP Support** – JSPs doesn’t look like normal java classes and web container provides support for JSP. Every JSP in the application is compiled by container and converted to Servlet and then container manages them like other servlets.

**Miscellaneous Task** – Web container manages the resource pool, does memory optimizations, run garbage collector, provides security configurations, support for multiple applications, hot deployment and several other tasks behind the scene that makes our life easier.

**Web Application Directory Structure**

Java Web Applications are packaged as Web Archive (WAR) and it has a defined structure. You can export above dynamic web project as WAR file and unzip it to check the hierarchy. It will be something like below image.

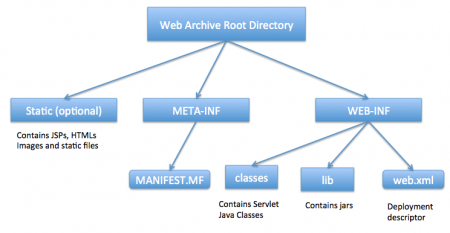
[](http://www.journaldev.com/wp-content/uploads/2013/08/WAR-directory-structure.png)

Fig 2.7 Web Archive Root Directory

# JSP

**2.4.1 INTRODUCTION TO JAVA SERVER PAGE**

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with **<%** and end with **%>**.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

**Why Use JSP?**

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison withthe CGI.

* Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
* JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
* JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC**, **JNDI**, **EJB**, **JAXP**, etc.
* JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.
* Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

**Advantages of JSP**

Following is the list of other advantages of using JSP over other technologies:

**2.4.2 JSP VS ACTIVE SERVER PAGE (ASP)**

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

**vs. Pure Servlets**

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

**vs. Server-Side Includes (SSI)**

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

**vs. JavaScript**

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

**vs. Static HTML**

Regular HTML, of course, cannot contain dynamic information.

**2.4.3SETTING UP JAVA DEVELOPMENT KIT**

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up the PATH environment variable appropriately.

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set the **PATH and JAVA\_HOME** environment variables to refer to the directory that contains **java** and **javac**, typically **java\_install\_dir/bin** and **java\_install\_dir** respectively.

If you are running Windows and install the SDK in **C:\jdk1.5.0\_20**, you need to add the following line in your **C:\autoexec.bat** file.

set PATH=C:\jdk1.5.0\_20\bin;%PATH%

set JAVA\_HOME=C:\jdk1.5.0\_20

Alternatively, on **Windows NT/2000/XP**, you can also right-click on **My Computer**, select **Properties**, then **Advanced**, followed by **Environment Variables**. Then, you would updatethe PATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if the SDK is installed in **/usr/local/jdk1.5.0\_20** and you use the C shell, you will put the following into your **.cshrc** file.

setenv PATH /usr/local/jdk1.5.0\_20/bin:$PATH setenv JAVA\_HOME /usr/local/jdk1.5.0\_20

Alternatively, if you use an **Integrated Development Environment (IDE)** like **BorlandJBuilder**, **Eclipse**, **IntelliJ IDEA**, or **Sun ONE Studio**, compile and run a simple programto confirm that the IDE knows where you installed Java.

**2.4.4SETTING UP WEB SERVER:TOMCAT**

A number of Web Servers that support JavaServer Pages and Servlets development are available in the market. Some web servers can be downloaded for free and Tomcat is one of them.

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server. Here are the steps to set up Tomcat on your machine:

* Once you downloaded the installation, unpack the binary distribution into a convenient location. For example, in **C:\apache-tomcat-5.5.29** on windows, or **/usr/local/apache-tomcat-5.5.29** on Linux/Unix and create **CATALINA\_HOME** environment variable pointing to these locations.

Tomcat can be started by executing the following commands on the Windows machine:

%CATALINA\_HOME%\bin\startup.bat

or

C:\apache-tomcat-5.5.29\bin\startup.bat

Tomcat can be started by executing the following commands on the Unix (Solaris, Linux, etc.)

machine:

$CATALINA\_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-5.5.29/bin/startup.sh

After a successful startup, the default web-applications included with Tomcat will be available by visiting

**http://localhost:8080/**

Upon execution, you will receive the following output:

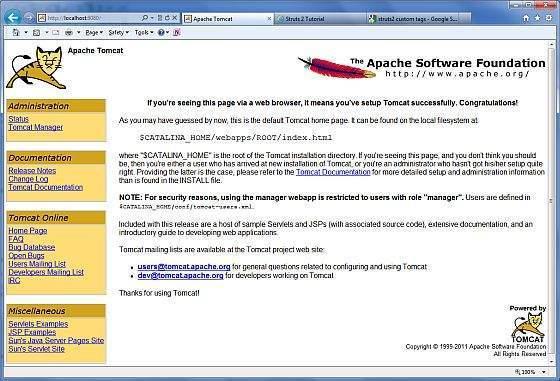


Fig 2.8 Tomcat excution

Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: **http://tomcat.apache.org**

Tomcat can be stopped by executing the following commands on the Windows machine:

%CATALINA\_HOME%\bin\shutdown

or

C:\apache-tomcat-5.5.29\bin\shutdown

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.)

machine:

$CATALINA\_HOME/bin/shutdown.sh

Or

/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh

**2.4.5SETTING UP CLASSPATH**

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.

If you are running Windows, you need to put the following lines in your **C:\autoexec.bat** file.

set CATALINA=C:\apache-tomcat-5.5.29

set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%

Alternatively, on **Windows NT/2000/XP**, you can also right-click on **My Computer**, select **Properties**, then **Advanced**, then **Environment Variables**. Then, you would update theCLASSPATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your **.cshrc** file.

setenv CATALINA=/usr/local/apache-tomcat-5.5.29

setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH

**NOTE:** Assuming that your development directory is **C:\JSPDev (Windows)** or/**usr/JSPDev (Unix)**, then you would need to add these directories as well in CLASSPATH

**2.4.6JSP ARCHITECTURE**

The web server needs a JSP engine, i.e., a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

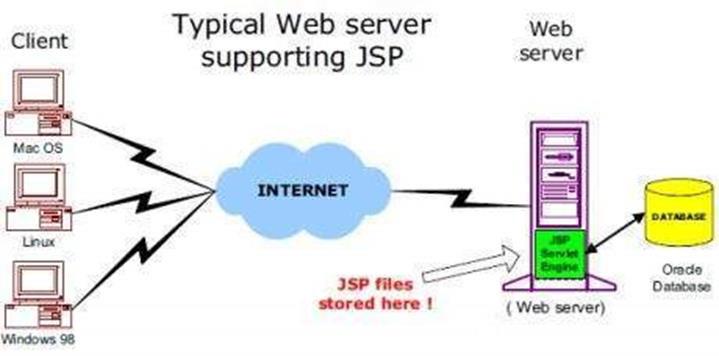


Fig 2.9 Jsp Architecture

**JSP Processing**

The following steps explain how the web server creates the Webpage using JSP:

* As with a normal page, your browser sends an HTTP request to the web server.
* The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
* The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to **println( )** statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
* The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
* A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. This output is further passed on to the web server by the servlet engine inside an HTTP response.
* The web server forwards the HTTP response to your browser in terms of static HTML content.
* Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram:

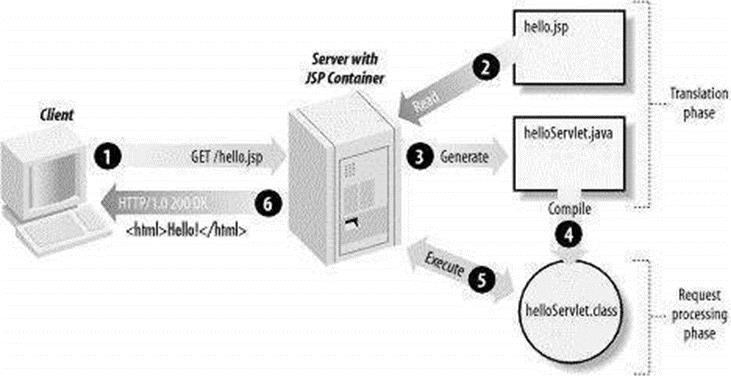


Fig 2.10.Jsp Processing

Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet

**2.4.7 JSP LIFE CYCLE**

In this chapter, we will discuss the lifecycle of JSP. The key to understanding the low-level functionality of JSP is to understand the simple life cycle they follow.

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

**Paths Followed By JSP**

The following are the paths followed by a JSP

* Compilation
* Initialization
* Execution
* Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below:

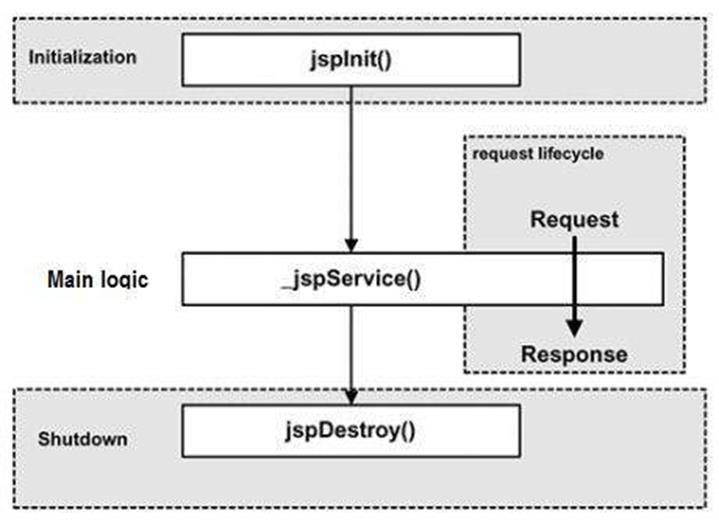


Fig 2.11.Jsp Life cycle

**JSP Compilation**

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps:

* Parsing the JSP.
* Turning the JSP into a servlet.
* Compiling the servlet.

**JSP Initialization**

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests.

If you need to perform JSP-specific initialization, override the **jspInit()** method:

public void jspInit(){

// Initialization code...

}

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

**JSP Execution**

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **\_jspService()** method in the JSP.

The \_jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

void \_jspService(HttpServletRequest request, HttpServletResponse response)

{

// Service handling code...

}

The **\_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e., **GET**, **POST**, **DELETE**, etc.

**JSP Cleanup**

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form:

public void jspDestroy()

{

// Your cleanup code goes here.

}

**2.4.8 JSP SYNTAX**

**Elements of JSP**

The elements of JSP have been described below:

**The Scriptlet**

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet:

<% code fragment %>

You can write the XML equivalent of the above syntax as follows:

<jsp:scriptlet>

code fragment

</jsp:scriptlet>

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP:

<html>

<head><title>Hello World</title></head>

<body>

Hello World!<br/>

<%

out.println("Your IP address is " + request.getRemoteAddr());

%>

</body>

</html

**2.4.9JSP DIRECTIVES**

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@ directive attribute="value" %>

There are three types of directive tag:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Directive** |  |  | **Description** |  |
|  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | <%@ page ... %> | |  | Defines page-dependent attributes, such as scripting | |
|  |  | language, error page, and buffering requirements. | |
|  |  |  |  |
|  |  | |  |  | |
|  | <%@ include ... %> | |  | Includes a file during the translation phase. | |
|  |  |  |  |  | |
|  | <%@ taglib ... %> | |  | Declares a tag library, containing custom actions, used in the | |
|  |  | Page | |
|  |  |  |  |
|  |  |  |  |  |  |

Tab 2.1 Jsp directives

**2.4.10JSP ACTIONS**

JSP actions use **constructs** in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard:

<jsp:action\_name attribute="value" />

Action elements are basically predefined functions. Following table lists out the available JSP Actions:

|  |  |
| --- | --- |
| **Syntax** | **Purpose** |
| jsp:include | Includes a file at the time the page is requested. |
| jsp:useBean | Finds or instantiates a JavaBean. |
| jsp:setProperty | Sets the property of a JavaBean. |
| jsp:getProperty | Inserts the property of a JavaBean into the output. |
| jsp:forward | Forwards the requester to a new page. |
| jsp:plugin | Generates browser-specific code that makes an OBJECT or  EMBED tag for the Java plugin. |
| jsp:element | Defines XML elements dynamically. |
| jsp:attribute | Defines dynamically-defined XML element's attribute. |
| jsp:body | Defines dynamically-defined XML element's body. |
| jsp:text | Used to write template text in JSP pages and documents. |

Tab 2.2 Jsp Actions

**2.4.11JSP IMPLICIT OBJECTIVES**

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Objects** |  |  | **Description** |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Request | |  | This is the **HttpServletRequest** object associated with the | |  |
|  |  |  |  | request. |  |  |
|  |  | |  |  | |  |
|  | Response | |  | This is the **HttpServletResponse** object associated with the | |  |
|  |  |  |  | response to the client. |  |  |
|  |  | |  |  | |  |
|  | Out | |  | This is the **PrintWriter** object used to send output to the | |  |
|  |  |  |  | client. |  |  |
|  |  | |  |  | |  |
|  | Session | |  | This is the **HttpSession** object associated with the request. | |  |
|  |  | |  |  |  |  |
|  | Application | |  | This is the **ServletContext** object associated |  |  |
|  |  |  |  | application context. |  |  |
|  |  | |  |  | |  |
|  | Config | |  | This is the **ServletConfig** object associated with the page. | |  |
|  |  | |  |  | |  |
|  | PageContext | |  | This encapsulates use of server-specific features like higher | |  |
|  |  |  |  | performance **JspWriters**. |  |  |
|  |  | |  |  | |  |
|  | Page | |  | This is simply a synonym for **this**, and is used to call the | |  |
|  |  |  |  | methods defined by the translated servlet class. |  |  |
|  |  | |  |  |  |  |
|  | Exception | |  | The **Exception** object allows the exception data |  |  |
|  |  |  |  | accessed by designated JSP. |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Tab 2.3 Jsp Implicit objctives

**2.4.12JSP PAGE DIRECTIVES**

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@ directive attribute="value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag:

|  |  |
| --- | --- |
| **Directive** | **Description** |
| <%@ page ... %> | Defines page-dependent attributes, such as scripting  language, error page, and buffering requirements. |
| <%@ include ... %> | Includes a file during the translation phase. |
| <%@ taglib ... %> | Declares a tag library, containing custom actions, used in the page |

Tab 2.4 jsp Page Directives

**The page Directive**

The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive:

<%@ page attribute="value" %>

You can write the XML equivalent of the above syntax as follows:

<jsp:directive.page attribute="value" />

**Attribute**Following table lists out the attributes associated with page directive

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Attribute** |  |  | **Purpose** |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Buffer | |  | Specifies a buffering model for the output stream. |  |  |
|  |  | |  |  |  |  |
|  | AutoFlush | |  | Controls the behavior of the servlet output buffer. |  |  |
|  |  | |  |  |  |  |
|  | ContentType | |  | Defines the character encoding scheme. |  |  |
|  |  |  |  |  |  |  |
|  | ErrorPage | |  | Defines the URL of another JSP that reports | on | Java |
|  |  | unchecked runtime exceptions. |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  | | |
|  | IsErrorPage | |  | Indicates if this JSP page is a URL specified by another JSP | | |
|  |  | page's errorPage attribute. |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  | |  |
|  | Extends | |  | Specifies a superclass that the generated servlet | | Must |
|  |  | extend. |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  | | |
|  | Import | |  | Specifies a list of packages or classes for use in the JSP as | | |
|  |  | the Java import statement does for Java classes. |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  | |
|  | Info | |  | Defines a string that can be accessed with the | servlet's | |
|  |  | **getServletInfo()** method. |  |  |
|  |  |  |  |  |  |
|  |  | |  |  | |  |
|  | IsThreadSafe | |  | Defines the threading model for the generated servlet. | |  |
|  |  | |  |  | | |
|  | Language | |  | Defines the programming language used in the JSP page. | | |
|  |  |  |  |  | | |
|  | Session | |  | Specifies whether or not the JSP page participates in HTTP | | |
|  |  | Sessions |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Tab 2.5 page directive Attributes

**3. SYSTEM ANALYSIS AND REQUIREMENTS**

**3.1 FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**3.1.1 ECONOMICAL FEASIBILITY**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### 3.1.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**3.1.3 SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**3.2 SOFTWARE AND HARDWARE REQUIREMENTS**

**3.2.1 Hardware Requirements**

* System : Pentium IV 2.4 GHz or Above
* Hard Disk : 80 GB.
* Floppy Drive : 1.44 Mb.
* Monitor : 15 VGA Colour.
* Mouse : Logitech.
* Ram : 2 GB

**3.2.2 Software Requirements**

* Operating system : Windows XP Professional/Vista/7/8/8.1 or Linux
* Front End : JAVA,JDK1.7
* Tool : NetBeans

**3.3 PERFORMANCE REQUIREMENTS**

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely with the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

The system should be able to interface with the existing system

* The system should be accurate
* The system should be better than the existing system

The existing system is completely dependent on the user to perform all the duties.

**4. SOFTWARE DESIGN**

**4.1 INTRODUCTION**

**software design** is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap and synergy with the disciplines of systems analysis, systems architecture and systems engineering.

**4.2 UML DIAGRAMS**

**Unified Modeling Language**

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagrams, which is as follows.

* + User Model View
    1. This view represents the system from the user’s perspective.
    2. The analysis representation describes a usage scenario from the end-user’s perspective.
  + Structural model view
    1. In this model, the data and functionality are arrived from inside the system.
    2. This model view models the static structures.
* Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

* Implementation Model View

In this the structural and behavioral as parts of the system are represented as they are to be built.

* Environmental Model View

In this the structural and behavioral aspect of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

* UML Analysis modeling, this focuses on the user model and structural model views of the system.
* UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

4**.2.1 CLASS DIAGRAM**

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

A class exists with three sections. In the diagram, classes are represented with boxes which contain three parts:

* The upper part holds the name of the class
* The middle part contains the attributes of the class
* The bottom part gives the methods or operations the class can take or undertake

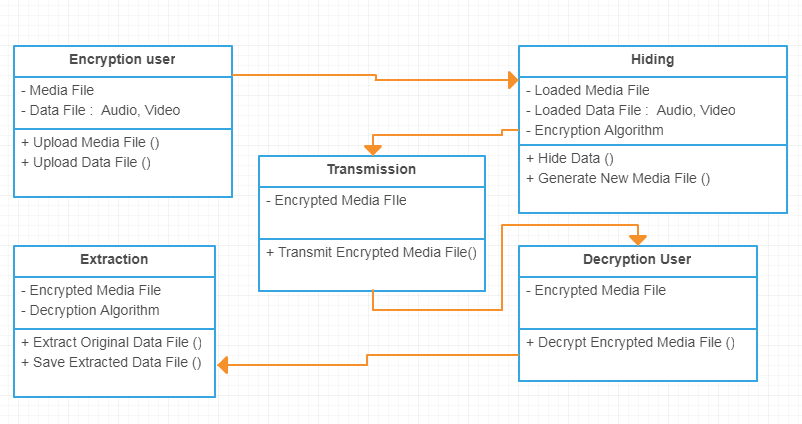
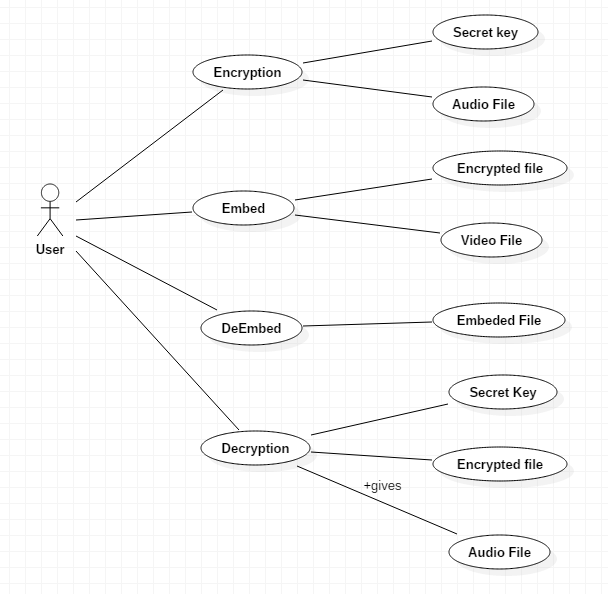


Fig 4.1 Class Diagram

* + 1. **USE CASE DIAGRAM**

A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a [use case](http://en.wikipedia.org/wiki/Use_Case). A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual [use case](http://en.wikipedia.org/wiki/Use_Case) and will often be accompanied by other types of diagrams as well.



4.2 Use Case Diagram

* + 1. **SEQUENCE DIAGRAM**

A sequence diagram is a kind of [interaction diagram](http://en.wikipedia.org/wiki/Interaction_diagram) that shows how processes operate with one another and in what order. It is a construct of a [Message Sequence Chart](http://en.wikipedia.org/wiki/Message_Sequence_Chart). A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical

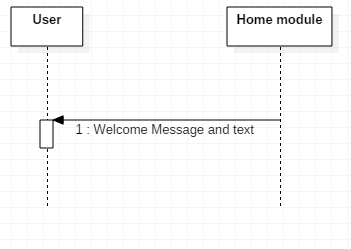


Fig:4.3.home sequence Diagram

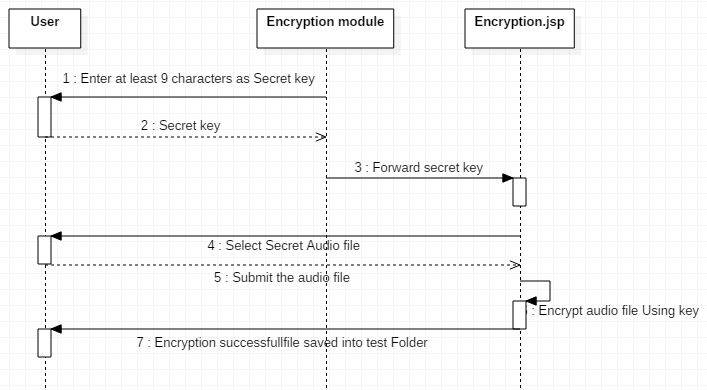


Fig:4.4.Encryption sequence Diagram

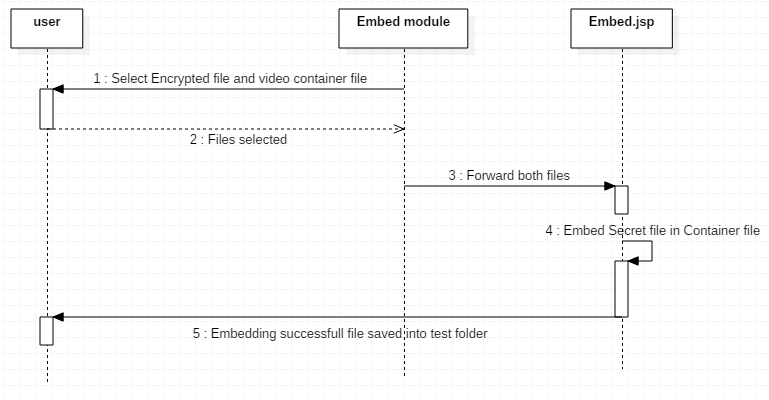


Fig:4.5.Embed sequence Diagram

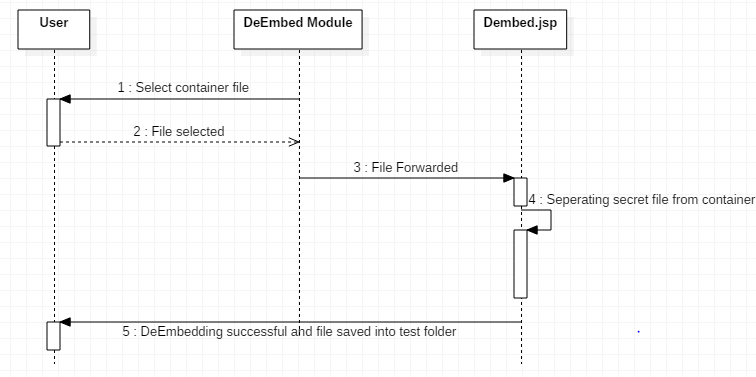


Fig:4.6.DeEnbed sequence Diagram

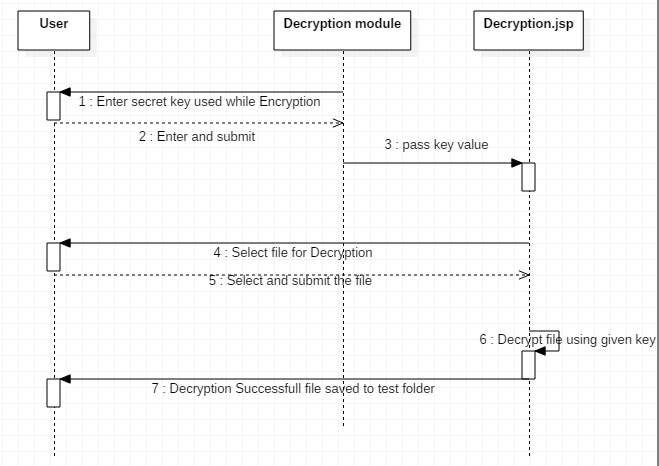


Fig:4.7.Decryption sequence Diagram

* + 1. **ACTIVITY DIAGRAM**

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

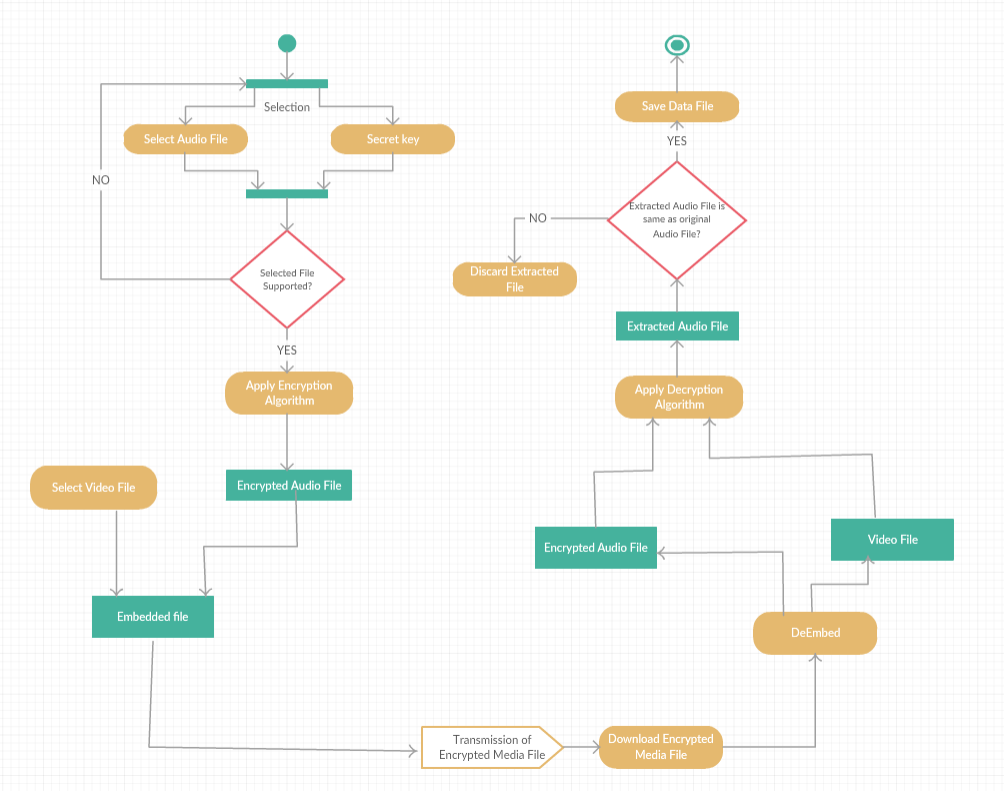


Fig:4.8 Activity Diagram

**4.2.5 DATA FLOW DIAGRAM**

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

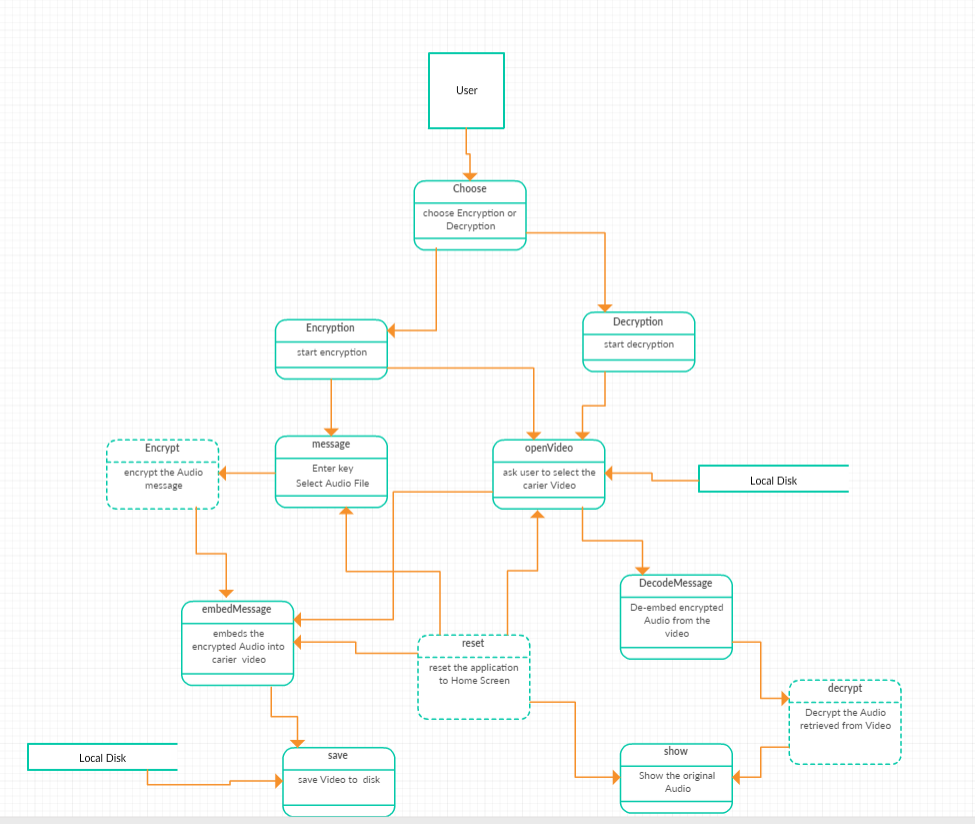
****

Fig:4.9 Data Flow Diagram

**4.3 MODULE DESCRIPTION**

MODULES

* **Encryption Module**
* **Embedding Module**
* **DeEmbedding Module**
* **DecryptionModule**

**4.3.1 Encryption Module:**

The Encryption Module is designed as such a way that it accepts the secret audio message from user and then asks for a secret unique key for encryption which is needed for decryption at the other end, this adds another layer to the steganography layer for additional security.

* + 1. **Embedding Module:**

The Embedding Module is designed as such a way that the proposed system must be capable of handling any type of data formats, such as if the user wishes to hide any secret audio file format then it must be compatible with all usual Video formats such as MP4, avi, so that the user can use any formats to hide the secret data. It uses it as container file used to transmit the data, If anyone snoops on it he can see only container file.

* + 1. **DeEmbedding Module:**

In this module we separate and extract the Secret audio file from video container file by selecting the file and running it through deembedding algorithm.

* + 1. **Decryption Module:**

We use the original key used while encryption to Decrypt the audio file to get the original message audio and the secret message is passed without anyone knowing.

**5. CODING TEMPLATES / CODE**

**5.1 CODE TEMPLATE**

**<%@page import="java.io.FileInputStream"%>**

**<%@page import="java.io.FileInputStream"%>**

**<%@page import="java.io.FileOutputStream"%>**

**<%@page import="java.io.IOException"%>**

**<%@page import="java.io.InputStream"%>**

**<%@page import="java.io.OutputStream"%>**

**<%@page import="javax.crypto.Cipher"%>**

**<%@page import="javax.crypto.CipherInputStream"%>**

**<%@page import="javax.crypto.CipherOutputStream"%>**

**<%@page import="javax.crypto.SecretKey"%>**

**<%@page import="javax.crypto.SecretKeyFactory"%>**

**<%@page import="javax.crypto.spec.DESKeySpec"%>**

**<%{**

**String key =request.getParameter("t1");**

**String path=request.getParameter("t2");**

**try {**

**String outpath = "d:\\test\\decrypted.mp3";**

**FileInputStream fis = new FileInputStream(path);**

**FileOutputStream fos = new FileOutputStream(outpath);**

**DESKeySpec dks = new DESKeySpec(key.getBytes());**

**SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");**

**SecretKey desKey = skf.generateSecret(dks);**

**Cipher cipher = Cipher.getInstance("DES"); // DES/ECB/PKCS5Padding for SunJCE**

**cipher.init(Cipher.DECRYPT\_MODE, desKey);**

**CipherOutputStream cos = new CipherOutputStream(fos, cipher);**

**byte[] bytes = new byte[64];**

**int numBytes;**

**while ((numBytes = fis.read(bytes)) != -1) {**

**cos.write(bytes, 0, numBytes);**

**}**

**out.println("Decryption successfull");**

**out.println("file saved to=");**

**out.println(outpath);**

**fos.flush();**

**fos.close();**

**fis.close();**

**}**

**catch (Throwable e) {**

**e.printStackTrace();**

**}**

**}**

**%>**

**<%@page contentType="text/html" pageEncoding="UTF-8"%>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">**

**<title>JSP Page</title>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<ul>**

**<li><a href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a class="active" href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<div><h1><b><center><font size="10">Hello World!</font></b></center></h1><br></div>**

**<br>**

**<br>**

**<%**

**String video=request.getParameter("video");**

**test1.deembed.deembeding(video);**

**out.println("Successfully retrieved secret file from container file : "+video);**

**%>**

**</div>**

**</body>**

**</html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<ul>**

**<li><a href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a class="active "href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<div><h1><b><center><font size="10">Select File for Decryption</font></b></center></h1><br></div>**

**<br>**

**<br>**

**<form action="Decrypt.jsp" method ="post">**

**<%**

**String s1=request.getParameter("t1");**

**%>**

**<input type ="hidden" name='t1' value='<%= s1%>' >**

**<center><input type ="file" name="t2" size = "50">**

**<input type ="submit" value ="submit"></center>**

**</form>**

**</div>**

**</body>**

**</html>**

**<%--**

**Document : t2**

**Created on : Feb 22, 2018, 3:29:53 PM**

**Author : Administrator**

**--%>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<ul>**

**<li><a href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a class="active" href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<div><h1><b><center><font size="10">Encryption</font></b></center></h1><br></div>**

**<br>**

**<br>**

**<form action="Encrypt.jsp" method ="post">**

**<%**

**String s1=request.getParameter("t1");**

**%>**

**<input type ="hidden" name='t1' value='<%= s1%>' >**

**<center><input type ="file" name="t2" size = "50">**

**<input type ="submit" value ="submit"></center>**

**</form>**

**</div>**

**</body>**

**</html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<ul>**

**<li><a href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a class="active" href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<div><h1><b><center><font size="10">Decryption</font></b></center></h1><br></div>**

**<br>**

**<br>**

**<p><h3><i><center>Enter secret key given while encryption</center></i></h3>**

**<form action="t2\_decrypt.jsp" method ="post">**

**<center>**

**<input type ="text" name="t1">**

**<input type ="submit" value ="submit">**

**</center>**

**</form>**

**</div>**

**</body>**

**</html>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<ul>**

**<li><a href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a class="active" href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<div><h1><b><center><font size="10">Enter Encryption Key</font></b></center></h1><br></div>**

**<br>**

**<br>**

**<p><h3><i><center>Enter at least 9 characters as secret key</center></i></h3>**

**<form action="t2.jsp" method ="post">**

**<center>**

**<input type ="text" name="t1">**

**<input type ="submit" value ="submit">**

**</center>**

**</form>**

**</p>**

**</div>**

**</body>**

**</html>**

**<%--**

**Document : index**

**Created on : 22 Feb, 2018, 8:45:46 PM**

**Author : arunn**

**--%>**

**<%@page contentType="text/html" pageEncoding="UTF-8"%>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">**

**<title>JSP Page</title>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<ul>**

**<li><a class="active" href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<div><h1><b><center><font size="10">Welcome to Video Steganography</font></b></center></h1><br></div>**

**<br>**

**<br>**

**</div>**

**</body>**

**</html>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 10%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #4CAF50;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">**

**<title>JSP Page</title>**

**</head>**

**<body bgcolor=" #F0FFFF">**

**<br>**

**<br>**

**<ul>**

**<li><a href="http://localhost:8080/Stegno\_\_/index.jsp">Home</a></li>**

**<li><a class="active" href="http://localhost:8080/Stegno\_\_/t1.jsp">Encryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/t1\_decrypt.jsp">Decryption</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/Embed1.jsp">Embed</a></li>**

**<li><a href="http://localhost:8080/Stegno\_\_/deembed1.jsp">Dembed</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px16px;height:200px;">**

**<%@page import="java.io.FileInputStream"%>**

**<%@page import="java.io.FileInputStream"%>**

**<%@page import="java.io.FileOutputStream"%>**

**<%@page import="java.io.IOException"%>**

**<%@page import="java.io.InputStream"%>**

**<%@page import="java.io.OutputStream"%>**

**<%@page import="javax.crypto.Cipher"%>**

**<%@page import="javax.crypto.CipherInputStream"%>**

**<%@page import="javax.crypto.CipherOutputStream"%>**

**<%@page import="javax.crypto.SecretKey"%>**

**<%@page import="javax.crypto.SecretKeyFactory"%>**

**<%@page import="javax.crypto.spec.DESKeySpec"%>**

**<%{**

**String key =request.getParameter("t1");**

**String path=request.getParameter("t2");**

**try {**

**FileInputStream is = new FileInputStream(path);**

**FileOutputStream os = new FileOutputStream("d:\\test\\encrypted.enc");**

**DESKeySpec dks = new DESKeySpec(key.getBytes());**

**SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");**

**SecretKey desKey = skf.generateSecret(dks);**

**Cipher cipher = Cipher.getInstance("DES"); // DES/ECB/PKCS5Padding for SunJCE**

**cipher.init(Cipher.ENCRYPT\_MODE, desKey);**

**CipherInputStream cis = new CipherInputStream(is, cipher);**

**byte[] bytes = new byte[64];**

**int numBytes;**

**while ((numBytes = cis.read(bytes)) != -1) {**

**os.write(bytes, 0, numBytes);**

**}**

**out.println("encryption successfull");**

**os.flush();**

**os.close();**

**cis.close();**

**} catch (Throwable e)**

**{ e.printStackTrace(); }**

**}**

**%>**

**</div>**

**</body>**

**</html>**

* 1. **RESULTS AND VALIDATION**

**6.1 OUTPUT SCREENS**

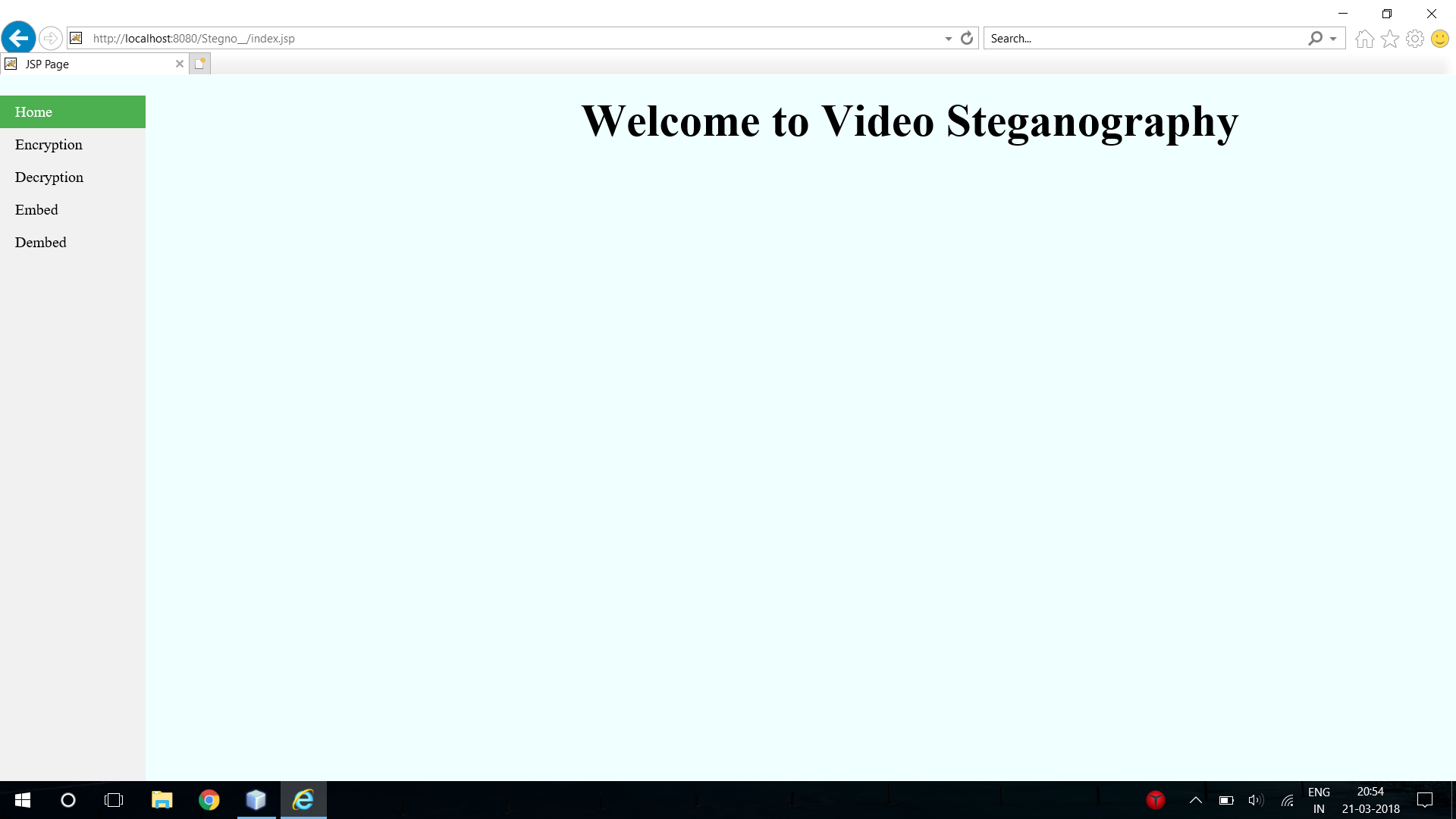
****

Fig:6.1. Home

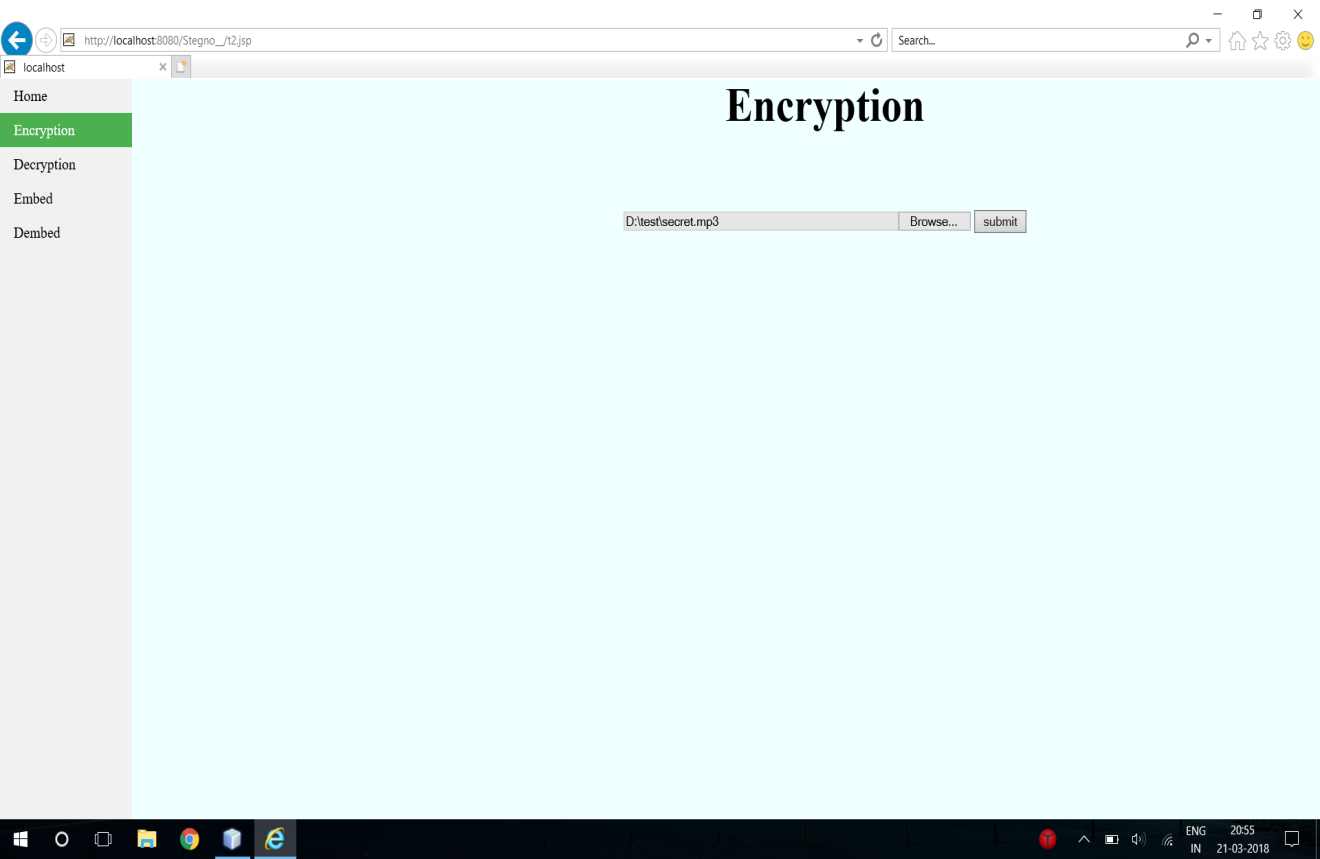
****

Fig:6.2 Encryption

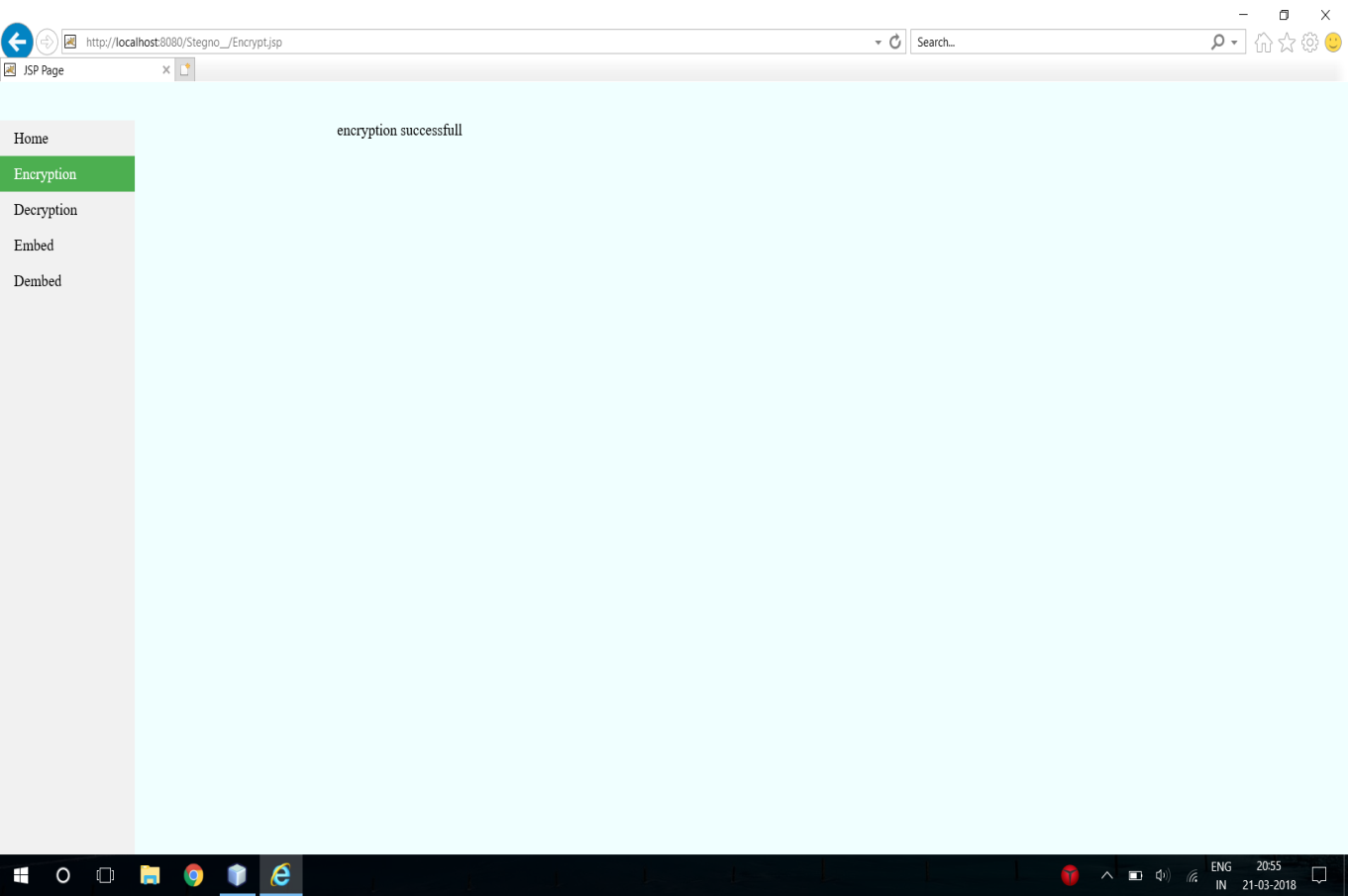
****

Fig:6.3 Encryption Successfull

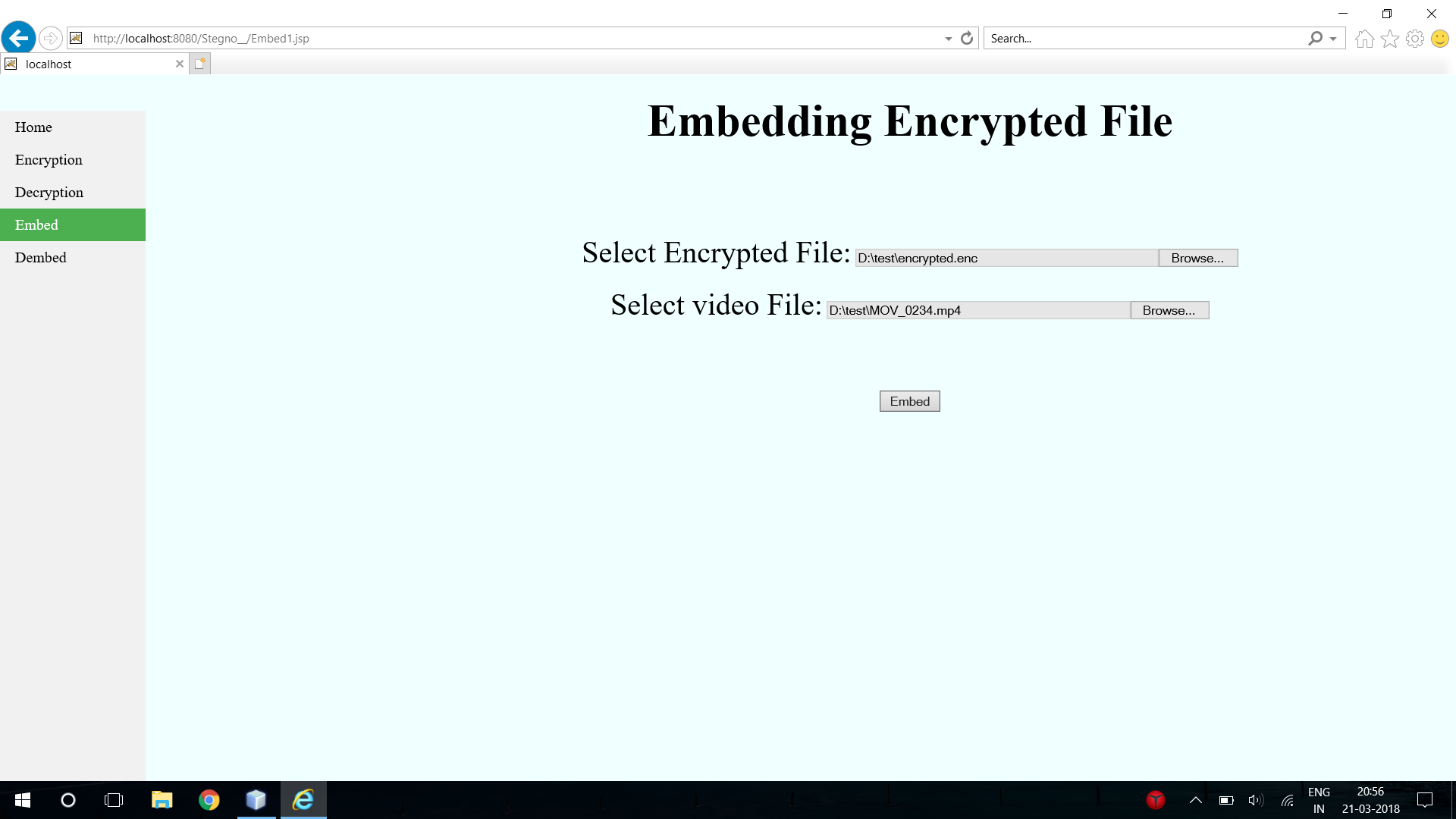
****

Fig:6.4 Embedding Encrypted File

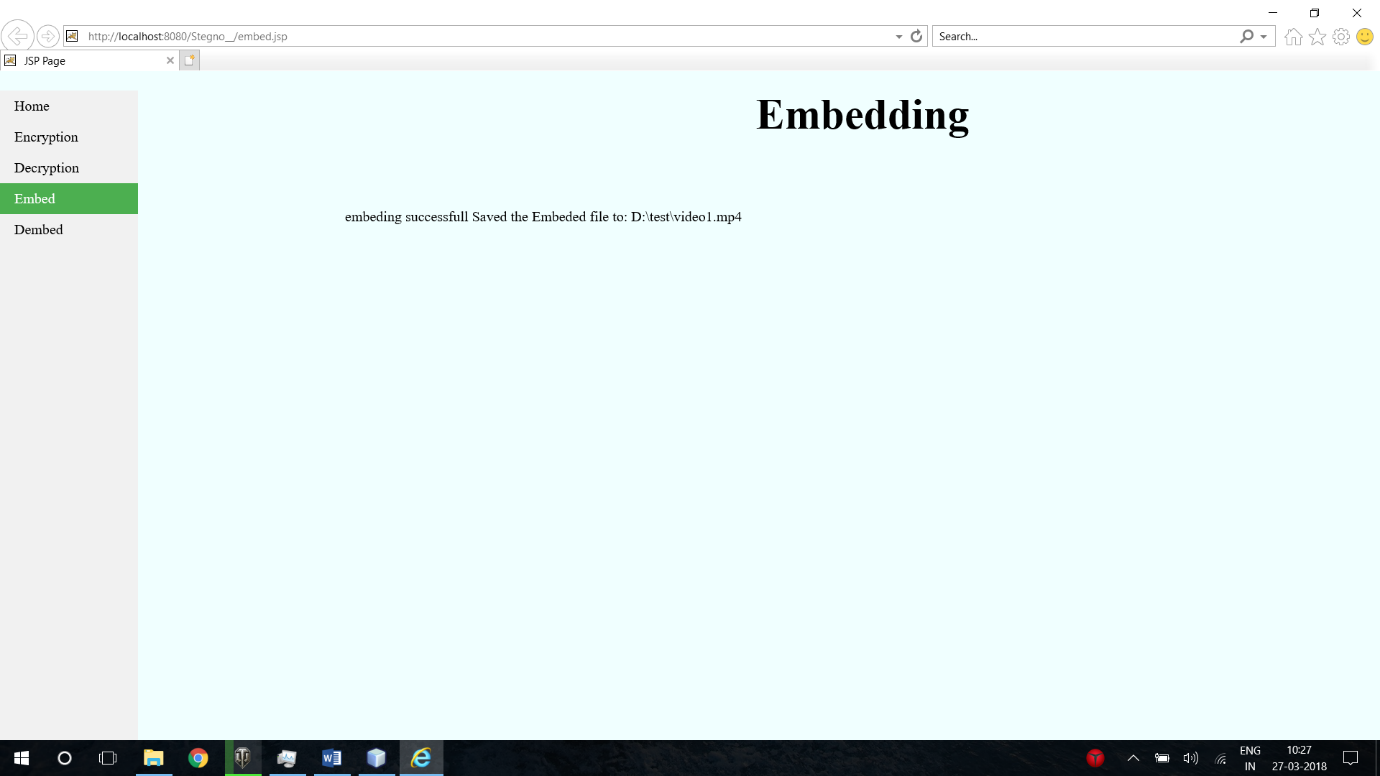
****

Fig:6.5 Embedding Successfull

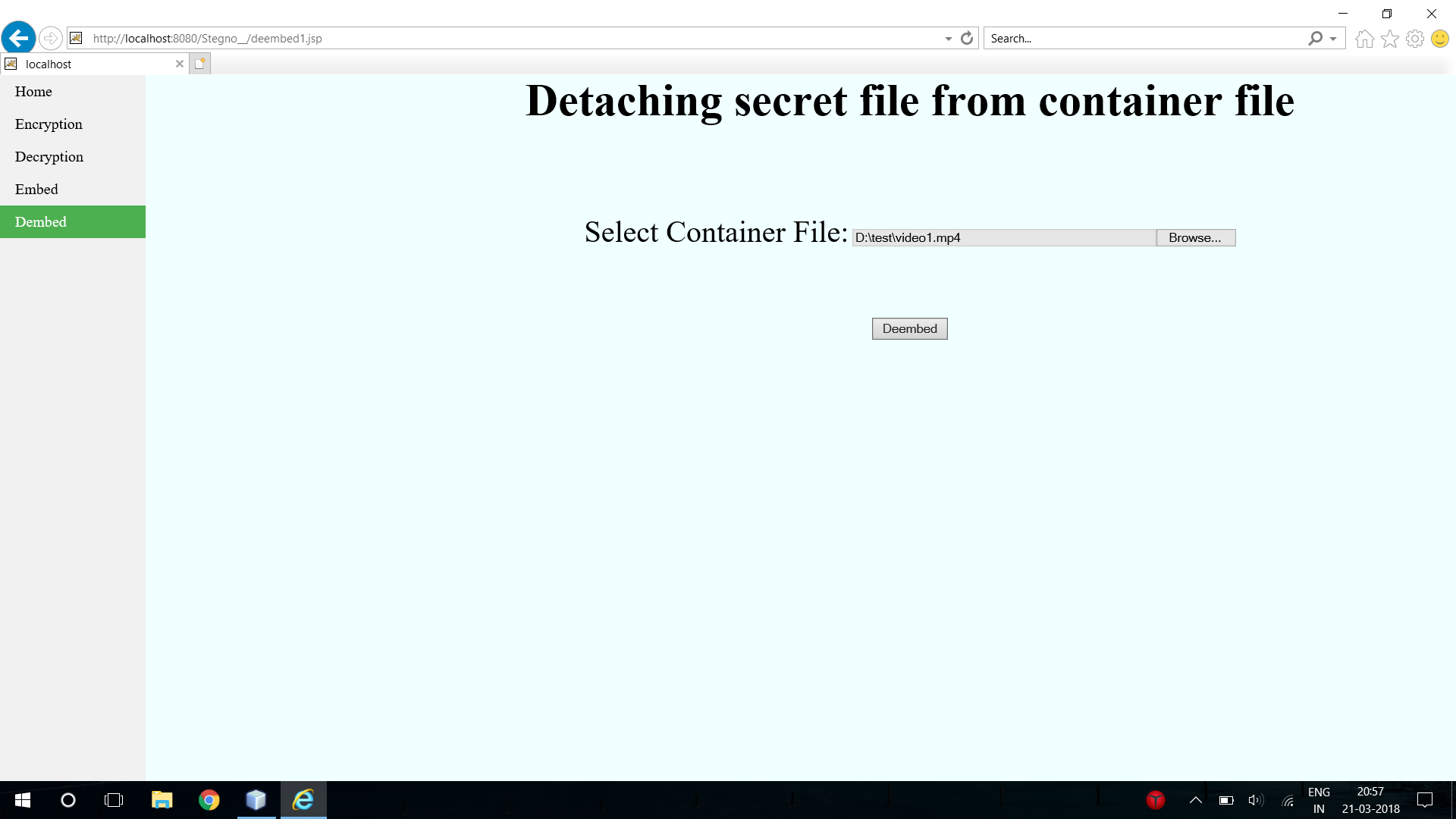
****

Fig:6.6 Detaching secret file from container file

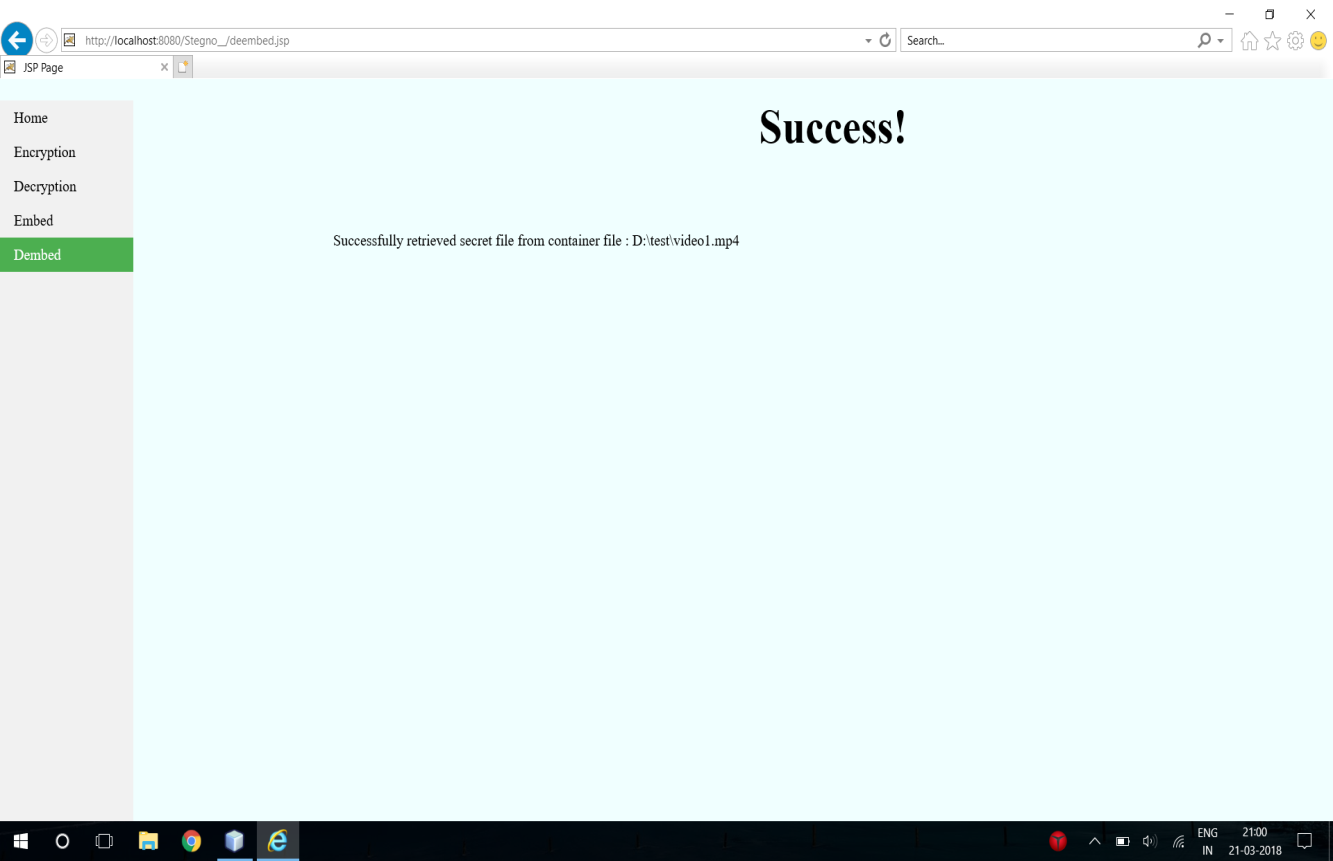
****

Fig:6.7 DeEmbeding Success

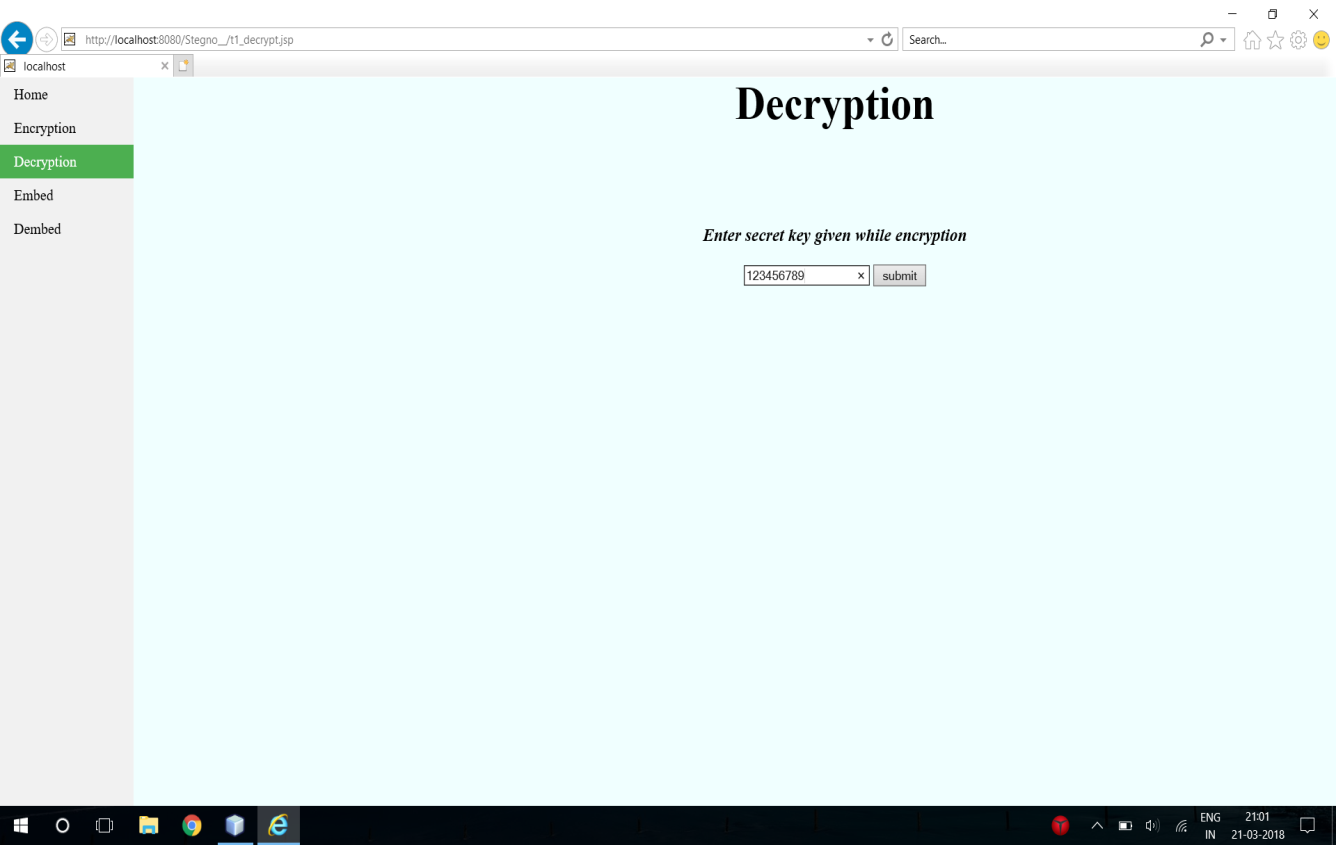
****

Fig:6.8 Decryption

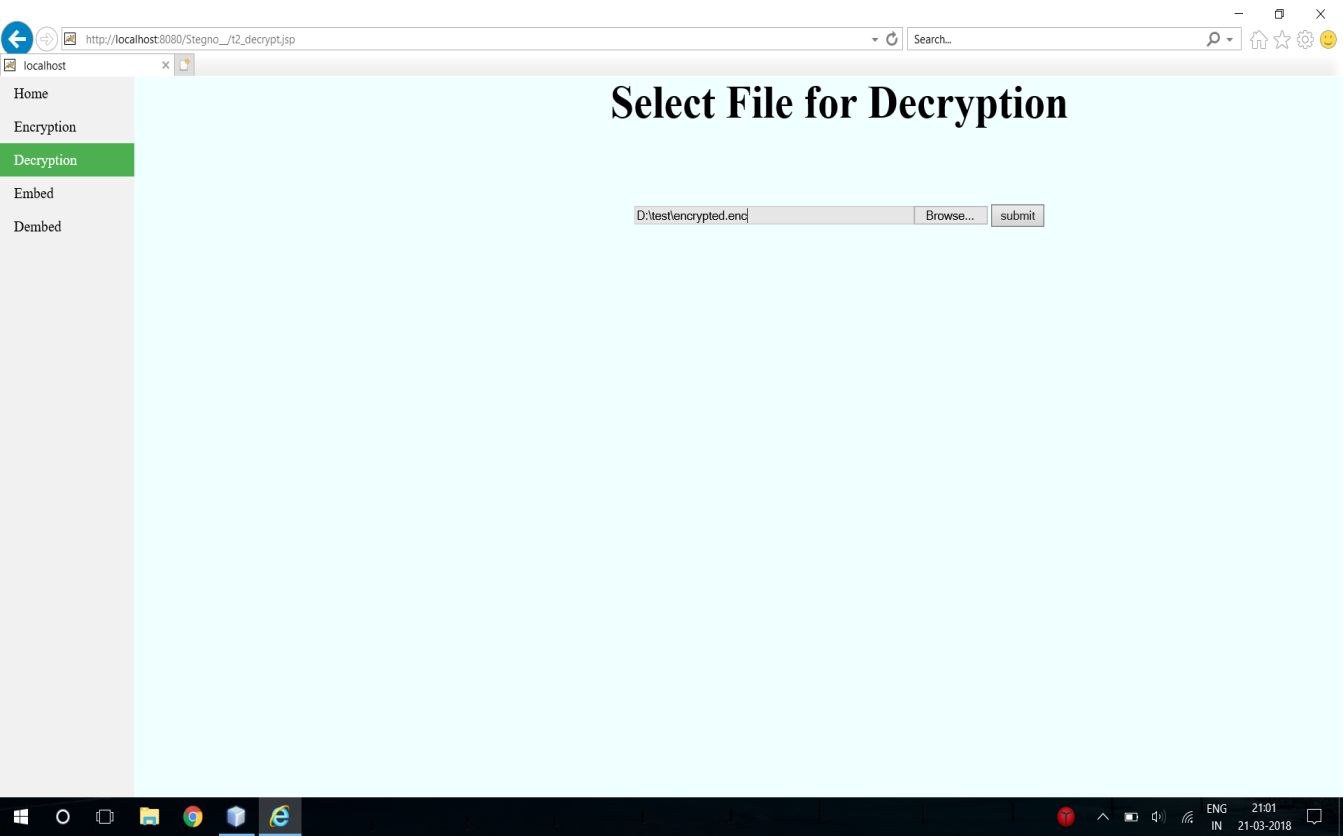
****

Fig:6.9 Select file for Decryption

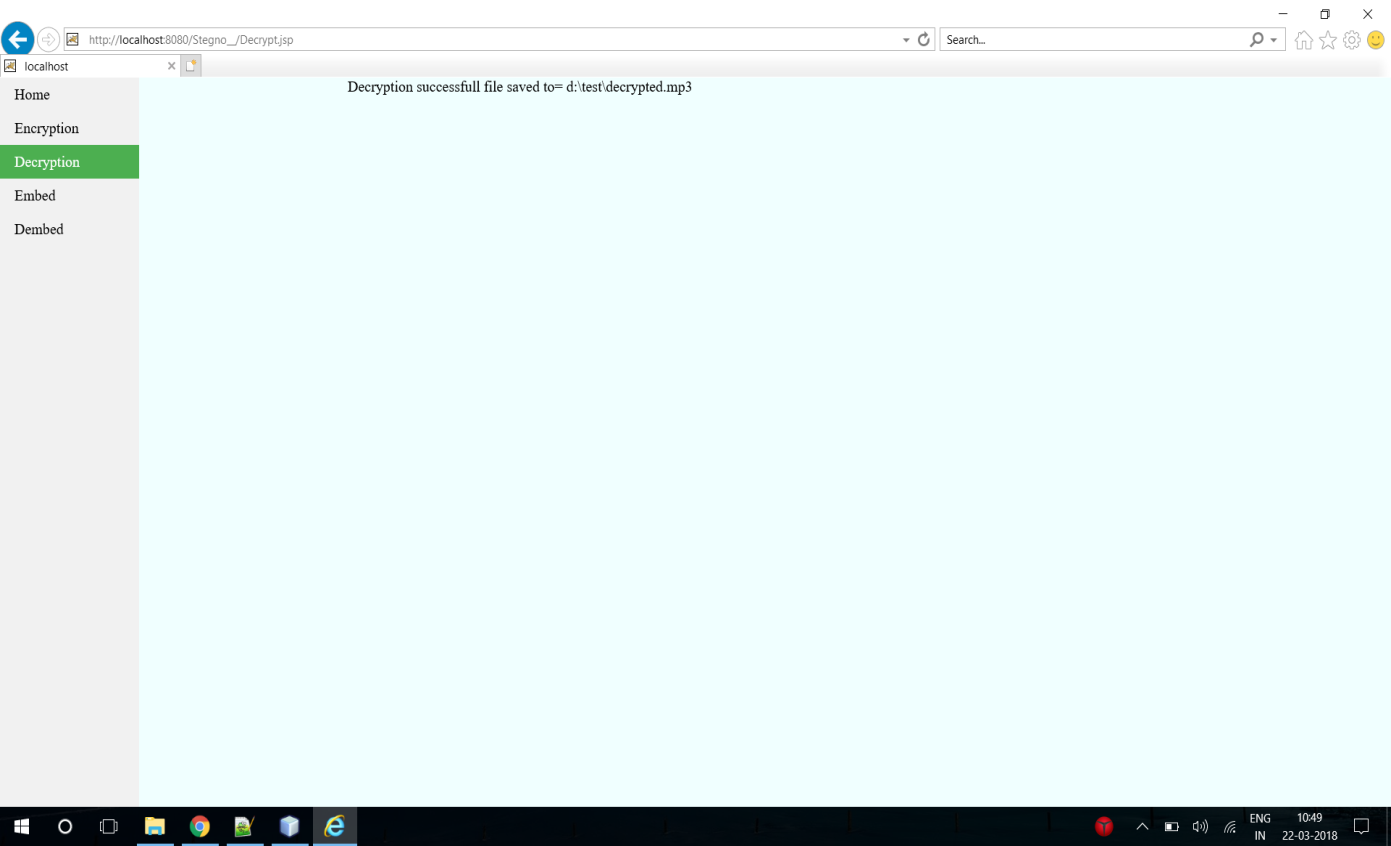
****

Fig:6.10 Decryption successfull file

### 7.SYSTEM TESTING

**7.1 INTRODUCTION**

  The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**7.2 TYPES OF TESTS**

**7.2.1 Unit testing**

         Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**7.2.2 Integration testing**

            Integration tests are designed to test integrated software components to determine if they actually run as one program.  Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at   exposing the problems that arise from the combination of components.

**7.2.3 Functional test**

       Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input               :  identified classes of valid input must be accepted.

Invalid Input              : identified classes of invalid input must be rejected.

Functions                  : identified functions must be exercised.

Output               : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

    Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**7.2.4 System Test**

    System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**7.2.5 White Box Testing**

       White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**7.2.6 Black Box Testing**

       Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**7.2.7 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

#### **7.3 TEST APPROACH**

**Testing can be done in two ways**

1. Bottom up approach
2. Top down approach

**Bottom up Approach**

Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. For each module in bottom up testing a short program executes the module and provides the needed data so that the module is asked to perform the way it will when embedded with in the larger system. When bottom level modules are tested attention turns to those on the next level that use the lower level ones they are tested individually and then linked with the previously examined lower level modules.

**Top down approach**

This type of testing starts from upper level modules. Since the detailed activities usually performed in the lower level routines are not provided stubs are written. A stub is a module shell called by upper level module and that when reached properly will return a message to the calling module indicating that proper interaction occurred. No attempt is made to verify the correctness of the lower level module.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**8.CONCLUSION**

In this paper, we propose a new video data hiding framework that makes use of erasure correction capability of RA codes and superiority of FZDH. The method is also robust to frame manipulation attacks via frame synchronization markers. First, we compare FZDH and QIM as the data hiding method of the proposed framework. We observe that FZDH is superior to QIM, especially for low embedding distortion levels. The framework is tested with MPEG-2, H.264 compression, scaling and frame-rate conversion attacks. Typical system parameters are reported for error-free decoding. The results indicate that the framework can be successfully utilized in video data hiding applications. For instance, Tardos fingerprinting , which is a randomized construction of binary fingerprint codes that are optimal against collusion attack, can be employed within the proposed framework with the following settings. The length of the Tardos fingerprint is 􀜣􀜿􀯢􀬶 ln 􀬵􀰌􀰭 [19], where *A* is a function of false positive probability (􀟝􀬵), false negative probability, and maximum size of colluder coalition, (􀜿􀯢). The minimum segment duration required for Tardos fingerprinting at different operating conditions are given in Table VI. We also compared the proposed framework against the canonical watermarking method, JAWS, and a more recent quantization based method . The results indicate a significant superiority over JAWS and a comparable performance with . The experiments also shed light on possible improvements on the proposed method. Firstly, the framework involves a number of thresholds (*T0*, *T1*, and *T2*), which are determined manually. The range of these thresholds can be analyzed by using a training set. Then some heuristics can be deduced for proper selection of these threshold values. Additionally, incorporation of Human Visual System based spatio-temporally adaptation of data hiding method parameters as in [13] remains as a future direction.

**9.BIBLIOGRAPHY**

**References Made From:**

R. Böhme, “Improved Statistical Steganalysis UsingModels of Heterogeneous Cover Signals,” Ph.D. dissertation, Faculty of Comput. Sci., Technische Universität, Dresden, Germany, 2008.

Y. Kim, Z. Duric, and D. Richards, “Modified matrix encoding technique for minimal distortion steganography,” in *Proc. 8th Int. WorkshopInf. Hiding*, J. L. Camenisch, C. S. Collberg, N. F. Johnson, and

P. Sallee, Eds., Alexandria, VA, Jul. 10–12, 2006, vol. 4437, Lecture Notes in Computer Science, pp. 314–327.

R. Zhang, V. Sachnev, and H. J. Kim, “Fast BCH syndrome coding for steganography,” in *Proc. 11th Int. Workshop Inf. Hiding,*, S. Katzenbeisserand A.-R. Sadeghi, Eds., Darmstadt, Germany, Jun. 7–10, 2009, vol. 5806, Lecture Notes in Computer Science, pp. 31–47.

V. Sachnev, H. J. Kim, and R. Zhang, “Less detectable JPEG steganography method based on heuristic optimization and BCH syndrome coding,” in *Proc. 11th ACM Multimedia Security Workshop*, J. Dittmann, S. Craver, and J. Fridrich, Eds., Princeton, NJ, Sep. 7–8, 2009, pp. 131–140.

T. Pevný, T. Filler, and P. Bas, “Using high-dimensional image models to perform highly undetectable steganography,” in *Proc. 12th Int. Workshop Inf. Hiding*, P. W. L. Fong, R. Böhme, and R. Safavi-Naini, Eds., Calgary, Canada, Jun. 28–30, 2010, vol. 6387, Lecture Notes in Computer Science, pp. 161–177.

J. Kodovský and J. Fridrich, “On completeness of feature spaces in blind steganalysis,” in *Proc. 10th ACMMultimedia SecurityWorkshop*, A. D. Ker, J. Dittmann, and J. Fridrich, Eds., Oxford, U.K., Sep. 22–23, 2008, pp. 123–132.

T. Filler and J. Fridrich, “Gibbs construction in steganography,” *IEEE*

*Trans. Inf. Forensics Security*, vol. 5, pp. 705–720, Sep. 2010.

J. Fridrich and T. Filler, “Practical methods for minimizing embedding

impact in steganography,” in *Proc. SPIE, Electron. Imag., Security, Steganography, Watermark. Multimedia Contents IX*, E. J. Delp and P. W. Wong, Eds., San Jose, CA, Jan. 29–Feb. 1, 2007, vol. 6505, pp. 02–03.

T. Filler and J. Fridrich, “Binary quantization using belief propagation over factor graphs of LDGM codes,” presented at the 45th Annu. Allerton Conf. Commun., Control, Comput., Allerton, IL, Sep. 26–28, 2007.

X. Zhang, W. Zhang, and S. Wang, “Efficient double-layered steganographic embedding,” *Electron. Lett.*, vol. 43, pp. 482–483, Apr. 2007.

W. Zhang, S. Wang, and X. Zhang, “Improving embedding efficiency of covering codes for applications in steganography,” *IEEE Commun. Lett.*, vol. 11, pp. 680–682, Aug. 2007.

W. Zhang, X. Zhang, and S. Wang, “Maximizing steganographic embedding efficiency by combining Hamming codes and wet paper codes,” in *Proc. 10th Int. Workshop Inf. Hiding,* , K. Solanki, K. Sullivan, and U. Madhow, Eds., Santa Barbara, CA, Jun. 19–21, 2008, vol. 5284, Lecture Notes in Computer Science, pp. 60–71.

T. Filler and J. Fridrich, “Wet ZZW construction for steganography,” presented at the 1st IEEE Int. Workshop Inf. Forensics Security, London, U.K., Dec. 6–9, 2009.

W. Zhang and X. Zhu, “Improving the embedding efficiency of wet

paper codes by paper folding,” *IEEE Signal Process. Lett.*, vol. 16, pp.

794–797, Sep. 2009.

W. Zhang and X. Wang, “Generalization of the ZZW embedding construction for steganography,” *IEEE Trans. Inf. Forensics Security*, vol. 4, pp. 564–569, Sep. 2009.

J. Fridrich, T. Pevný, and J. Kodovský, “Statistically undetectable JPEG steganography: Dead ends, challenges, and opportunities,” in *Proc. 9th ACM Multimedia Security Workshop*, J. Dittmann and J. Fridrich, Eds., Dallas, TX, Sep. 20–21, 2007, pp. 3–14.

**10. REFERENCES**

<http://java.sun.com>

<http://www.sourcefordgde.com>

<http://www.networkcomputing.com/>

<http://www.roseindia.com/>

<http://www.java2s.com/>

http://www.javadb.com/