CLOUD DATA SECURITY USING BACKTRACKING ALGORITHM AND ASCII STEGNOGRAPHY

B.DEVADHARSHINI, M.Sc(computer science specialization with information security and cyber forensics).,

Rathinam college of arts and science , Eachanari, Coimbatore-641021

Bharathiyar University

ABSTRACT

In cloud computing, data security plays an important role where confidentiality, authentication, integrity, non repudiation are given importance. The usual technique for providing confidentiality of transmitted data is cryptography. This paper provides a technique to encrypt the data using a key involving ECC hash value as the password. Three set of security are used to provide secure data transmission with the ASCII Stenography acting as vital security element thereby providing authentication. In the present world of cloud data transmission it is difficult to transmit data from one place to another with security. This is because hackers are becoming more powerful nowadays. To ensure secured data transmission there are several techniques being followed. One among them is cryptography which is the practice and study of hiding information. The proposed method is BackTrack-ASCII algorithm, this is a new cryptographic algorithm which is used for secure the data in cloud. Encryption and decryption require the use of some secret information, usually referred to as a hash key. ECC algorithm is used to generate the key. The data to be encrypted is called as plain text. The plain text is converted into ASCII code, which is added to ASCII code of cover message which is generated by Backtrack algorithm, also the key is added to these. The encrypted data obtained as a result of encryption process is called as cipher text. Depending on the encryption mechanism used, the same key might be used for both encryption and decryption, while for other mechanisms, the keys used for encryption and decryption might be different.

1. INTRODUCTION

Cloud computing has been envisioned as the next generation information technology (IT) architecture for enterprises, due to its long list of unprecedented advantages in the IT history: on-demand self-service, ubiquitous network access, location independent resource pooling, rapid resource elasticity, usage-based pricing and transference of risk. As a disruptive technology with profound implications, cloud computing is transforming the very nature of how businesses use information technology.

One fundamental aspect of this paradigm shifting is that data are being centralized or outsourced to the cloud. From users' perspective, including both individuals and IT enterprises, storing data remotely to the cloud in a flexible on-demand manner brings appealing benefits: relief of the burden for storage management, universal data access with location independence, and avoidance of capital expenditure on hardware, software, and personnel maintenances, etc.,

Cloud Computing

Cloud computing, or something being in the cloud, is an expression used to describe a variety of different types of computing concepts that involve a large number of computers connected through a real-time communication network such as the Internet. In science, cloud computing is a synonym for distributed computing over a network and means the ability to run a program on many connected computers at the same time. The phrase is also more commonly used to refer to network-based services which appear to be provided by real server hardware, which in fact are served up by virtual hardware, simulated by software running on one or more real machines. Such virtual servers do not physically exist and can therefore be moved around and scaled up (or down) on the fly without affecting the end user—arguably, rather like a cloud. The popularity of the term can be attributed to its use in marketing to sell hosted services in the sense of application service provisioning that run client server software on a remote location.

Advantages

Cloud computing relies on sharing of resources to achieve coherence and economies of scale similar to a utility (like the electricity grid) over a network. At the foundation of cloud computing is the broader concept of converged infrastructure and shared services. The cloud also focuses on maximizing the effectiveness of the shared resources. Cloud resources are usually not only shared by multiple users but are also dynamically re-allocated per demand. This can work for allocating resources to users. For example, a cloud computer facility, which serves European users during European business hours with a specific application (e.g. email) while the same resources are getting reallocated and serve North American users during North America's business hours with another application (e.g. web server). This approach should maximize the use of computing powers thus reducing environmental damage as well since less power, air conditioning, rackspace, etc. is required for a variety of functions.

The term "moving to cloud" also refers to an organization moving away from a traditional CAPEX model (buy the dedicated hardware and depreciate it over a period of time) to the OPEX model (use a shared cloud infrastructure and pay as you use it).

Proponents claim that cloud computing allows companies to avoid upfront infrastructure costs, and focus on projects that differentiate their businesses instead of infrastructure. Proponents also claim that cloud computing allows enterprises to get their

applications up and running faster, with improved manageability and less maintenance, and enables IT to more rapidly adjust resources to meet fluctuating and unpredictable business demand.

Existing system

There are various types of Cryptographic algorithms. In general they are categorized based on the number of keys that are employed for encryption and decryption The three types of algorithms are depicted as follows:

1) *Secret Key Cryptography (SKC):* Uses a single key for both encryption and decryption. The most common algorithms in use include Data Encryption Standard (DES), Advanced Encryption Standard (AES).

2) *Public Key Cryptography (PKC):* Uses one key for encryption and another for decryption. RSA (Rivest, Shamir, Adleman) algorithm is an example.

3) *Hash Functions:* Uses a mathematical transformation to irreversibly "encrypt" information. MD Message Digest, The existing techniques involve the use of keys involving prime numbers and the like (RSA).

The user in one location embeds his/her valuable data via the proposed quantum steganography protocol and uploads the covered data to the fog cloud. The intended receiver in another location accesses the data from the fog cloud and extracts the intended content via the proposed extraction approach. This paper also presents a novel quantum steganography protocol based on hash function and quantum entangled states. To the best of our knowledge, there is no prior quantum steganography protocol that authenticates an embedded secret message.

1.2 Proposed system

The proposed method is BackTrack-ASCII algorithm, which consists of the following process,

1. Cover Message Generation - **Backtracking Algorithm**

2. Secret Key Generation (Sk) - **ECC Hashing Algorithm**

3. Mixing Operation - **ASCII Steganography**

a) ASCII code generation for Secret Message (As)

b) ASCII code generation for Cover Message (Ac)

c) Stego-message generation (Sm) = As + Ac + Sk

4. Secret Message Extraction - **ASCII Steganography**

a) Secret Key Verification

b) Separating Secret Key (Ss) = Sm - Sk

c) Original Message (m) = Ss – Ac

Literature Survey

1. IFCIoT: Integrated fog cloud IoT: A novel architectural paradigm for the future Internet of Things, A. Munir, P. Kansakar, and S. U. Khan

The fog nodes (e.g., edge servers, smart routers, base stations) receive computation offloading requests and sensed data from various IoT devices. To enhance performance, energy efficiency, and real-time responsiveness of applications, we propose a reconfigurable and layered fog node (edge server) architecture that analyzes application characteristics and reconfigures the architectural resources to better meet peak workload demands. The layers of the proposed fog node architecture include application layer, analytics layer, virtualization layer, reconfiguration layer, and hardware layer. The layered architecture facilitates abstraction and implementation for fog computing paradigm that is distributed in nature and where multiple vendors (e.g., applications, services, data and content providers) are involved. We illustrate the mapping of our proposed reconfigurable and adaptive fog architecture to the intelligent transportation systems (ITS) as a consumer applications use case. The main contributions of this article are as follows: • A novel integrated fog cloud IoT (IFCIoT) architectural paradigm that harnesses the benefits of IoT, fog, and cloud computing in a unified archetype. The IFCIoT architecture promises increased performance, energy efficiency, quicker response time, scalability, and better localized accuracy for future IoT and CPS applications. We propose an energy-efficient reconfigurable layered fog node (edge server) architecture that will adapt according to fog computing application requirements. The layers of the proposed architecture include application layer, analytics layer, virtualization layer, reconfiguration layer, and hardware layer. The layered architecture facilitates abstraction and implementation for fog computing paradigm that is distributed in nature and where different services, applications, data and content providers are involved. We discuss fog computing as a key driver for future intelligent transportation systems (ITS) and illustrate the mapping of our proposed reconfigurable and adaptive fog architecture to ITS as a consumer applications use case. We highlight other potential consumer applications of the IFCIoT architecture, such as smart cities, localized weather maps and environmental monitoring, and real-time agricultural data analytics and control.

2. Fog computing and its role in the Internet of Things, F. Bonomi, R. Milito, J. Zhu, and S. Addepalli

Fog Computing extends the Cloud Computing paradigm to the edge of the network, thus enabling a new breed of applications and services. Defining characteristics of the Fog are: a) Low latency and location awareness; b) Wide-spread geographical distribution; c) Mobility; d) Very large number of nodes, e) Predominant role of wireless access, f) Strong presence of streaming and real time applications, g) Heterogeneity. In this paper we argue that the above characteristics make the Fog the appropriate

platform for a number of critical Internet of Things (IoT) services and applications, namely, Connected Vehicle, Smart Grid , Smart Cities, and, in general, Wireless Sensors and Actuators Networks (WSANs). Fog Computing paradigm, delineate its characteristics, and those of the platform that supports Fog services. The following section takes a close look at a few key applications and services of interest that substantiate our argument in favor of the Fog as the natural component of the platform required for the support for the Internet of Things. In the fourth section we examine analytics and big data in the context of applications of interest. The recognition that some of these applications demand real-time analytics as well as long-term global data mining illustrates the interplay and complementary roles of Fog and Cloud. We have outlined the vision and defined key characteristics of Fog Computing, a platform to deliver a rich portfolio of new services and applications at the edge of the network. The motivating examples peppered throughout the discussion range from conceptual visions to existing point solution prototypes. We envision the Fog to be a unifying platform, rich enough to deliver this new breed of emerging services and enable the development of new applications. We welcome collaborations on the substantial body of work ahead: 1) Architecture of this massive infrastructure of compute, storage, and networking devices; 2) Orchestration and resource management of the Fog nodes; 3) Innovative services and applications to be supported by the Fog.

3. Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare, B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya

Internet of Things (IoT) offers a seamless platform to connect people and objects to one another for enriching and making our lives easier. This vision carries us from compute-based centralized schemes to a more distributed environment offering a vast amount of applications such as smart wearables, smart home, smart mobility, and smart cities. In this paper we discuss applicability of IoT in healthcare and medicine by presenting a holistic architecture of IoT eHealth ecosystem. Healthcare is becoming increasingly difficult to manage due to insufficient and less effective healthcare services to meet the increasing demands of rising aging population with chronic diseases. We propose that this requires a transition from the clinic-centric treatment to patient-centric healthcare where each agent such as hospital, patient, and services are seamlessly connected to each other. This patient-centric IoT eHealth ecosystem needs a multi-layer architecture: 1) device, 2) fog computing and 3) cloud to empower handling of complex data in terms of its variety, speed, and latency. This fog-driven IoT architecture is followed by various case examples of services and applications that are implemented on those layers. Those examples range from mobile health, assisted living, e-medicine, implants, early warning systems, to population monitoring in smart cities. We then finally address the challenges of IoT eHealth such as

data management, scalability, regulations, interoperability, device-network-human interfaces, security, and privacy

4. A security model for preserving the privacy of medical big data in a healthcare cloud using a fog computing facility with pairing based cryptography, H. A. A. Hamid

Nowadays, telemedicine is an emerging healthcare service where the healthcare professionals can diagnose, evaluate, and treat a patient using telecommunication technology. To diagnose and evaluate a patient, the healthcare professionals need to access the electronic medical record (EMR) of the patient, which might contain huge multimedia big data including x-rays, ultrasounds, CT scans, MRI reports, etc. For efficient access and supporting mobility for both the healthcare professionals as well as the patients, the EMR needs to be kept in big data storage in the healthcare cloud. In spite of the popularity of the healthcare cloud, it faces different security issues; for instance, data theft attacks are considered to be one of the most serious security breaches of healthcare data in the cloud. In this paper, the main focus has been given to secure healthcare private data in the cloud using a fog computing facility. To this end, a tri-party oneround authenticated key agreement protocol has been proposed based on the bilinear pairing cryptography that can generate a session key among the participants and communicate among them securely. Finally, the private healthcare data are accessed and stored securely by implementing a decoy technique. In this paper, a methodology is presented to secure patients' MBD in the healthcare cloud using the decoy technique with a fog computing facility. It serves as a second gallery to contain decoy MBD (DMBD) that appear to the attacker as if it is the original MBD (OMBD). Unlike other methods, where the decoy files are called when an attacker is detected as accessing the system, in our proposed methodology the decoy files are retrieved from the beginning to ensure better security. Additionally, it uses a double security technique by encrypting the original file when an attacker recognizes that he/she is dealing with a decoy gallery; he/she would need to figure out how to decode the original gallery. As a result, our methodology ensures that the users' MBD are 100% secure and shortens the process. There is no need to worry if the user is an attacker, since by default it offers the decoy big data gallery directly to any user and keeps the original one hidden, which is only made available to a legitimate user after successful verification.

2.MODULES DESCRIPTION:

1. HIDING SENDER
2. HIDING MESSAGE
3. MESSAGES TRANSMISSION
4. DECRYPT THE MESSAGE
5. DECRYPT THE RECEIVER

The detailed description of the modules are ,

Hiding Sender

In the present world scenario it is difficult to transmit data from one place to another with security. This is because hackers

are becoming more powerful nowadays. To ensure secured data transmission there are several techniques being followed. One among them is cryptography which is the practice and study of hiding information.

When an sender's message send to the receiver this application will start to

work and receiver send the sender's id(user id).Then user login their id and verifying id acknowledgements .This process for comparing each other that contacting user was correct ,If not the sender rejects the user requests.

Hiding Message

In this technique the first step is to assign a unique color for each receiver. Each color is represented with a set of three values. For example violet red color is represented in RGB format as (238, 58,140). The next step is to assign a set of three key values to each receiver.

At the receiver's side, thereceiver is aware of his own color and other key values.

The encrypted color from the sender is decrypted by subtracting the key values from the received set of color values. It is then tested for a match with the color stored at the sender's database. Only when the colors are matched the actual data can be decrypted using Armstrong numbers. Usage of colors as a password in this way ensures more security to the data providing authentication. This is because only when the colors at the sender and receiver's side match with each other the actual data could be accessed.

Message Transmision

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

Today's technology-led business environment is influenced by multiple factors placing significant importance on software security. Software security implies identifying and understanding common threats, designing for security, and subjecting all software artifacts to thorough risk analysis assessment.

It can set a messaging configuration parameter to determine if plain passwords are allowed or if passwords must be encrypted.

Decrypt The Message

Encrypt provides symmetric encryption functionality via the CryptoKey object. CryptoKey's core methods, EncryptFile, DecryptFile, EncryptText and DecryptText, allow you to implement file and text encryption in your application in just a few lines of code.

An instance of the CryptoKey object is created using CryptoContext's methods GenerateKey and GenerateKeyFromPassword. The former generates a random key, while the latter generates a key based on a text password. A key

generated by the GenerateKey method must be serialized to a file or other permanent storage in order to be used for decryption later. A password-derived key does not have to be serialized as it can always be re-created using the same password .

DECRYPT THE RECEIVER

Decryption involves the process of getting back the original data using decryption key. The data given by the receiver (the color) is matched with the data stored at the sender's end. For this process the receiver must be aware of his own color being assigned and the key values.

The received data now with substitute with the key value in the receiver side,then we get back a set of values.And then it compares to the data which is stored in the sender's side.if both get matched together following decr yption

In an enterprise environment, using password-derived keys may not be feasible as everybody would have to share a password to encrypt and decrypt files. It makes more sense to encrypt files using randomly generated keys that are stored in a key repository and dispensed to eligible authenticated users upon request. Needless to say, these keys would have to be encrypted.

3. System Features:

3.1 System Feature

SOFTWARE USED:

- Language: JAVA
- Front End: J2EE
- Back End: MySQL

HARDWARE USED:

- 3.5GHZ Processer
- 40GB hard disk
- 1GB RAM

There are three typical solutions to this problem.

1. **Cookies.** You can use HTTP cookies to store information about a shopping session, and each subsequent connection can look up the current session and then extract information about that session from some location on the server machine. This is an excellent alternative, and is the most widely used approach. However, even though servlets have a high-level and easy-to-use interface to cookies, there are still a number of relatively tedious details that need to be handled:
    - Extracting the cookie that stores the session identifier from the other cookies (there may be many, after all),
    - Setting an appropriate expiration time for the cookie (sessions interrupted by 24 hours probably should be reset), and

- Associating information on the server with the session identifier (there may be far too much information to actually store it in the cookie, plus sensitive data like credit card numbers should never go in cookies).

2. **URL Rewriting**. You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. This is also an excellent solution, and even has the advantage that it works with browsers that don't support cookies or where the user has disabled cookies. However, it has most of the same problems as cookies, namely that the server-side program has a lot of straightforward but tedious processing to do. In addition, you have to be very careful that every URL returned to the user (even via indirect means like Location fields in server redirects) has the extra information appended. And, if the user leaves the session and comes back via a bookmark or link, the session information can be lost.
3. **Hidden form fields**. HTML forms have an entry that looks like the following: <INPUT TYPE="HIDDEN" NAME="session" VALUE="..."> . This means that, when the form is submitted, the specified name and value are included in the GET or POST data. This can be used to store information about the session. However, it has the major disadvantage that it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.

Servlets provide an outstanding technical solution: the HttpSession API. This is a high-level interface built on top of cookies or URL-rewriting. In fact, on many servers, they use cookies if the browser supports them, but automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled. But the servlet author doesn't need to bother with many of the details, doesn't have to explicitly manipulate cookies or information appended to the URL, and is automatically given a convenient place to store data that is associated with each session.

The Session Tracking API

Using sessions in servlets is quite straightforward, and involves looking up the session object associated with the current request, creating a new session object when necessary, looking up information associated with a session, storing information in a session, and discarding completed or abandoned sessions.

Looking up the HttpSession object associated with the current request.

This is done by calling the getSession method of HttpServletRequest. If this returns null, you can create a new session, but this is so

commonly done that there is an option to automatically create a new session if there isn't one already. Just pass true to getSession. Thus, your first step usually looks like this:

```
HttpSession session = request.getSession(true);
```

Looking up Information Associated with a Session.

HttpSession objects live on the server; they're just automatically associated with the requester by a behind-the-scenes mechanism like cookies or URL-rewriting. These session objects have a builtin data structure that let you store any number of keys and associated values. In version 2.1 and earlier of the servlet API, you use getValue("key") to look up a previously stored value. The return type is Object, so you have to do a typecast to whatever more specific type of data was associated with that key in the session. The return value is null if there is no such attribute. In version 2.2, getValue is deprecated in favor of getAttribute, both because of the better naming match with setAttribute (the match for getValue is putValue, not setValue), and because setAttribute lets you use an attached HttpSessionBindingListener to monitor values, while putValue doesn't. Nevertheless, since few commercial servlet engines yet support version 2.2, I'll use getValue in my examples. Here's one representative example, assuming ShoppingCart is some class you've defined yourself that stores information on items being purchased.

```
HttpSession session = request.getSession(true);
ShoppingCart previousItems =
(ShoppingCart)session.getValue("previousItems");
if (previousItems != null) {
  doSomethingWith(previousItems);
} else {
  previousItems = new ShoppingCart(...);
  doSomethingElseWith(previousItems);
}
```

In most cases, you have a specific attribute name in mind, and want to find the value (if any) already associated with it. However, you can also discover all the attribute names in a given session by calling getValueNames, which returns a String array. In version 2.2, use getAttributeNames, which has a better name and which is more consistent in that it returns an Enumeration, just like the getHeaders and getParameterNames methods of HttpServletRequest.

Although the data that was explicitly associated with a session is the part you care most about, there are some other pieces of information that are sometimes useful as well.

- getId. This method returns the unique identifier generated for each session. It is sometimes used as the key name when there is only a single value associated with a session, or when logging information about previous sessions.

- isNew. This returns true if the client (browser) has never seen the session, usually because it was just created rather than being referenced by an incoming client request. It returns false for preexisting sessions.
- getCreationTime. This returns the time, in milliseconds since the epoch, at which the session was made. To get a value useful for printing out, pass the value to the Date constructor or the setTimeInMillis method of GregorianCalendar.
- getLastAccessedTime. This returns the time, in milliseconds since the epoch, at which the session was last sent from the client.
- getMaxInactiveInterval. This returns the amount of time, in seconds, that a session should go without access before being automatically invalidated. A negative value indicates that the session should never timeout.

Associating Information with a Session

As discussed in the previous section, you read information associated with a session by using getValue (or getAttribute in version 2.2 of the servlet spec). To specify information, you use putValue (or setAttribute in version 2.2), supplying a key and a value. Note that putValue replaces any previous values. Sometimes that's what you want (as with the referringPage entry in the example below), but other times you want to retrieve a previous value and augment it (as with the previousItems entry below).

Here's an example:

```
HttpSession session = request.getSession(true);

session.putValue("referringPage", request.getHeader("Referer"));
ShoppingCart previousItems =

(ShoppingCart)session.getValue("previousItems");
if (previousItems == null) {
  previousItems = new ShoppingCart(...);
}
String itemID = request.getParameter("itemID");

previousItems.addEntry(Catalog.getEntry(itemID));
// You still have to do putValue, not just modify the cart, since
// the cart may be new and thus not already stored in the session.
session.putValue("previousItems", previousItems);
```

`statement`

The objective of the Statement interface is to pass to the database the SQL string for execution and to retrieve any results from the database in the form of a ResultSet. Only one ResultSet can be open per statement at any one time. For example, two ResultSets cannot be compared to each other if both ResultSets stemmed from the same SQL statement. If an SQL statement is re-issued for any reason, the old Resultset is automatically closed.

ResultSet

A ResultSet is the retrieved data from a currently executed SQL statement. The data from the query is delivered in the form of a table. The rows of the table are returned to the program in sequence. Within any one row, the multiple columns may be accessed in any order

A pointer known as a cursor holds the current retrieved record. When a ResUltSet is returned, the cursor is positioned before the first record and the next command (equivalent to the embedded SQL FETCH command) pulls back the first row. A ResultSet cannot go backwards. In order to re-read a previously retrieved row, the program must close the ResultSet and re-issue the SQL statement. Once the last row has been retrieved the statement is considered closed, and this causes the Resu1~Set to be automatically closed.

CallableStatement

This interface is used to execute previously stored SQL procedures in a way which allows standard statement issues over many relational DBMSs. Consider the SQL example:

SELECT cname FROM tnaine WHERE cname = var;

If this statement were to be stored, the program would need a way to pass the parameter var into the callable procedure. Parameters passed into the call are referred to sequentially, by number. When defining a variable type in JDBC the program must ensure that the type corresponds with the database field type for IN and OUT parameters.

DatabaseMetaData

This interface supplies information about the database as a whole. MetaData refers to information held about data. The information returned is in the form of ResultSet5. Normal ResultSet methods, as explained previously, may be used in this instance. If metadata is not available for the particular request then an SQLException will occur

Driver

For each database driver a class that implements the Driver interface must be provided. When such a class is loaded it should register itself with the DriverManager, which will then allow it to be accessed by a program.

PreparedStatement

A PreparedStatement object is an SQL statement which is pre-compiled and stored. This object can then be executed multiple times much more efficiently than preparing and issuing the same statement each time it is needed. When defining a variable type in JDBC, the program must ensure that the type corresponds with the database field type for IN and OUT parameters.

ResultSetMetaData

This interface allows a program to determine types and properties in any columns in a ResultSet. It may be used to find out a data type

for a particular field before assigning its variable type.

DriverPropertyinfo

This class is only of interest to advanced programmers. Its purpose is to interact with a particular driver to determine any properties needed for connections.

Date

The purpose of the Date class is to supply a wrapper to the standard Java Date class which extends to allow JDBC to recognise an SQL DATE.

Time

The purpose of the Time class is to supply a wrapper to the standard Java Time class which extends to allow JDBC to recognise an SQL TIME.

Timestamp

The purpose of the Times tamp class is to supply a wrapper to the standard Java Date class which extends to allow JDBC to recognise an SQL TIMESTAMP

Types

The Types class determines any constants that are used to identify SQL types.

Numeric

The object of the Numeric class is to provide high precision in numeric computations that require fixed point resolution. Examples include monetary or encryption key applications. These equate to database SQL NUMERIC or DECIMAL types.

Driver Interface

The driver side of the JDBC layer is the part that interfaces to the actual database, and therefore is generally written by database vendors. Most developers only need to know how to install and use drivers. The JDBC Driver API defines a set of interfaces which have to be implemented by a vendor

JDBC is based on Microsoft's Open Database Connectivity (ODBC) interface which many of the mainstream databases have adopted. Therefore, a JDBCODBC bridge is supplied as part of JDBC, which allows most databases to be accessed before the Java driver is released. Although efficient and fast, it is recommended that the actual database JDBC driver is used rather than going through another level of abstraction with ODBC.

Developers have the power to develop and test applications that use the JDBC-ODBC bridge. If and when a proper driver becomes available they will be able to slot in the new driver and have the applications utilise it instantly, without the need for rewriting. However, do not assume the JDBC-ODBC bridge is a bad alternative. It is a small and very efficient way of accessing databases.

Application Areas

JDBC has been designed and implemented for use in connecting to databases. Fortunately, JDBC has made no restrictions, over and above the standard Java security mechanisms, for complete systems. To this end, a number of overall system configurations are feasible for accessing databases.

1. Java application which accesses local database
2. Java applet accesses server-based database
3. Database access from an applet via a stepping stone

Using a JDBC driver

JavaSoft has defined the following driver categorization system:

type 1

These drivers use a bridging technology to access a database. The JDBC-ODBC bridge that comes with the JDK 1.1 is a good example of this kind of driver. It provides a gateway to the ODBC API. Implementations of that API in turn do the actual database access. Bridge solutions generally require software to be installed on client systems, meaning that they are not good solutions for applications that do not allow you to install software on the client.

type 2

The type 2 drivers are native API drivers. This means that the driver contains Java code that calls native C or C++ methods provided by the individual database vendors that perform the database access. Again, this solution requires software on the client system.

type 3

Type 3 drivers provide a client with a generic network API that is then translated into database specific access at the server level. In other words, the JDBC driver on the client uses sockets to call a middleware application on the server that translates the client requests into an API specific to the desired driver. As it turns out, this kind of driver is extremely flexible since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

type 4

Using network protocols built into the database engine, type 4 drivers talk directly to the database using Java sockets. This is the most direct pure Java solution. In nearly every case, this type of driver will come only from the database vendor.

Regardless of data source location, platform, or driver (Oracle, Microsoft, etc.), JDBC makes connecting to a data source less difficult by providing a collection of classes that abstract details of the database interaction. Software engineering with JDBC is also conducive to module reuse. Programs can easily be ported to a different infrastructure for which you have data stored (whatever platform you choose to use in the future) with only a driver substitution.

As long as you stick with the more popular database platforms (Oracle, Informix, Microsoft, MySQL, etc.), there is almost certainly a JDBC driver written to let your programs connect and manipulate data. You can download a specific JDBC driver from the manufacturer of your database management system (DBMS) or from a third party (in the case of less popular open source products) [5]. The JDBC driver for your database will come with specific instructions to make the class files of the driver available to the Java Virtual Machine, which your program is going to run. JDBC drivers use Java's built-in `DriverManager` to open and access a database from within your Java program.

To begin connecting to a data source, you first need to instantiate an object of your JDBC driver. This essentially requires only one line of code, a command to the `DriverManager`, telling the Java Virtual Machine to load the bytecode of your driver into memory, where its methods will be available to your program. The `String` parameter below is the fully qualified class name of the driver you are using for your platform combination:

```
Class.forName("org.gjt.mm.mysql.Driver").ne
wInstance();
```

Connecting to a database

In order to connect to a database, you need to perform some initialization first. Your JDBC driver has to be loaded by the Java Virtual Machine classloader, and your application needs to check to see that the driver was successfully loaded. We'll be using the ODBC bridge driver, but if your database vendor supplies a JDBC driver, feel free to use it instead.

```
// Attempt to load database driver
try
{
        // Load Sun's jdbc-odbc driver
        Class.forName("sun.jdbc.odbc.Jdbc
OdbcDriver").newInstance();
}
catch (ClassNotFoundException cnfe) // driver
not found
{
        System.err.println ("Unable to load
database driver");
        System.err.println ("Details : " +
cnfe);
        System.exit(0);
}
```

We try to load the JdbcOdbcDriver class, and then catch the ClassNotFoundException if it is thrown. This is important, because the application might be run on a non-Sun virtual machine that doesn't include the ODBC bridge, such as Microsoft's JVM. If this occurs, the driver won't be installed, and our application should exit gracefully.

Once our driver is loaded, we can connect to the database. We'll connect via the driver manager class, which selects the appropriate driver for the database we specify. In this case, we'll only be using an ODBC database, but in more complex applications, we might wish to use

different drivers to connect to multiple databases. We identify our database through a URL. No, we're not doing anything on the web in this example - a URL just helps to identify our database.

A JDBC URL starts with "jdbc:" This indicates the protocol (JDBC). We also specify our database in the URL. As an example, here's the URL for an ODBC datasource called 'demo'. Our final URL looks like this :

jdbc:odbc:demo

To connect to the database, we create a string representation of the database. We take the name of the datasource from the command line, and attempt to connect as user "dba", whose password is "sql".

```
// Create a URL that identifies database
String url = "jdbc:odbc:" + args[0];

// Now attempt to create a database connection
Connection db_connection =
        DriverManager.getConnection (url, "dba", "sql");
```

As you can see, connecting to a database doesn't take much code.

Executing database queries

In JDBC, we use a statement object to execute queries. A statement object is responsible for sending the SQL statement, and returning a set of results, if needed, from the query. Statement objects support two main types of statements - an update statement that is normally used for operations which don't generate a response, and a query statement that returns data.

```
// Create a statement to send SQL
Statement db_statement = db_connection.createStatement();
```

Once you have an instance of a statement object, you can call its executeUpdate and executeQuery methods. To illustrate the executeUpdate command, we'll create a table that stores information about employees. We'll keep things simple and limit it to name and employee ID.

```
// Create a simple table, which stores an employee ID and name
db_statement.executeUpdate
  ("create table employee { int id, char(50) name };");
// Insert an employee, so the table contains data
db_statement.executeUpdate
  ("insert into employee values (1, 'John Doe');");
// Commit changes
db_connection.commit();
```

Now that there's data in the table, we can execute queries. The response to a query will be returned by the executeQuery method as a ResultSet object. ResultSet objects store the last response to a query for a given statement object. Instances of ResultSet have methods following the pattern of getXX where XX is the name of a data type. Such data types include numbers (bytes, ints, shorts, longs, doubles, big-

decimals), as well as strings, booleans, timestamps and binary data.

```
// Execute query
ResultSet result = db_statement.executeQuery
        ("select * from employee");

// While more rows exist, print them
while (result.next() )
{
        // Use the getInt method to obtain emp. id
        System.out.println ("ID : " + result.getInt("ID"));

        // Use the getString method to obtain emp. name
        System.out.println ("Name : " + result.getString("Name"));
        System.out.println ();
}
```

5. Other Nonfunctional Requirements

System Testing and Executing

What is testing?

Testing is a set of activities that can be planned in advance and conducted systematically.

5.1 Software Testing:

Software testing is a critical element of software quality assurances and represents the ultimate review of specifications, design and coding.

Software testing process is the means by which people, methods, measurements, tools and equipments are integrated to test a software product.

Software testing ensures that the system works accurately and efficiently before the live action commences.

The quality and effectiveness of software testing and primarily determined by the quality of the test processed used.

Testing has its own cycle and the candidate system is subject to a variety of tests.

The testing process begins with the product requirements phase and from there parallels the entire development process.

In other words, for each phase of the development process there is an important testing activity.

Black Box	White Box
Designing test cases with the knowledge of inputs, outputs and functional requirements of programs	Design test cases by understanding the actual code.

Stages of testing:

1. Plan test

a. Collect and organize test planning information

b. Create the test plan.

c. Key resulting artifacts

i. Test plan

2. Design Test

a. Identify a set of verifiable test cases for each build.

b. Design each test case.

c. Key resulting artifacts

3. Implement test:

a. Design test packages and classes

b. Implement test components and sub systems

c. Create reusable test scripts

d. Maintain trace ability to test

e. Key resulting artifacts

- Test class
- Test components
- Test packages
- Test subsystems
- Test

4. Execute test- Integration

a. Execute tests and capture test results

b. Key resulting artifacts

Test Results

5. Execute test – System test

a. Execute tests and capture test results

b. Key resulting artifacts

Test Results

6. Evaluate Test

User

a) Evaluate test results and log chance requests.
b) Calculate and deliver key measures of test.
c) Generate the test evaluation summary.
d) Execute tests and capture test results.
e) Key resulting artifacts

- Test evaluation summary
- Change requests

Levels of testing:

The various levels of testing are

- White Box Testing
- Black Box Testing
- Unit Testing
- Functional Testing
- Performance Testing
- Integration Testing
- Objective
- Integration Testing
- Validation Testing
- System Testing
- Structure Testing
- Output Testing
- Acceptance Testing

White Box Testing

Execution of every path in the program.

Black Box Testing

Exhaustive input testing is required to find all errors.

Unit Testing

Unit testing, also known as Module Testing, focuses verification efforts on the module. The

module is tested separately and this is carried out at the programming stage itself.

Unit Test comprises of the set of tests performed by an individual programmer before integration of the unit into the system.

Unit test focuses on the smallest unit of software design- the software component or module.

Using component level design, important control paths are tested to uncover errors within the boundary of the module.

Unit test is white box oriented and the step can be conducted in parallel for multiple components.

Functional Testing:

Functional test cases involve exercising the code with normal input values for which the expected results are known, as well as the boundary values

Objective:

The objective is to take unit-tested modules and build a program structure that has been dictated by design.

Performance Testing:

Performance testing determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization of the program unit. It occurs throughout all steps in the testing process.

Integration Testing:

It is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with in the interface.

It takes the unit tested modules and builds a program structure.

All the modules are combined and tested as a whole.

Integration of all the components to form the entire system and a overall testing is executed.

Validation Testing:

Validation test succeeds when the software functions in a manner that can be reasonably expected by the client.

Software validation is achieved through a series of black box testing which confirms to the requirements.

Black box testing is conducted at the software interface.

The test is designed to uncover interface errors, is also used to demonstrate that software functions are operational, input is properly accepted, output are produced and that the integrity of external information is maintained.

System Testing:

Tests to find the discrepancies between the system and its original objective, current specifications and system documentation.

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 was, 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.

5.2 Feasibility Study

Feasibility study is the test of a system proposal according to its workability, impact on the organization, ability to meet user needs, and effective use of recourses. It focuses on the evaluation of existing system and procedures analysis of alternative candidate system cost estimates. Feasibility analysis was done to determine whether the system would be feasible.

The development of a computer based system or a product is more likely plagued by resources and delivery dates. Feasibility study helps the analyst to decide whether or not to proceed, amend, postpone or cancel the project, particularly important when the project is large, complex and costly. Once the analysis of the user requirement is complement, the system has to check for the compatibility and feasibility of the software package that is aimed at. An important outcome of the preliminary investigation is the determination that the system requested is feasible.

Technical Feasibility:

The technology used can be developed with the current equipments and has the technical capacity to hold the data required by the new system.

- This technology supports the modern trends of technology.
- Easily accessible, more secure technologies.

Technical feasibility on the existing system and to what extend it can support the proposed addition.We can add new modules easily without affecting the Core Program. Most of parts are running in the server using the concept of stored procedures.

Operational Feasibility:

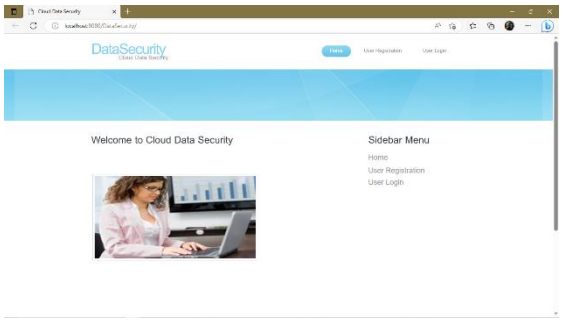
This proposed system can easily implemented, as this is based on JSP coding (JAVA) & HTML .The database created is with

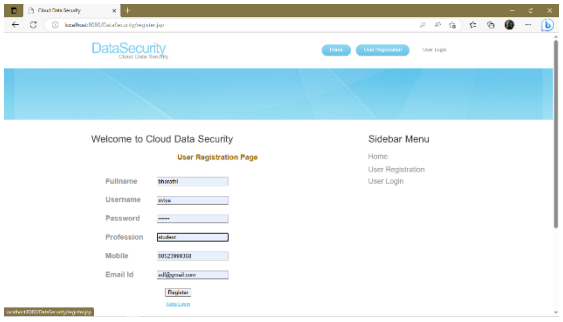
MySql server which is more secure and easy to handle. The resources that are required to implement/install these are available. The personal of the organization already has enough exposure to computers. So the project is operationally feasible.

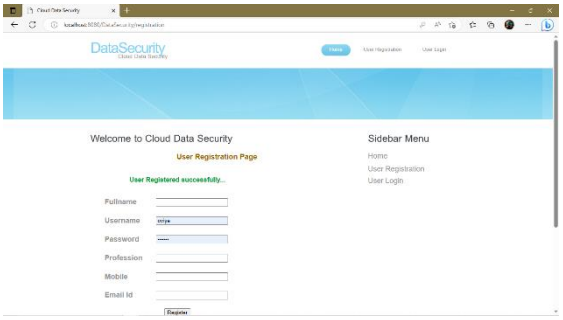
Economical Feasibility:

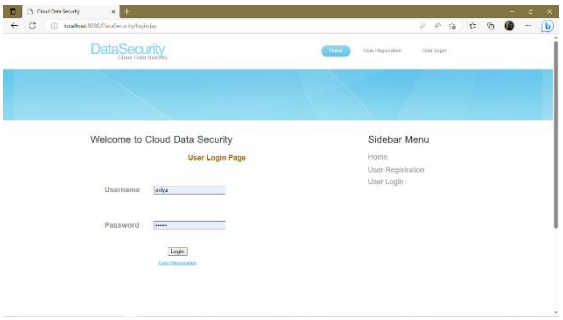
Economic analysis is the most frequently used method for evaluating the effectiveness of a new system. More commonly known cost/benefit analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the decision is made to design and implement the system. An entrepreneur must accurately weigh the cost versus benefits before taking an action. This system is more economically feasible which assess the brain capacity with quick & online test. So it is economically a good project.

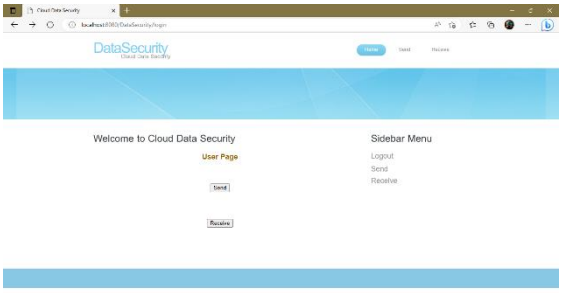
EXPERIMENTAL RESULTS:

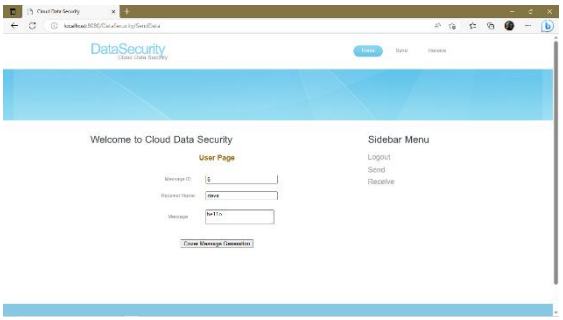


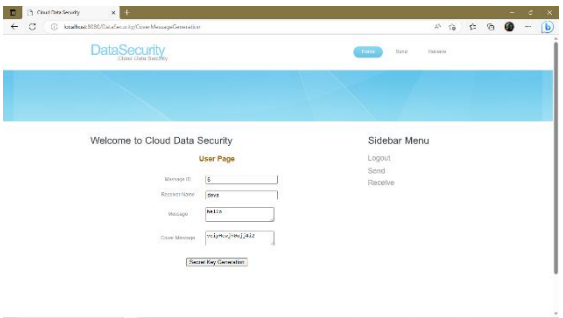


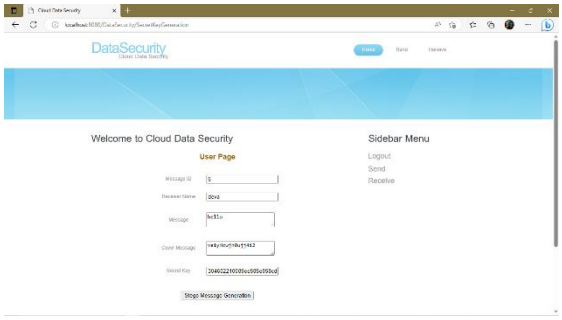


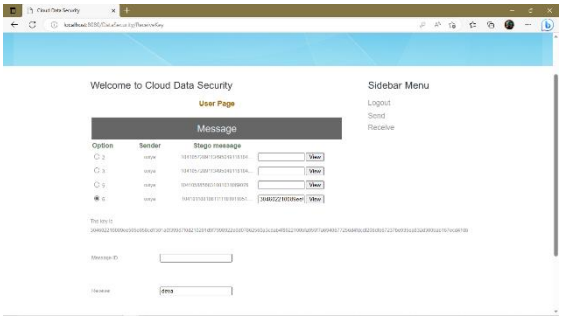


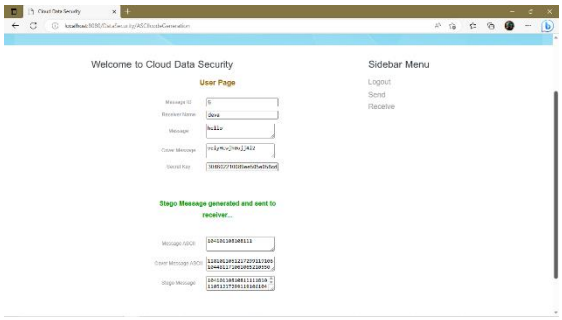


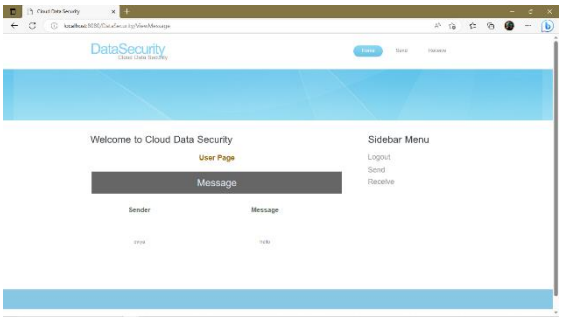


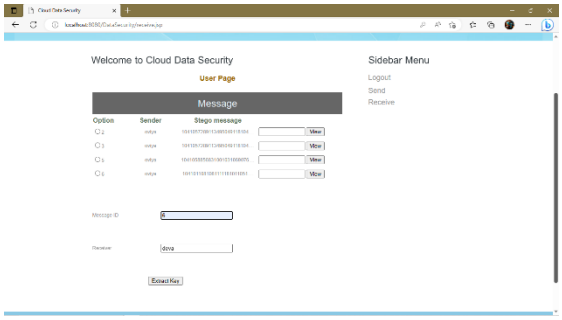


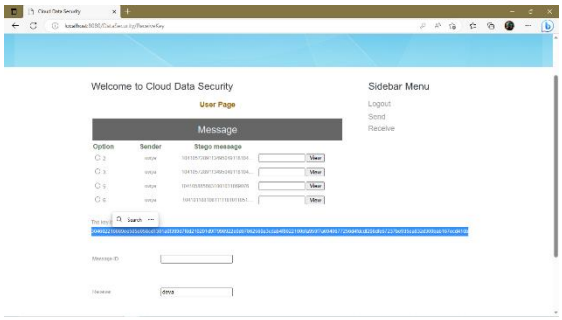












6.1 CONCLUSION

The above combination of secret key and public key cryptography can be applied mainly in military where data security is given more importance. This technique provides more security with increase in key length of the Armstrong numbers. Thus usage of three set of keys namely colors, additional set of key values and Armstrong numbers in this technique ensures that the data is transmitted securely and accessed only by authorized people.

REFERENCES

[1] Atul Kahate, "Cryptography and Network Security ", Tata McGraw Hill Publications

[2] http://aix1.uottawa.ca/~jkhoury/cryptography.htm

[3] http://www.scribd.com/doc/29422982/Data-Compression-and- Encoding-Using-Col

[4] Singh, S., *The Code Book*, New York: Doubleday, 1999

[5] Miller, C., Lial, M. and Schneider, D., *Fundamentals of College Algebra*, 3rd Edition, Scott, Foresman and Company, 1990

[6] Diffie, W. and Hellman, M., "New Directions in Cryptography", *IEEE Transactions on Information Theory*, November, 1976

[7] Rivest, R., Shamir, A. and Adelman, L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, February, 1978.

[8] Denning, D. and Denning, P., *Internet Besieged, Countering Cyberspace Scoflaws*, ACM Press (Addison-Wesley), New York, NY, 1998

[9] Denning, D., *Information Warfare and Security*, New York: Addison- Wesley/ACM Press, 1999

[10]Kahn, D., *The Codebreakers: The Story of Secret Writing*, New York: Macmillan, 1967; Scribner, 1996