International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Attribute Based Storage System with Secure Deduplication in a Hybrid Cloud Setting

Sai Ramya Akula

UG student, BTech, Computer Science, Andhra University

ABSTRACT:

Attribute-based encryption (ABE) has been widely employed in cloud computing, where a data provider outsources his or her encrypted data to a cloud service provider, who can then share the data with users who have certain credentials (or attributes). The normal ABE system, on the other hand, does not allow safe deduplication, which is critical for reducing storage space and network traffic by eliminating redundant copies of identical data. We offer an attribute-based storage system with safe deduplication in a hybrid cloud context, where a private cloud is in charge of duplicate detection and a public cloud is in charge of storage. Our technology offers two benefits over previous data deduplication solutions. For starters, rather of exchanging decryption keys, it may be used to securely share data with users by establishing access controls. Second, it achieves the standard concept of semantic security for data confidentiality, whereas existing systems only do so by specifying a weaker security concept. In addition, we present a method for converting ciphertexts with the same plaintext but different access rules into ciphertexts with the same plaintext but different access policies without revealing the underlying plaintext.

1.INTRODUCTION

1.1 Background:- Cloud computing greatly facilitates data providers who want to outsource their data to the cloud without disclosing sensitive data to third parties and who only want users with specific credentials to be able to access the data. This necessitates data being stored in encrypted forms with access control policies that ensure that only users with specific attributes (or credentials) can decrypt the encrypted data. Attribute-based encryption (ABE) is an encryption technique that meets this requirement, in which a user's private key is associated with an attribute set, a message is encrypted under an access policy(or access structure) over a set of attributes, and a user can decrypt a ciphertext with his or her private key if his or her set of attributes satisfies the access policy associated with this ciphertext. The typical ABE system, on the other hand, fails to perform secure deduplication, which is a method of reducing storage space and network traffic by removing redundant copies of encrypted data stored in the cloud.

1.2 Problem Description: In the design of an attribute-based storage system that supports secure deduplication of encrypted data in the cloud, we address the following scenario: the cloud will not store a file more than once, despite receiving numerous copies of the same file encrypted under different access permissions.

2.PROJECT ANALYSIS

2.1 EXISTING SYSTEM:

When a user uploads data that already exists in cloud storage, the user should be prevented from accessing data that existed before he took possession of it by uploading it (backward secrecy) .These dynamic ownership changes may occur often in a practical cloud system, and as a result, they must be carefully managed to prevent the cloud service's security from deteriorating. Because the hash of the file, which is used as a "proof" for the complete file, is vulnerable to being leaked to outside adversaries due to its relatively tiny size, most existing schemes have been suggested in order to perform a PoW process in an efficient and robust manner in the former approach. When a data owner uploads data that does not already exist in cloud storage, he is referred to as an initial uploader; if the data already exists, he is referred to as a subsequent uploader because this suggests that other owners may have uploaded the same material previously.

2.2 PROPOSED SYSTEM:

The purpose of this project is to save storage space for cloud storage services while simultaneously providing safe deduplication.

However, numerous deduplication processes have used the similar approach. The Attribute Authority assigns each user a decryption key based on their set of attributes, which is regarded as the most significant obstacle for efficient and secure cloud storage services in a continuously changing ownership context. Every time a data provider uploads a file to the cloud, it is checked for storage purposes. The majority of the techniques have been presented to enable data encryption while also taking advantage of deduplication. For security reasons, each user receives a secure key from the administrator. The

user is not allowed to take any keys or download hypertext files; instead, they may only download encrypted data. Attribute authority organizes and maintains every detail.

ADVANTAGES OF PROPOSED SYSTEM:

- For starters, rather than sharing decryption keys, it may be used to share data with users in a secure manner by establishing access policies.
- • Second, it achieves the standard concept of semantic security for data secrecy, whereas existing systems only do so by specifying a weaker security concept. In addition, we present a method for converting ciphertexts with the same plaintext but different access rules into ciphertexts with the same plaintext but different access policies without revealing the underlying plaintext.

3.1 FUNCTIONAL REQUIREMENTS:

3.1 Process Flow:

- When a data provider sends a file storage request, it first constructs a tag T and a label L for the data, and then encrypts it using an access policy based on a set of attributes.
- Each data source creates a proof document demonstrating the relationship between the tag T, the label L, and the encrypted message. During the checking process of any freshly produced storage request, proof is employed.
- The private cloud examines the authenticity of the proof pf after receiving a storage request, and then compares the new tag T to existing tags in the system.
- If no match exists for this new tag T, the private cloud adds the tag T and the label L to a tag-label list and sends the label and the encrypted data (L, ct) to the public cloud for storage.
- If there is a match, the incoming ciphertext's underlying plaintext has already been stored, and the new ciphertext is deleted.
- With the attribute-based private key generated by the Attribute Authority, the user can access data from the cloud.
- Each user verifies the decrypted message's validity by comparing it to the label, and accepts the message if it matches the label.

3.2 NON-FUNCTIONAL REQUIREMENTS:

Performance:

ABE's performance is measured in terms of execution time, data and network overhead, energy consumption, and CPU and memory utilization.

Availability:

The data supplier should have access to all of the user's records. Private keys should be kept in close proximity to the user

Usability:

The process flow is simple to follow, and the flow navigation is straightforward with the required menu.

4. SOFTWARE ENVIRONMENT

4.1 Java Technology:

Java is a programming language as well as a platform.

4.2 ODBC 4.2

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for database systems providers and application developers. Programmers had to utilize proprietary languages for each database they wished to connect to before ODBC became the de facto standard for Windows programs to interface with database systems.

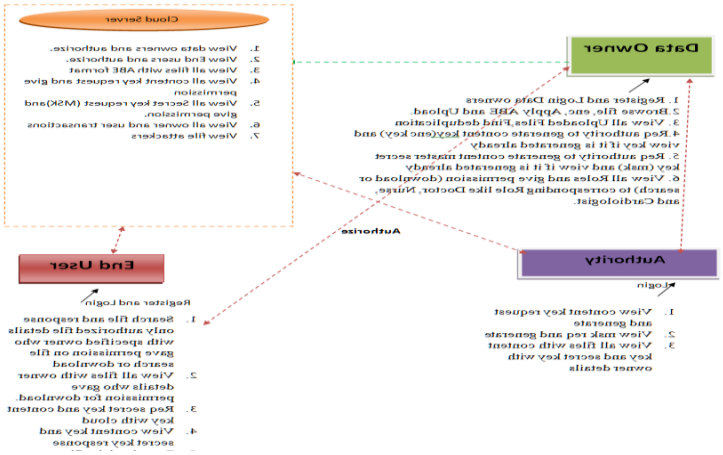
4.3 JDBC 4.3

Sun Microsystems created Java Database Connectivity, or JDBC, in order to create an independent database standard API for Java. JDBC is a general SQL database access technology that provides a uniform interface to a wide range of relational database management systems (RDBMSs).

5. SYSTEM DESIGN

DESIGN is a multi-step process that concentrates on data structure, software architecture, procedural details, and module interfaces. Before coding begins, the design phase converts the requirements into a presentation of software that can be tested for quality..

5.1 SYSTEM ARCHITECTURE



MODULES:

1. DATA OWNER
2. CLOUD SERVER
3. AUTHORITY
4. END USER

5.2.1 MODULES DESCRIPTION:

1.Data Owner:

In this module, the data owner must first register with the cloud server and obtain authorization. After receiving authorization from the cloud, the data owner will encrypt and upload the file to the cloud server, after which the data owner will request the content key and master secret key from the authority for the file he uploaded and finds deduplication, and only after the keys are generated will the file be uploaded to the cloud server. After the file has been uploaded, the data owner must grant download and search permission for each file so that people can search and download it.

2. Cloud Server:

A cloud server is a computer that maintains a cloud and provides data storage. Data owners encrypt their files and store them in the cloud for cloud End users to access. Users will need the permission of the content key and the MSK master secret key to access the shared data files. And the authorization will be granted by the cloud. In addition, all transactions and attacks associated with the files are displayed.

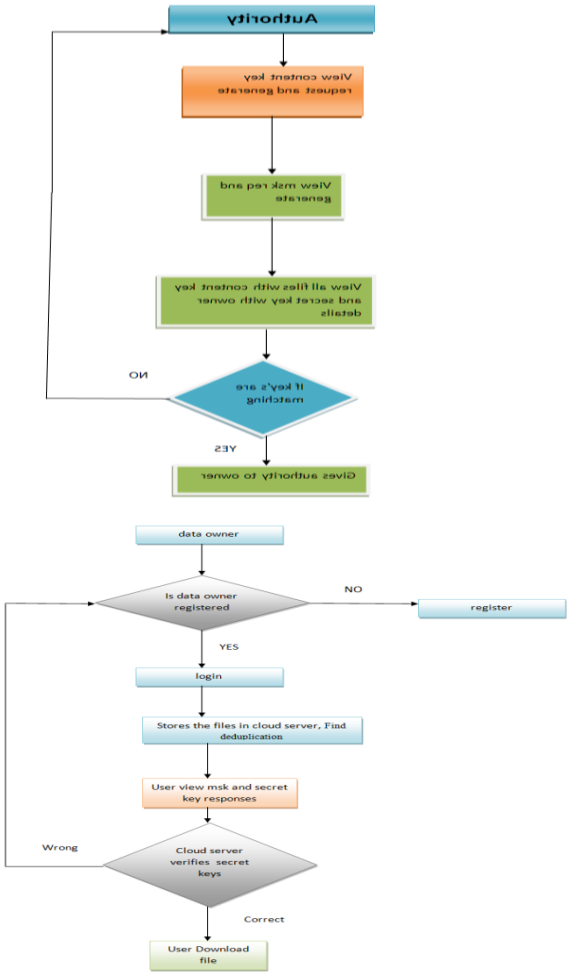
3. Authority:

Authority is in charge of generating the content key and the secret key that the end user has requested. With the content key and master secret key generated with the appropriate data owner details of the particular file, authority can examine all files.

4. End User:

To access files in the cloud, the user must first register and login. The cloud has given the user permission to validate the registration. To download the file, the user must first request the MSK master secret key and content key. Only if the data owner of the file has granted permissions can the user download and search the file.

FLOWCHART:



6. SYSTEM CODING

DBCONN.JAVA

```
<%@ page import="java.sql.*"%>
<%@ page import="java.util.*" %>
<%
Connection connection = null;
 try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/p4","root","root");
    String sql="";
}
catch(Exception e)
{
System.out.println(e);
}
%>
```

BLOCKUSER.JSP

```
<%@page import="java.net.InetAddress"%>
<%@page import="javax.swing.JOptionPane"%>
<%@ page import="java.sql.*"%>
<%@ include file="connect.jsp" %>
<%@ page import="java.util.Date" %>
```

```
<%@ page import="com.oreilly.servlet.*"%>
<%@ page import ="java.text.SimpleDateFormat" %>
<%@ page import ="javax.crypto.Cipher" %>
<%@ page import ="org.bouncycastle.util.encoders.Base64" %>
<%
        String type="attacker";
   try{
        String username=request.getParameter("userid");
      application.setAttribute("aid",username);
        String ip=InetAddress.getLocalHost().getHostAddress();
        SimpleDateFormat sdfDate = new SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat sdfTime = new SimpleDateFormat("HH:mm:ss");
        Date now = new Date();
        String strDate = sdfDate.format(now);
        String strTime = sdfTime.format(now);
        String dt = strDate + "   " + strTime;
        String sql2="SELECT ip_address FROM attacker where ip_address='"+ip+"' ";

        Statement stmt2 = connection.createStatement();
        ResultSet rs2 =stmt2.executeQuery(sql2);
        if(rs2.next()==true)
        {
         %>
          This IP Address has been Blocked...!!!
        <p><a href="index.html">Home Page</a></p>
<p><a href="attacker_login.jsp">Back</a></p>
        <%
        }
        else if(rs2.next()==false)
        {
        String detail="Attacker Logged In";
        PreparedStatement ps=connection.prepareStatement("insert into
        attacker2(ip_address,details,dt,uname) values(?,?,?,?)");
        ps.setString(1,ip);
        ps.setString(2,detail);

        ps.setString(3,dt);
        ps.setString(4,username);
        response.sendRedirect("u_search_tweet2.jsp");

        }
        }
        catch(Exception e)
        {
                out.print(e);
        }
%>
```

TWEETFILTER.JSP

```
<%@ include file="connect.jsp" %>
<%@ page import="java.io.*"%>
<%@ page import="java.util.*" %>
<%@ page import="java.util.Date" %>
<%@ page import="com.oreilly.servlet.*"%>
<%@ page import ="java.text.SimpleDateFormat" %>
<%
  ArrayList list = new ArrayList();
  ServletContext context = getServletContext();
```

```
 String dirName =context.getRealPath("Gallery/");
 String filter=null,categorie=null,location1=null,sk=null,bin = "", paramname=null;

 FileInputStream fs=null;
 File file1 = null;
 try {

        MultipartRequest multi = new MultipartRequest(request, dirName,      10 * 1024 * 1024);
        Enumeration params = multi.getParameterNames();
        while (params.hasMoreElements())
        {
        paramname = (String) params.nextElement();
        if(paramname.equalsIgnoreCase("categorie"))
        {
                categorie=multi.getParameter(paramname);
        }
        if(paramname.equalsIgnoreCase("filter"))
        {
                filter=multi.getParameter(paramname);

        }
}
        FileInputStream fs1 = null;
        String query1="select * from filter  where categorie='"+categorie+"' and filter='"+filter+"' ";
        Statement st1=connection.createStatement();
        ResultSet rs1=st1.executeQuery(query1);
        if ( rs1.next() )
        {
          out.print("filter Already Exist");
        }
        else
        {
         String strQuery2 = "insert into filter(categorie,filter)   values('"+categorie+"','"+filter+"')";
         connection.createStatement().executeUpdate(strQuery2);
         out.print("Filter Added Successfully");
        }
        }
        catch (Exception e)
        {
          out.println(e.getMessage());
        }
%>
```

BLOCKSTATE.JSP

```
        <%@ include file="connect.jsp" %>
        <%
        try {
                        String id=request.getParameter("id");
                        String str = "Yes";
                Statement st1 = connection.createStatement();
                String query1 ="update user set bstate='"+str+"' where id="+id+" ";
                        st1.executeUpdate (query1);
                        connection.close();
                        response.sendRedirect("admin_all_users.jsp");
        }
        catch(Exception e)
        {
                        out.println(e.getMessage())}
```

7. SYSTEM TESTING:

The goal of testing is to find flaws. Testing is the practice of attempting to find all possible flaws or weaknesses in a work product. It allows you to test the functionality of individual components, subassemblies, assemblies, and/or a finished product. It is the process of testing software to ensure that it meets its requirements and meets user expectations, and that it does not fail in an unacceptable way. There are many different types of tests. Each test type is designed to satisfy a distinct testing need.

8. TEST CASES:

Testcase 1:
Description: Whether the user can register by providing his/her details
Result: User registered successfully
Test case 2:
Description: Whether user can login by providing his/her credentials
Result: User login failed because the administrator not authorized.
Test case 2:
Description: Whether user can login by providing his/her credentials
Result: User logged in successfully because the user authorized by adminstrator
Test case 3:
Description: User search his friends by entering his name and send request
Result: Friend displayed as a result and request sent successfully
Test case 4:
Description: User accepting friend request
Result: Friends request accepted and both are friends now
Test case 5:
Description: User creating tweet by entering Tweet name, Picture, Description
Result: Tweet Created successfully
Test case 6:
Description: User search tweet and comment the tweet posted by his/her friend
Result: Commented successfully and filtered
Test case 7:
Description: Administrator viewing trending tweets
Result: Trending tweet displayed based on ranking
Test case 8:
Description: Administrator adding tweet filter to trending topic, both positive and negative words
Result: Filter set added successfully

9.CONCLUSION:

Cloud computing has made attribute-based encryption (ABE) popular, allowing data providers to outsource encrypted data to the cloud and share it with users who have specific business credentials. Deduplication, on the other hand, is a useful approach for conserving storage space and network bandwidth by removing redundant copies of identical data. Standard ABE systems, on the other hand, do not offer safe deduplication, making them prohibitively expensive to use in some commercial storage services. We offered a novel technique to realizing an attribute-based storage system that supports secure deduplication in this work. Our storage system is based on a hybrid cloud architecture, in which a private cloud controls compute and a public cloud controls storage. The private cloud is given a trapdoor key associated with the matching ciphertext, which it can use to transfer the ciphertext from one access policy to ciphertexts of the same plaintext from any other access policies without knowing the plaintext.

References

[1] S. Keelveedhi, M. Bellare, and T. Ristenpart, “Dupless: Serveraidedencryption for deduplicated storage,” in Proceedings of the22th USENIX Security Symposium, Washington, DC, USA, August14-16, 2013. USENIX Association, 2013, pp. 179–194.

[2] Sahai and B. Waters, “Fuzzy identity-based encryption,” in Advancesin Cryptology – EUROCRYPT 2005, 24th Annual InternationalConference on the Theory and Applications of Cryptographic Techniques,Aarhus, Denmark, May 22-26, 2005, Proceedings, ser. Lecture Notesin Computer Science, vol. 3494. Springer, 2005, pp. 457–473

[3] K. R. Choo, M. Herman, M. Iorga, and B. Martini, “Cloud forensics:State-of-the-art and future directions,” Digital Investigation,vol. 18, pp. 77–78, 2016