A Privacy-Preserving and Untraceable Group Data Sharing Scheme in Cloud computing

OBJECTIVE:

We propose a novel privacy-preserving mechanism that supports public auditing on shared data stored in the cloud. Recent studies have been worked to promote the cloud computing evolve towards the internet of services. Subsequently, security and privacy issues are becoming key concerns with the increasing popularity of cloud services.

DOMAIN: Cloud computing

Abstract and introduction:

In this paper, we propose a novel privacy-preserving mechanism that supports public auditing on shared data stored in the cloud. In particular, we exploit ring signatures to compute verification metadata needed to audit the correctness of shared data. With our mechanism, the identity of the signer on each block in shared data is kept private from public verifiers, who are able to efficiently verify shared data integrity without retrieving the entire file. In addition, our mechanism is able to perform multiple auditing tasks simultaneously instead of verifying them one by one.

The propose system, a privacy-preserving public auditing mechanism for shared data in the cloud. We utilize ring signatures to construct homomorphism authenticators, so that a public verifier is able to audit shared data integrity without retrieving the entire data, yet it cannot distinguish who is the signer on each block. To improve the efficiency of verifying multiple auditing tasks, we further extend our mechanism to support batch auditing. There are two interesting problems we will continue to study
for our future work. One of them is traceability, which means the ability for the group manager to reveal the identity of the signer based on verification metadata in some special situations.

AES is associate degree unvarying instead of Feistel cipher. It’s supported ‘substitution–permutation network’. It contains of a series of joined operations, a number of that involve exchange inputs by specific outputs and other involve shuffling bits around.

Interestingly, AES performs all its computations on bytes instead of bits. Hence, AES treats the 128 bits of a plaintext block as sixteen bytes. These sixteen bytes square measure organized in four columns and 4 rows for process as a matrix

**EXISTING SYSTEM:**

The existing mechanism a new significant privacy issue introduced in the case of shared data with the use of the leakage of identity privacy to public verifiers. The traditional approach for checking data correctness is to retrieve the entire data from the cloud, and then verify data integrity by checking the correctness of signatures.

To securely introduce an effective third party auditor (TPA), the following two fundamental requirements have to be met: 1) TPA should be able to efficiently audit the cloud data storage without demanding the local copy of data, and introduce no additional on-line burden to the cloud user; 2) The third party auditing process should bring in no new vulnerabilities towards user data privacy

**LIMITATIONS**

- As users no longer physically possess the storage of their data, traditional cryptographic primitives for the purpose of data security protection cannot be directly adopted.
- They do not perform the multiple auditing tasks in simultaneously.
- Loss of data’s.
● Does not provide any privacy for private data’s.
● Authentication time takes too long.

PROPOSED SYSTEM:

The propose system, a privacy-preserving public auditing mechanism for shared data in the cloud. We utilize ring signatures to construct homomorphism authenticators, so that a public verifier is able to audit shared data integrity without retrieving the entire data, yet it cannot distinguish who is the signer on each block.

To improve the efficiency of verifying multiple auditing tasks, we further extend our mechanism to support batch auditing. There are two interesting problems we will continue to study for our future work. One of them is traceability, which means the ability for the group manager to reveal the identity of the signer based on verification metadata in some special situations.

Stands for "Simple Mail Transfer Protocol." this can be the protocol used for causation e-mail over the web. Your e-mail shopper uses SMTP to send a message to the mail server, and also the mail server uses SMTP to relay that message to the proper receiving mail server. Basically, SMTP could be a set of commands that certify and direct the transfer of electronic message. Once configuring the settings for your e-mail program, you always ought to set the SMTP server to your native net Service Provider's SMTP settings. However, the incoming mail server (IMAP or POP3) ought to be set to your mail account's server, which can differ than the SMTP server.

ADVANTAGES:

● The proposed system can perform multiple auditing tasks simultaneously
● They improve the efficiency of verification for multiple auditing tasks.
● High security provide for file sharing.
● Admin has control deleting users
● Users can send request to auditor.
SYSTEM ARCHITECTURE:

HARDWARE AND SOFTWARE SPECIFICATION:

Software Requirement:

1. Language - Java(JDK 1.7)
2. OS - Windows 7 32bit
3. MySql Server
4. NetBeans IDE 7.1.2

Hardware Requirement:

1. 1 GB RAM
2. 80 GB Hard Disk
3. Above 2GHz Processor
4. Data Card
Module:

1. User Registration:

For the registration of user with identity ID the group manager randomly selects a number. Then the group manager adds into the group user list which will be used in the traceability phase. After the registration, user obtains a private key which will be used for group signature generation and file decryption.

![Registration Key Distribution Diagram]

2. Public Auditing:

Homomorphic authenticators are unforgeable verification metadata generated from individual data blocks, which can be securely aggregated in such a way to assure an auditor that a linear combination of data blocks is correctly computed by verifying only the aggregated authenticator. Overview to achieve privacy-preserving public auditing, we propose to uniquely integrate the Homomorphic authenticator with random mask technique. In our protocol, the linear combination of sampled blocks in the server’s response is masked with randomness generated by a pseudo random function (PRF). The proposed scheme is as follows:

- Setup Phase
- Audit Phase
3. Sharing Data:

The canonical application is data sharing. The public auditing property is especially useful when we expect the delegation to be efficient and flexible. The schemes enable a content provider to share her data in a confidential and selective way, with a fixed and small ciphertext expansion, by distributing to each authorized user a single and small aggregate key.

4. Integrity Checking:

Hence, supporting data dynamics for privacy-preserving public risk auditing is also of paramount importance. Now we show how our main scheme can be adapted to build upon the existing work to support data dynamics, including block level operations of modification, deletion and insertion. We can adopt this technique in our design to achieve privacy-preserving public risk auditing with support of data dynamics. The user download the particular file not download entire file.
LITERATURE SURVEY:


We consider the problem of constructing an erasure code for storage over a network when the data sources are distributed. Specifically, we assume that there are n storage nodes with limited memory and k < n sources generating the data. We want a data collector, who can appear anywhere in the network, to query any k storage nodes and be able to retrieve the data. We introduce Decentralized Erasure Codes, which are linear codes with a specific randomized structure inspired by network coding on random bipartite graphs. We show that decentralized erasure codes are optimally sparse, and lead to reduced communication, storage and computation cost over random linear coding.

2. Repair Locality from a Combinatorial Perspective.
Author:Anyu Wang and Zhifang ZhangKey Laboratory of Mathematics Mechanization, IEEE Dec.2014.

Plutus is a cryptographic storage system that enables secure file sharing without placing much trust on the file servers. In particular, it makes novel use of cryptographic primitives to protect and share files. Plutus features highly scalable key management while allowing individual users to retain direct control over who gets access to their files. We explain the mechanisms in Plutus to reduce the number of cryptographic keys exchanged between users by using file groups, distinguish file read and write access, handle user revocation efficiently, and allow an untrusted server to authorize file writes. We have built a prototype of Plutus on OpenAFS. Measurements of this prototype show that
Plutus achieves strong security with overhead comparable to systems that encrypt all network traffic.

3. On the Effective Parallel Programming of Multi-core Processors.
   Author: Prof.dr.ir. H.J. Sips Technische Universities Delft, promotor
   Prof.dr.ir. A.J.C. van Gemund Technische Universities Delft Prof.dr.ir. H.E.
   Bal. 7 December 2010.

Availability is a storage system property that is both highly desired and yet minimally engineered. While many systems provide mechanisms to improve availability—such as redundancy and failure recovery—how to best configure these mechanisms is typically left to the system manager. Unfortunately, few individuals have the skills to properly manage the trade-offs involved, let alone the time to adapt these decisions to changing conditions. Instead, most systems are configured statically and with only a cursory understanding of how the configuration will impact overall performance or availability. While this issue can be problematic even for individual storage arrays, it becomes increasingly important as systems are distributed—and absolutely critical for the wide area peer-to-peer storage infrastructures being explored. This paper describes the motivation, architecture and implementation for a new peer-to-peer storage system, called Total Recall that automates the task of availability management. In particular, the Total Recall system automatically measures and estimates the availability of its constituent host components, predicts their future availability based on past behavior, calculates the appropriate redundancy mechanisms and repair policies, and delivers user-specified availability while maximizing efficiency.
4. Parallel Reed/Solomon Coding on Multicore Processors.

Author: Peter Sobs  
Institute of Computer Engineering University of Luebeck, Germany. 2010 EEE DOI 10.1109/SNAPI.2010.16

This paper sketches the design of PAST, a large-scale, Internet-based, global storage utility that provides scalability, high availability, persistence and security. PAST is a peer-to-peer Internet application and is entirely selforgaining. PAST nodes serve as access points for clients, participate in the routing of client requests, and contribute storage to the system. Nodes are not trusted, they may join the system at any time and may silently leave the system without warning. Yet, the system is able to provide strong assurances, efficient storage access, load balancing and scalability.
DFD Level 0:

User

Cloud

Upload

Public auditing

Download file

Retrieve original data
DFD 1:

User

Cloud

Encrypt file

Upload file

Retrieve file

Server

Public verifier

Public Audit

Retrieve the file
DFD 2:

User

Cloud

Retrieve original file

Encrypt file

Upload file

Server

Public verifier

Public Audit

Homomorphic signature

Download file
USE CASE DIAGRAM:

A Use-Case is a description of a systems behavior from a users stand point. For system developer this is a valuable tool: it’s a tried-and-true technique for gathering system requirements from a user’s point of view. That is important if the goal is to build a system that real people can use. A little stick figure is used to identify an actor the ellipse represents use-case.
CLASS DIAGRAM:

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP). The concept is several years old but has been refined as OOP modeling paradigms have evolved.
SEQUENCE DIAGRAM:

IT shows the interaction between a set of objects, through the messages that may be dispatched between them.

The diagrams consists of interacting objects and actors, with messages in-between them it is common to focus the model on scenarios specified by use-cases. It is also often useful input to the detailed class diagram to try to model the specified
COLLABRATION DIAGRAM:

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML).

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.
**ACTIVITY DIAGRAM:**

It shows the flow through a program from an defined start point to an end point. Activity diagrams describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel. Basic elements in activity diagrams are activities, branches (conditions or selections), transitions, forks and joins.

Activity diagrams should be used in conjunction with other modelling techniques such as interaction diagrams and state diagrams. The main reason to use activity diagrams is to model the workflow behind the system being designed. Activity Diagrams are also useful for: analysing a use case by describing what actions need to take place and when they should occur; describing a complicated sequential algorithm; and modelling applications with parallel processes. Activity diagrams do not give detail about how objects behave or how objects collaborate.
Java (programming language)

History

The JAVA language was created by James Gosling in June 1991 for use in a set top box project. The language was initially called Oak, after an oak tree that stood outside Gosling's office - and also went by the name Green - and ended up later being renamed to Java, from a list of random words. Gosling's goals were to implement a virtual machine and a language that had a familiar C/C++ style of notation. The first public implementation was Java 1.0 in 1995. It promised "Write Once, Run Anywhere" (WORA), providing no-cost runtimes on popular platforms. It was fairly secure and its security was configurable, allowing network and file access to be restricted. Major web browsers soon incorporated the ability to run secure Java applets within web pages. Java quickly became popular. With the advent of Java 2, new versions had multiple configurations built for different types of platforms. For example, J2EE was for enterprise applications and the greatly stripped down version J2ME was for mobile applications. J2SE was the designation for the Standard Edition. In 2006, for marketing purposes, new J2 versions were renamed Java EE, Java ME, and Java SE, respectively.

In 1997, Sun Microsystems approached the ISO/IEC JTC1 standards body and later the Ecma International to formalize Java, but it soon withdrew from the process. Java remains a standard that is controlled through the Java Community Process. At one time, Sun made most of its Java implementations available without charge although they were proprietary software. Sun's revenue from Java was generated by the selling of licenses for specialized products such as the Java Enterprise System. Sun distinguishes between its Software Development Kit (SDK) and Runtime Environment (JRE) which is a subset of the SDK, the primary distinction being that in the JRE, the compiler, utility programs, and many necessary header files are not present.
On 13 November 2006, Sun released much of Java as free software under the terms of the GNU General Public License (GPL). On 8 May 2007, Sun finished the process, making all of Java's core code open source, aside from a small portion of code to which Sun did not hold the copyright.

**Primary goals**

There were five primary goals in the creation of the Java language:

- It should use the object-oriented programming methodology.
- It should allow the same program to be executed on multiple operating systems.
- It should contain built-in support for using computer networks.
- It should be designed to execute code from remote sources securely.
- It should be easy to use by selecting what were considered the good parts of other object-oriented languages.

**The Java Programming Language:**

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

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance

Each of the preceding buzzwords is explained in *The Java Language Environment*, a white paper written by James Gosling and Henry McGilton.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler.
A .class file does not contain code that is native to your processor; it instead contains byte codes — the machine language of the Java Virtual Machine\(^1\) (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

An overview of the software development process.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris\(^\text{TM}\) Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the Java Hot Spot virtual machine perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.
Through the Java VM, the same application is capable of running on multiple platforms.

**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 Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying 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*

The *Java Application Programming Interface (API)*

You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What CanJavaTechnologyDo? Highlights some of the functionality provided by the API.

The API and Java Virtual Machine insulate the program from the underlying hardware.
As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

**Java Runtime Environment**

The Java Runtime Environment, or JRE, is the software required to run any application deployed on the Java Platform. End-users commonly use a JRE in software packages and Web browser plug-in. Sun also distributes a superset of the JRE called the Java 2 SDK (more commonly known as the JDK), which includes development tools such as the Java compiler, Javadoc, Jar and debugger.

One of the unique advantages of the concept of a runtime engine is that errors (exceptions) should not 'crash' the system. Moreover, in runtime engine environments such as Java there exist tools that attach to the runtime engine and every time that an exception of interest occurs they record debugging information that existed in memory at the time the exception was thrown (stack and heap values). These Automated Exception Handling tools provide 'root-cause' information for exceptions in Java programs that run in production, testing or development environments.

**Uses OF JAVA**

Blue is a smart card enabled with the secure, cross-platform, object-oriented Java Card API and technology. Blue contains an actual on-card processing chip, allowing for enhance able and multiple functionality within a single card. Applets that comply with the Java Card API specification can run on any third-party vendor card that provides the necessary Java Card Application Environment (JCAE). Not only can multiple applet programs run on a single card, but new applets and functionality can be added after the card is issued to the customer

- Java Can be used in Chemistry.
- In NASA also Java is used.
- In 2D and 3D applications java is used.
- In Graphics Programming also Java is used.
- In Animations Java is used.
- In Online and Web Applications Java is used.

**JSP:**

**JavaServer Pages (JSP)** is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request. The technology allows Java code and certain pre-defined actions to be embedded into static content.

The JSP syntax adds additional XML-like tags, called JSP actions, to be used to invoke built-in functionality. Additionally, the technology allows for the creation of JSP tag libraries that act as extensions to the standard HTML or XML tags. Tag libraries provide a platform independent way of extending the capabilities of a Web server.

JSPs are compiled into Java Servlet by a JSP compiler. A JSP compiler may generate a servlet in Java code that is then compiled by the Java compiler, or it may generate byte code for the servlet directly. JSPs can also be interpreted on-the-fly reducing the time taken to reload changes.

JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic web content. JSP technology enables rapid development of web-based applications that are server and platform-independent.
The Advantages of JSP

Active Server Pages (ASP). ASP is a similar technology from Microsoft. 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. Pure Servlet. JSP doesn't give you anything that you couldn't in principle do with a Servlet. But it is more convenient to write (and to modify!) regular HTML than to have a zillion println statements that generate the HTML. Plus, by separating the look from the content you can put different people on different tasks: your Web page design experts can build the HTML, leaving places for your Servlet programmers to insert the dynamic content.

Server-Side Includes (SSI). SSI is a widely-supported technology for including externally-defined pieces into a static Web page. JSP is better because it lets you use Servlet instead of a separate program to generate that dynamic part. Besides, SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
JavaScript. JavaScript can generate HTML dynamically on the client. This is a useful capability, but only handles situations where the dynamic information is based on the client's environment.

With the exception of cookies, HTTP and form submission data is not available to JavaScript. And, since it runs on the client, JavaScript can't access server-side resources like databases, catalogs, pricing information, and the like. Static HTML. Regular HTML, of course, cannot contain dynamic information. JSP is so easy and convenient that it is quite feasible to augment HTML pages that only benefit marginally by the insertion of small amounts of dynamic data. Previously, the cost of using dynamic data would preclude its use in all but the most valuable instances.

ARCHITECTURE OF JSP

- The browser sends a request to a JSP page.
- The JSP page communicates with a Java bean.
- The Java bean is connected to a database.
- The JSP page responds to the browser.
SERVLETS – FRONT END

The Java Servlet API allows a software developer to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlet are the Java counterpart to non-Java dynamic Web content technologies such as PHP, CGI and ASP.NET. Servlet can maintain state across many server transactions by using HTTP cookies, session variables or URL rewriting.

The Servlet API, contained in the Java package hierarchy javax.Servlet, defines the expected interactions of a Web container and a Servlet. A Web container is essentially the component of a Web server that interacts with the Servlet. The Web container is responsible for managing the lifecycle of Servlet, mapping a URL to a particular Servlet and ensuring that the URL requester has the correct access rights.

A Servlet is an object that receives a request and generates a response based on that request. The basic Servlet package defines Java objects to represent Servlet requests and responses, as well as objects to reflect the Servlet configuration parameters and execution environment. The package javax.Servlet.Http defines HTTP-specific subclasses of the generic Servlet elements, including session management objects that track multiple requests and responses between the Web server and a client. Servlet may be packaged in a WAR file as a Web application.

Servlet can be generated automatically by Java Server Pages (JSP), or alternately by template engines such as Web Macro. Often Servlet are used in conjunction with JSPs in a pattern called "Model 2", which is a flavour of the model-view-controller pattern.

Servlet are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages. Building Web pages on the fly is useful (and commonly done) for a number of reasons.
The Web page is based on data submitted by the user. For example the results pages from search engines are generated this way, and programs that process orders for e-commerce sites do this as well. The data changes frequently. For example, a weather-report or news headlines page might build the page dynamically, perhaps returning a previously built page if it is still up to date. The Web page uses information from corporate databases or other such sources. For example, you would use this for making a Web page at an on-line store that lists current prices and number of items in stock.

**The Servlet Run-time Environment**

A Servlet is a Java class and therefore needs to be executed in a Java VM by a service we call a Servlet engine. The Servlet engine loads the servlet class the first time the Servlet is requested, or optionally already when the Servlet engine is started. The Servlet then stays loaded to handle multiple requests until it is explicitly unloaded or the Servlet engine is shut down.

Some Web servers, such as Sun's Java Web Server (JWS), W3C's Jigsaw and Gefion Software's Lite Web Server (LWS) are implemented in Java and have a built-in Servlet engine. Other Web servers, such as Netscape's Enterprise Server, Microsoft's Internet Information Server (IIS) and the Apache Group's Apache, require a Servlet engine add-on module. The add-on intercepts all requests for Servlet, executes them and returns the response through the Web server to the client. Examples of Servlet engine add-ons are Gefion Software's WAI Cool Runner, IBM's Web Sphere, Live Software's JRun and New Atlanta's Servlet Exec.

All Servlet API classes and a simple Servlet-enabled Web server are combined into the Java Servlet Development Kit (JSDK), available for download at Sun's official Servlet site. To get started with Servlet I recommend that you download the JSDK and play around with the sample Servlet.
Life Cycle OF Servlet

- The Servlet lifecycle consists of the following steps:
  - The Servlet class is loaded by the container during start-up.

  The container calls the init() method. This method initializes the Servlet and must be called before the Servlet can service any requests. In the entire life of a Servlet, the init() method is called only once. After initialization, the Servlet can service client-requests.

  Each request is serviced in its own separate thread. The container calls the service() method of the Servlet for every request.

  The service() method determines the kind of request being made and dispatches it to an appropriate method to handle the request. The developer of the Servlet must provide an implementation for these methods. If a request for a method that is not implemented by the Servlet is made, the method of the parent class is called, typically resulting in an error being returned to the requester. Finally, the container calls the destroy() method which takes the Servlet out of service. The destroy() method like init() is called only once in the lifecycle of a Servlet.

- Request and Response Objects

  The do Get method has two interesting parameters: HttpServletRequest and HttpServletResponse. These two objects give you full access to all information about the request and let you control the output sent to the client as the response to the request. With CGI you read environment variables and stdin to get information about the request, but the names of the environment variables may vary between implementations and some are not provided by all Web servers.

  The HttpServletRequest object provides the same information as the CGI environment variables, plus more, in a standardized way. It also provides methods for extracting HTTP parameters from the query string or the request body depending on the type of request (GET or POST). As a Servlet developer you access parameters the same way for both types of requests. Other methods give you access to all request headers and help you parse date and cookie headers.
Instead of writing the response to stdout as you do with CGI, you get an OutputStream or a PrintWriter from the HttpServletResponse. The OutputStream is intended for binary data, such as a GIF or JPEG image, and the PrintWriter for text output. You can also set all response headers and the status code, without having to rely on special Web server CGI configurations such as Non Parsed Headers (NPH). This makes your Servlet easier to install.

**ServletConfig and Servlet Context:**

There is only one Servlet Context in every application. This object can be used by all the Servlet to obtain application level information or container details. Every Servlet, on the other hand, gets its own ServletConfig object. This object provides initialization parameters for a servlet. A developer can obtain the reference to Servlet Context using either the ServletConfig object or Servlet Request object.

All servlets belong to one servlet context. In implementations of the 1.0 and 2.0 versions of the Servlet API all servlets on one host belongs to the same context, but with the 2.1 version of the API the context becomes more powerful and can be seen as the humble beginnings of an Application concept. Future versions of the API will make this even more pronounced.

Many servlet engines implementing the Servlet 2.1 API let you group a set of servlets into one context and support more than one context on the same host. The Servlet Context in the 2.1 API is responsible for the state of its servlets and knows about resources and attributes available to the servlets in the context. Here we will only look at how Servlet Context attributes can be used to share information among a group of servlets.

There are three Servlet Context methods dealing with context attributes: getAttribute, setAttribute and removeAttribute. In addition the servlet engine may provide ways to configure a servlet context with initial attribute values. This serves as a welcome addition to the servlet initialization arguments for configuration information used by a group of servlets, for instance the database identifier we talked about above, a style sheet URL for an application, the name of a mail server, etc.
JDBC

Java Database Connectivity (JDBC) is a programming framework for Java developers writing programs that access information stored in databases, spreadsheets, and flat files. JDBC is commonly used to connect a user program to a "behind the scenes" database, regardless of what database management software is used to control the database. In this way, JDBC is cross-platform. This article will provide an introduction and sample code that demonstrates database access from Java programs that use the classes of the JDBC API, which is available for free download from Sun's site.

A database that another program links to is called a data source. Many data sources, including products produced by Microsoft and Oracle, already use a standard called Open Database Connectivity (ODBC). Many legacy C and Perl programs use ODBC to connect to data sources. ODBC consolidated much of the commonality between database management systems. JDBC builds on this feature, and increases the level of abstraction. JDBC-ODBC bridges have been created to allow Java programs to connect to ODBC-enabled database software.

JDBC Architecture

Two-tier and Three-tier Processing Models

The JDBC API supports both two-tier and three-tier processing models for database access.
In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user.

MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

![Diagram of three-tier architecture](image)

Until recently, the middle tier has often been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java byte code into efficient machine-specific code and
technologies such as Enterprise JavaBeans™, the Java platform is fast becoming the standard platform for middle-tier development. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features.

With enterprises increasingly using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of a three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed transactions, and disconnected rowsets. The JDBC API is also what allows access to a data source from a Java middle tier.
Coding:

Database

/*

SQLyog - Free MySQL GUI v5.13

Host - 5.0.22-community-nt : Database - erasurecode

*********************************************************************

Server version : 5.0.22-community-nt

*/

SET NAMES utf8;

SET SQL_MODE='';

create database if not exists `erasurecode`;

USE `erasurecode`;

SET@OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO';

/*Table structure for table `admin` */

DROP TABLE IF EXISTS `admin`;

CREATE TABLE `admin` (
  `username` varchar(30) default NULL,
  `password` varchar(30) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*Data for the table `admin` */

insert into `admin` (`username`, `password`) values ('deva','deva');

/*Table structure for table `gender` */

DROP TABLE IF EXISTS `gender`;

CREATE TABLE `gender` (  
   `Gender` varchar(20) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `gender` */

insert into `gender` (`Gender`) values ('Male');

insert into `gender` (`Gender`) values ('Female');

/*Table structure for table `ownerprofile` */

DROP TABLE IF EXISTS `ownerprofile`;

CREATE TABLE `ownerprofile` (  
   `UserName` varchar(35) default NULL,  
   `groupname` varchar(35) default NULL,  
   `filename` varchar(700) NOT NULL,  
   `server1` varchar(100) default NULL,  
   `server2` varchar(100) default NULL,
)
'server3' varchar(100) default NULL,

'server4' varchar(100) default NULL,

PRIMARY KEY ('filename')

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

/*Data for the table `ownerprofile` */

Insert into `ownerprofile`
(`UserName`,`groupname`,`filename`,`server1`,`server2`,`server3`,`server4`) values
('ram','ram','3.jpg','ef3cad6eee251f670014365fde992889e73f1a8e',"ef3cad6eee251f670014365fde992889e73f1a8ekey1",'ef3cad6eee251f670014365fde992889e73f1a8ekey2','ef3cad6eee251f670014365fde992889e73f1a8ekey3);

/*Table structure for table `registration` */

DROP TABLE IF EXISTS `registration`;

CREATE TABLE `registration` (  
    'username' varchar(30) default NULL,
    'password' varchar(30) default NULL,
    'gender' varchar(20) default NULL,
    'email' varchar(50) default NULL,
    'phoneno' varchar(50) default NULL,
    'userproductkey' varchar(50) default NULL
)
/*Data for the table `registration` */

insert into `registration`
(`username`, `password`, `gender`, `email`, `phoneno`, `userproductkey`) values
('ram','ram','male','deva@gamil.com','1234','asdfasdfsdfasdfsdf');

insert into `registration`
(`username`, `password`, `gender`, `email`, `phoneno`, `userproductkey`) values ('deva',NULL,NULL,NULL,NULL,NULL);

insert into `registration`
(`username`, `password`, `gender`, `email`, `phoneno`, `userproductkey`) values ('kumar',NULL,NULL,NULL,NULL,NULL);

SET SQL_MODE=@OLD_SQL_MODE;

**Admin Servlet Code:**

*/

package com.ErasureCode;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.*;
import javax.servlet.http.*;

/**
 * @author Admin
 * *
 * @author Admin
 */

public class AdminServlet extends HttpServlet {

    private Object out;

    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
     * methods.
     */

}
protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

try (PrintWriter out = response.getWriter()) {

/* TODO output your page here. You may use following sample code. */

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet AdminServlet</title>");

out.println("</head>");

out.println("<body>");

out.println("<h1>Servlet AdminServlet at " + request.getContextPath() + "</h1>");

out.println("</body>");

}
out.println("</html>");

}
/**
 * Handles the HTTP <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 *
 * @Override
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {

 HttpSession session1=request.getSession();
 Connection con=null;
 Statement st=null;
 ResultSet rs=null;

 try(PrintWriter out = response.getWriter())
  {
String Username=request.getParameter("username");

String Password=request.getParameter("password");

System.out.println("this line my cheking========"+Username);

System.out.println("this line my cheking========"+Password);

Class.forName("com.mysql.jdbc.Driver");

con=DriverManager.getConnection("jdbc:mysql://localhost:3306/imageselectionerasurecord","root","password");

st=con.createStatement();

rs=st.executeQuery("select * from admin where username='"+Username+"' and password='"+Password+'"';

if(rs.next())
{
    response.sendRedirect("UserReg.jsp");
}

else
{
}
// out.print("Sorry UserName or Password Error!");

RequestDispatcher rd=request.getRequestDispatcher("Admin1.jsp");

rd.include(request, response);

out.print("<br><br><br><h1><center>Sorry UserName or Password Error!<"+"</h1>");

}

}

catch(Exception ex)
{

ex.printStackTrace();

}
Testing

The various levels of testing are

1. White Box Testing
2. Black Box Testing
3. Unit Testing
4. Functional Testing
5. Performance Testing
6. Integration Testing
7. Objective
8. Integration Testing
9. Validation Testing
10. System Testing
11. Structure Testing
12. Output Testing
13. User Acceptance Testing

White Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.
White-box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage

White-box testing is a method of testing the application at the level of the source code. The test cases are derived through the use of the design techniques mentioned above: control flow testing, data flow testing, branch testing, path testing, statement coverage and decision coverage as well as modified condition/decision coverage. White-box testing is the use of these techniques as guidelines to create an error free environment by examining any fragile code.

These White-box testing techniques are the building blocks of white-box testing, whose essence is the careful testing of the application at the source code level to prevent any hidden errors later on. These different techniques exercise every visible path of the source code to minimize errors and create an error-free environment. The whole point of white-box testing is the ability to know which line of the code is being executed and being able to identify what the correct output should be.

**Levels**

1. Unit testing. White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.

2. Integration testing. White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated
environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.

3. Regression testing. White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

White-box testing's basic procedures involve the understanding of the source code that you are testing at a deep level to be able to test them. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analysed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

1. Input, involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white-box testing to layout all of the basic information.

2. Processing Unit, involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.

3. Output; prepare final report that encompasses all of the above preparations and results.

Black Box Testing

Black-box testing is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings (see white-box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.
Test procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

Test cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

Test design techniques

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition tables
- Equivalence partitioning
- Boundary value analysis

Unit testing

In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an
individual method. Unit tests are created by programmers or occasionally by white box testers during the development process.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended. Its implementation can vary from being very manual (pencil and paper) to being formalized as part of build automation.

Testing will not catch every error in the program, since it cannot evaluate every execution path in any but the most trivial programs. The same is true for unit testing. Additionally, unit testing by definition only tests the functionality of the units themselves. Therefore, it will not catch integration errors or broader system-level errors (such as functions performed across multiple units, or non-functional test areas such as performance).

Unit testing should be done in conjunction with other software testing activities, as they can only show the presence or absence of particular errors; they cannot prove a complete absence of errors. In order to guarantee correct behaviour for every execution path and every possible input, and ensure the absence of errors, other techniques are required, namely the application of formal methods to proving that a software component has no unexpected behaviour.

Software testing is a combinatorial problem. For example, every Boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

This obviously takes time and its investment may not be worth the effort. There are also many problems that cannot easily be tested at all – for example those that are nondeterministic or involve multiple threads. In addition, code for a unit test is likely to be at least as buggy as the code it is testing. Fred Brooks in The Mythical Man-Month quotes: *never take two chronometers to sea. Always take one or*
three. Meaning, if two chronometers contradict, how do you know which one is correct?

Another challenge related to writing the unit tests is the difficulty of setting up realistic and useful tests. It is necessary to create relevant initial conditions so the part of the application being tested behaves like part of the complete system. If these initial conditions are not set correctly, the test will not be exercising the code in a realistic context, which diminishes the value and accuracy of unit test results.

To obtain the intended benefits from unit testing, rigorous discipline is needed throughout the software development process. It is essential to keep careful records not only of the tests that have been performed, but also of all changes that have been made to the source code of this or any other unit in the software. Use of a version control system is essential. If a later version of the unit fails a particular test that it had previously passed, the version-control software can provide a list of the source code changes (if any) that have been applied to the unit since that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately if such a process is not implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite, increasing false positives and reducing the effectiveness of the test suite.

Unit testing embedded system software presents a unique challenge: Since the software is being developed on a different platform than the one it will eventually run on, you cannot readily run a test program in the actual deployment environment, as is possible with desktop programs.[7]

Functional testing

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional Testing usually describes what the system does.
Functional testing differs from system testing in that functional testing "verifies a program by checking it against ... design document(s) or specification(s)", while system testing "validate a program by checking it against the published user or system requirements" (Kane, Falk, Nguyen 1999, p. 52).

Functional testing typically involves five steps. The identification of functions that the software is expected to perform

1. The creation of input data based on the function's specifications
2. The determination of output based on the function's specifications
3. The execution of the test case
4. The comparison of actual and expected outputs

Performance testing

In software engineering, performance testing is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the implementation, design and architecture of a system.

Testing types

Load testing

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration.
This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards bottlenecks in the application software.

**Stress testing**

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

**Soak testing**

Soak testing, also known as endurance testing, is usually done to determine if the system can sustain the continuous expected load. During soak tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended, significant period of time. The goal is to discover how the system behaves under sustained use.

**Spike testing**

Spike testing is done by suddenly increasing the number of or load generated by, users by a very large amount and observing the behaviour of the system. The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load.

**Configuration testing**

Rather than testing for performance from the perspective of load, tests are created to determine the effects of configuration changes to the system's components on the system's performance and behaviour. A common example would be experimenting with different methods of load-balancing.
**Isolation testing**

Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Often used to isolate and confirm the fault domain.

**Integration testing**

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

**Purpose**

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.

Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up. Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.
**Big Bang**

In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

A type of Big Bang Integration testing is called **Usage Model testing**. Usage Model Testing can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use.

Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

For integration testing, Usage Model testing can be more efficient and provides better test coverage than traditional focused functional integration testing. To be more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

**Top-down and Bottom-up**

**Bottom Up Testing** is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This
approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

**Top Down Testing** is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

**Sandwich Testing** is an approach to combine top down testing with bottom up testing.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link

**Verification and validation**

**Verification and Validation** are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fulfills its intended purpose. These are critical components of a quality management system such as ISO 9000. The words "verification" and "validation" are sometimes preceded with "Independent" (or IV&V), indicating that the verification and validation is to be performed by a disinterested third party.

It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?" In practice, the usage of these terms varies. Sometimes they are even used interchangeably.

The PMBOK guide, an IEEE standard, defines them as follows in its 4th edition

- **"Validation."** The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification.
- **"Verification."** The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation.
- Verification is intended to check that a product, service, or system (or portion thereof, or set thereof) meets a set of initial design specifications. In the development phase, verification procedures involve performing special tests to model or simulate a portion, or the entirety, of a product, service or system, then performing a review or analysis of the modelling results. In the post-development phase, verification procedures involve regularly repeating tests devised specifically to ensure that the product, service, or system continues to meet the initial design requirements, specifications, and regulations as time progresses. It is a process that is used to evaluate whether a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. Verification can be in development, scale-up, or production. This is often an internal process.

- Validation is intended to check that development and verification procedures for a product, service, or system (or portion thereof, or set thereof) result in a product, service, or system (or portion thereof, or set thereof) that meets initial requirements. For a new development flow or verification flow, validation procedures may involve modelling either flow and using simulations to predict faults or gaps that might lead to invalid or incomplete verification or development of a product, service, or system (or portion thereof, or set thereof). A set of validation requirements, specifications, and regulations may then be used as a basis for qualifying a development flow or verification flow for a product, service, or system (or portion thereof, or set thereof). Additional validation procedures also include those that are designed specifically to ensure that modifications made to an existing qualified development flow or verification flow will have the effect of producing a product, service, or system (or portion thereof, or set thereof) that meets the initial design requirements, specifications, and regulations; these validations help to keep the flow qualified. It is a process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders. This is often an external process.
• It is sometimes said that validation can be expressed by the query "Are you building the right thing?" and verification by "Are you building it right?". "Building the right thing" refers back to the user's needs, while "building it right" checks that the specifications are correctly implemented by the system. In some contexts, it is required to have written requirements for both as well as formal procedures or protocols for determining compliance.

• It is entirely possible that a product passes when verified but fails when validated. This can happen when, say, a product is built as per the specifications but the specifications themselves fail to address the user’s needs.

Activities

Verification of machinery and equipment usually consists of design qualification (DQ), installation qualification (IQ), operational qualification (OQ), and performance qualification (PQ). DQ is usually a vendor's job. However, DQ can also be performed by the user, by confirming through review and testing that the equipment meets the written acquisition specification.

If the relevant document or manuals of machinery/equipment are provided by vendors, the later 3Q needs to be thoroughly performed by the users who work in an industrial regulatory environment. Otherwise, the process of IQ, OQ and PQ is the task of validation.

The typical example of such a case could be the loss or absence of vendor's documentation for legacy equipment or do-it-yourself (DIY) assemblies (e.g., cars, computers etc.) and, therefore, users should endeavour to acquire DQ document beforehand. Each template of DQ, IQ, OQ and PQ usually can be found on the internet respectively,

whereas the DIY qualifications of machinery/equipment can be assisted either by the vendor's training course materials and tutorials, or by the published guidance books, such as step-by-step series if the acquisition of machinery/equipment is not bundled with on-site qualification services.
This kind of the DIY approach is also applicable to the qualifications of software, computer operating systems and a manufacturing process. The most important and critical task as the last step of the activity is to generating and archiving machinery/equipment qualification reports for auditing purposes, if regulatory compliances are mandatory.

Qualification of machinery/equipment is venue dependent, in particular items that are shock sensitive and require balancing or calibration, and re-qualification needs to be conducted once the objects are relocated. The full scales of some equipment qualifications are even time dependent as consumables are used up (i.e. filters) or springs stretch out, requiring recalibration, and hence re-certification is necessary when a specified due time lapse Re-qualification of machinery/equipment should also be conducted when replacement of parts, or coupling with another device, or installing a new application software and restructuring of the computer which affects especially the pre-settings, such as on BIOS, registry, disk drive partition table, dynamically-linked (shared) libraries, or an ini file etc., have been necessary. In such a situation, the specifications of the parts/devices/software and restructuring proposals should be appended to the qualification document whether the parts/devices/software are genuine or not.

Torres and Hyman have discussed the suitability of non-genuine parts for clinical use and provided guidelines for equipment users to select appropriate substitutes which are capable to avoid adverse effects. In the case when genuine parts/devices/software are demanded by some of regulatory requirements, then re-qualification does not need to be conducted on the non-genuine assemblies. Instead, the asset has to be recycled for non-regulatory purposes.

When machinery/equipment qualification is conducted by a standard endorsed third party such as by an ISO standard accredited company for a particular division, the process is called certification. Currently, the coverage of ISO/IEC 15408 certification by an ISO/IEC 27001 accredited organization is limited; the scheme requires a fair amount of efforts to get popularized.
System testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification

Types of tests to include in system testing

The following examples are different types of testing that should be considered during System testing:

- Graphical user interface testing
- Usability testing
- Software performance testing
- Compatibility testing
- Exception handling
- Load testing
- Volume testing
- Stress testing
- Security testing
● Scalability testing
● Sanity testing
● Smoke testing
● Exploratory testing
● Ad hoc testing
● Regression testing
● Installation testing
● Maintenance testing Recovery testing and failover testing.

Accessibility testing, including compliance with:
● Americans with Disabilities Act of 1990
● Section 508 Amendment to the Rehabilitation Act of 1973
● Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

**Structure Testing:**

It is concerned with exercising the internal logic of a program and traversing particular execution paths.

**Output Testing:**

● Output of test cases compared with the expected results created during design of test cases.
● Asking the user about the format required by them tests the output generated or displayed by the system under consideration.
● Here, the output format is considered into two ways, one is on screen and another one is printed format.
● The output on the screen is found to be correct as the format was designed in the system design phase according to user needs.
● The output comes out as the specified requirements as the user’s hard copy.
User acceptance Testing:

- Final Stage, before handling over to the customer which is usually carried out by the customer where the test cases are executed with actual data.
- The system under consideration is tested for user acceptance and constantly keeping touch with the prospective system user at the time of developing and making changes whenever required.
- It involves planning and execution of various types of test in order to demonstrate that the implemented software system satisfies the requirements stated in the requirement document.

Two set of acceptance test to be run:

1. Those developed by quality assurance group.
2. Those developed by customer.
**Future work:**

As a response, erasure coding as an alternative to backup has emerged as a method of protecting against drive failure. Raid just does not cut it in the age of high-capacity HDDs. The larger a disk's capacity, the greater the chance of bit error. And when a disk fails, the Raid rebuild process begins, during which there is no protection against a second (or third) mechanism failure. So not only has the risk of failure during normal operation grown with capacity, it is much higher during Raid rebuild, too. Also, rebuild times were once measured in minutes or hours, but disk transfer rates have not kept pace with the rate of disk capacity expansion, so large Raid rebuilds can now take days or even longer.
CONCLUSION:

In this paper, we present a secure and collusion-resistant proxy re-encryption protocol and an untraceable and faulttolerant OCLT-ORAM protocol for group data sharing in a cloud storage scheme. Based on key exchange, the proposed approach can efficiently generate the users conference key, which can be used to protect the security of shared data and prevent malicious user collusion with other users. In addition, security of shared group data in the cloud and access control are achieved with respect to the proxy re-encryption technique.

Moreover, according to the operation algorithms and the novel OCLT storage structure, our OCLT-ORAM protocol can support untraceability of address sequences and efficiency in data storage. Fault-tolerant and tamper protection features are accomplished with respect to pointer tuples. The sufficient security proof indicates the security of our protocol. The experimental comparison results could be considered as validation of the performance of our protocol, making it substantially more convincing.
REFERENCES:


