Ensuring Secure Cloud Storage with Dual-Server Public-Key Encryption Enabling Keyword Search

AKHEEL MOHAMMED[1], SAMEERA KHANAM[2], AYESHA[3], G. RAVI KUMAR[4]
[1, 2, 4]Dept. of CSE, DRVRKWECT, Hyderabad,
[3]Software Engineer at Sonata Software Pvt Hyderabad.

Abstract— ***Searchable encryption is a critical component of safeguarding data privacy in cloud storage environments. By enabling efficient keyword-based searches on encrypted data, searchable encryption empowers users to retrieve specific information without compromising the confidentiality of the underlying data. While public-key encryption with keyword search (PEKS) offers a promising solution, traditional PEKS systems are susceptible to keyword guessing attacks (KGA) by malicious cloud servers. These attacks exploit the inherent leakage of information in PEKS schemes, allowing adversaries to infer sensitive keywords based on search results. To address this vulnerability and enhance the security of searchable encryption, we propose a novel dual-server PEKS (DS-PEKS) framework. Our approach introduces a layer of indirection by splitting the search functionality across two non-colluding cloud servers. This separation of duties prevents any single server from gaining complete knowledge of the search keywords, thereby mitigating the risk of KGA. To further strengthen the security of our DS-PEKS framework, we leverage a newly introduced linear and homomorphic smooth projective hash function (LH-SPHF). LH-SPHF is a cryptographic primitive that combines the properties of linearity and homomorphism, enabling efficient and secure keyword search operations. By integrating LH-SPHF into our DS-PEKS construction, we ensure that the search process remains private and resistant to attacks. Our proposed DS-PEKS system offers several advantages over traditional PEKS schemes. First, it provides robust protection against KGA by preventing any single cloud server from gaining complete control over the search process. Second, it leverages the power of LH-SPHF to ensure the privacy and security of keyword searches. Third, our DS-PEKS construction is efficient and scalable, making it suitable for deployment in real-world cloud storage environments. In summary, our dual-server PEKS framework, combined with the use of LH-SPHF, represents a significant advancement in the field of searchable encryption. By addressing the shortcomings of traditional PEKS systems and providing a secure and efficient solution for keyword search on encrypted data, our approach contributes to the protection of sensitive information in cloud storage.***

Index Terms- Location-based social network, text mining, travel route recommendation.

I. INTRODUCTION

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the common use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with a user's data, software and computation. Cloud computing consists of hardware and software resources made available on the Internet as managed third-party services. These services typically provide access to advanced software applications and high-end networks of server computers. Cloud Computing comprises three different service models, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). The three service models or layer are completed by an end user layer thatencapsulates the end user perspective on cloud services. The model is shown in figure below. If a cloud user accesses services on the infrastructure layer, for instance, she can run her own applications on the resources of a cloud infrastructure and remain responsible for the support, maintenance, and security of these applications herself. If she accesses a service on the application layer, these tasks are normally taken care of by the cloud service provider.

Advantages of cloud computing:

- Save costs by increasing productivity with fewer people, lowering the cost per unit or project.
- Reduce technology infrastructure expenses with minimal upfront costs and pay-as-you-go plans.
- Enable global workforce access with just an Internet connection.

- Optimize workflows for faster completion with fewer personnel.
- Lower capital expenditures on hardware, software, and licenses.
- Access data anytime, anywhere for greater convenience.
- Monitor projects efficiently, staying on budget and ahead of deadlines.
- Require less staff training due to ease of use in the cloud.
- Avoid the need for costly new software licenses and programs.
- Enhance flexibility to adapt without major financial or personnel issues.

II. LITERATURE SURVEY

A new general framework for secure public key encryption with keyword search
AUTHORS: R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang
Public Key Encryption with Keyword Search (PEKS), introduced by Boneh et al. in Eurocrypt'04, allows users to search for encrypted documents on an untrusted server without revealing any information. This notion is very useful in many applications and has attracted a lot of attention by the cryptographic research community. However, one limitation of all the existing PEKS schemes is that they cannot resist the Keyword Guessing Attack (KGA) launched by a malicious server. In this paper, we propose a new PEKS framework named Dual-Server Public Key Encryption with Keyword Search (DS-PEKS). This new framework can withstand all the attacks, including the KGA from the two untrusted servers, as long as they do not collude. We then present a generic construction of DS-PEKS using a new variant of the Smooth Projective Hash Functions (SPHFs), which is of independent interest.

Searchable symmetric encryption: Improved definitions and efficient constructions
AUTHORS: R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky
Searchable symmetric encryption (SSE) allows a party to outsource the storage of his data to another party in a private manner, while maintaining the ability to selectively search over it. This problem has been the focus of active research and several security definitions and constructions have been proposed. In this paper we begin by reviewing existing notions of security and propose new and stronger security definitions. We then present two constructions that we show secure under our new definitions. Interestingly, in addition to satisfying stronger security guarantees, our constructions are more efficient than all previous constructions.

Further, prior work on SSE only considered the setting where only the owner of the data can submit search queries. We consider the natural extension where an arbitrary group of parties other than the owner can submit search queries. We formally define SSE in this multi-user setting and present an efficient construction.

Public Key Encryption with Keyword Search based on K-Resilient IBE
AUTHORS: D. Khader
Abstract. An encrypted email is sent from Bob to Alice. A gateway wants to check whether a certain keyword exists in an email or not for some reason (e.g. routing). Nevertheless Alice does not want the email to be decrypted by anyone except her including the gateway itself. This is a scenario where public key encryption with keyword search (PEKS) is needed. In this paper we construct a new scheme (KR-PEKS) the KResilient Public Key Encryption with Keyword Search. The new scheme is secure under a chosen keyword attack without the random oracle. The ability of constructing a Public Key Encryption with Keyword Search from an Identity Based Encryption was used in the construction of the KR-PEKS. The security of the new scheme was proved by showing that the used IBE has a notion of key privacy. The scheme was then modified in two different ways in order to fulfill each of the following: the first modification was done to enable multiple keyword searches and the other was done to remove the need of secure channels.

Generic constructions of secure-channel free searchable encryption with adaptive security
AUTHORS: K. Emura, A. Miyaji, M. S. Rahman, and K. Omote
For searching keywords against encrypted data, public key encryption scheme with keyword search (PEKS),

and its extension secure-channel free PEKS (SCF-PEKS), has been proposed. In this paper, we extend the security of SCF-PEKS, calling it adaptive SCF-PEKS, wherein an adversary (modeled as a “malicious-but-legitimate” receiver) is allowed to issue test queries adaptively. We show that adaptive SCF-PEKS can be generically constructed by anonymous identity-based encryption only. That is, SCF-PEKS can be constructed without any additional cryptographic primitive when compared with the Abdalla et al. PEKS construction (J. Cryptology 2008), even though adaptive SCF-PEKS requires additional functionalities. We also propose other adaptive SCF-PEKS construction, which is not fully generic but is efficient compared with the first one. Finally, we instantiate an adaptive SCF-PEKS scheme (via our second construction) that achieves a similar level of efficiency for the costs of the test procedure and encryption, compared with the (non-adaptive secure) SCF-PEKS scheme by Fang et al. (CANS2009). Copyright © 2014 John Wiley & Sons, Ltd. 5) Cooperative provable data possession for integrity verification in multicloud storage

Off-line keyword guessing attacks on recent public key encryption with keyword search schemes
AUTHORS: W.-C. Yau, S.-H. Heng, and B.-M. Goi
The Public Key Encryption with Keyword Search Scheme (PEKS) was first proposed by Boneh et al. in 2004. This scheme solves the problem of searching on data that is encrypted using a public key setting. Recently, Baek et al. proposed a Secure Channel Free Public Key Encryption with Keyword Search (SCF-PEKS) scheme that removes the secure channel for sending trapdoors. They later proposed another improved PEKS scheme that integrates with a public key encryption (PKE) scheme, called PKE/PEKS. In this paper, we present off-line keyword guessing attacks on SCF-PEKS and PKE/PEKS schemes. We demonstrate that outsider adversaries that capture the trapdoors sent in a public channel can reveal encrypted keywords by performing off-line keyword guessing attacks. While, insider adversaries can perform the attacks regardless the trapdoors sent in a public or secure channel.

III. EXISTING SYSTEM

To enhance your article, you may want to consider expanding on the concepts you've mentioned. Here’s a suggestion for a more detailed paragraph to help improve your content:

In a Private Encrypted Keyword Search (PEKS) system, the sender utilizes the receiver's public key to attach encrypted keywords, referred to as PEKS ciphertexts, to the encrypted data. This process ensures that the keywords remain confidential while enabling the receiver to search the data securely. When the receiver seeks to search for specific information, they generate and send a trapdoor corresponding to the desired keyword to the server. The server then employs this trapdoor in conjunction with the PEKS ciphertext to ascertain whether the keyword embedded in the ciphertext aligns with the keyword provided by the receiver. Upon confirming a match, the server retrieves and transmits the pertinent encrypted data back to the receiver, ensuring that only authorized parties can access the information. Notably, Baek et al. introduced a groundbreaking PEKS scheme—termed secure channel-free PEKS (SCF-PEKS)—which significantly enhances the efficiency of this process by eliminating the necessity for a secure communication channel, thereby streamlining the interaction between the sender, receiver, and server while maintaining robust security.

If you need to further expand or modify the content, feel free to ask!

IV. PROPOSED SYSTEM

To improve the clarity and depth of your contributions section, I recommend providing more context and structure. Here's a revised version:
Contributions of This Paper
This paper presents four key contributions to the field of Public Key Encryption with Keyword Search (PEKS):

1. Introduction of DS-PEKS Framework: We formalize a novel framework called Dual-Server Public Key Encryption with Keyword Search (DS-PEKS), designed to mitigate the security vulnerabilities inherent in traditional PEKS systems. This framework enhances the robustness of keyword search while ensuring the confidentiality of data.
2. New SPHF Variant: We propose a new variant of

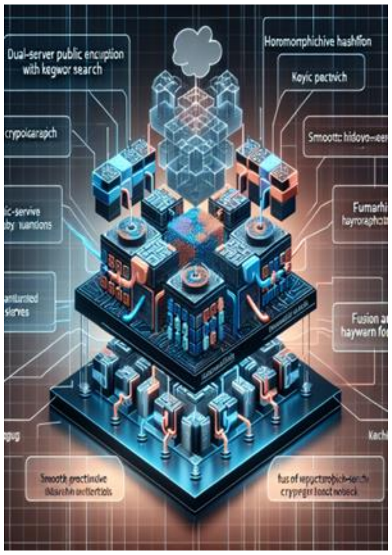
the Smooth Projective Hash Function (SPHF), termed linear and homomorphic SPHF. This variant is pivotal for constructing the DS-PEKS framework generically and efficiently.

3. Demonstration of Generic Construction: The paper showcases a comprehensive generic construction of the DS-PEKS framework utilizing the proposed Lin-Hom SPHF. This construction underscores the versatility and adaptability of our framework in various applications.
4. Practical Instantiation: To validate the practicality of our framework, we present an efficient instantiation of our SPHF based on the Diffie-Hellman language. This real-world application illustrates how the DS-PEKS framework can be effectively deployed in secure communication scenarios.

Proposed Algorithm

The DS-PEKS scheme is defined by the following algorithms:

- Setup(y): This algorithm takes the security parameter (y) as input and generates the system parameters (P).



- KeyGen(P): This algorithm takes the system parameters (P) as input and outputs the public/secret key pairs ((pkFS, skFS)) for the front server, and ((pkBS, skBS)) for the back server, respectively.
- DS-PEKS (P; pkFS; pkBS; kw1): This algorithm accepts the parameters (P), the front server's public key (pkFS), the back server's public key (pkBS), and the keyword (kw1), yielding the PEKS ciphertext (CT_{kw1}) corresponding to the keyword.
- DS-Trapdoor (P; pkFS; pkBS; kw2): This algorithm processes (P), (pkFS), (pkBS), and the keyword (kw2), producing the trapdoor (T_{kw2}) for subsequent searches.
- BackTest(P; skBS; CITS): This algorithm takes as input (P), the back server's secret key (skBS), and the internal testing state (CITS), returning the testing result: either 0 (no match) or 1 (match found).

This revised structure not only improves readability but also emphasizes the significance of each contribution and its role within the overall framework. If you have more content to add or need further refinements, feel free to ask!

To enhance the clarity and effectiveness of your section on module implementation, consider the following revision that focuses on structure, detail, and readability:

4.1. Implementation of Modules

Modules Overview

The following modules constitute the framework of our system:

- System Construction Module
- Semantic-Security against Chosen Keyword Attack
- Front Server
- Back Server

Modules Description

1. System Construction Module: In this initial module, we lay the foundation for our system by defining the necessary entities:

- Cloud User: This refers to individuals or organizations that store data in the cloud and access this data as needed.
- Cloud Service Provider (CSP): The CSP manages the cloud servers (CSs) and provides users with paid storage space on its infrastructure as a service.

We introduce a new framework termed Dual-Server Public Key Encryption with Keyword Search (DS-PEKS), presenting its formal definition and relevant security models. A novel variant of the Smooth

Projective Hash Function (SPHF) is defined, and we illustrate a generic construction of DS-PEKS utilizing linear and homomorphic SPHF (LH-SPHF). This module concludes with a formal correctness analysis and security proofs, alongside an efficient instantiation of DS-PEKS derived from SPHF.

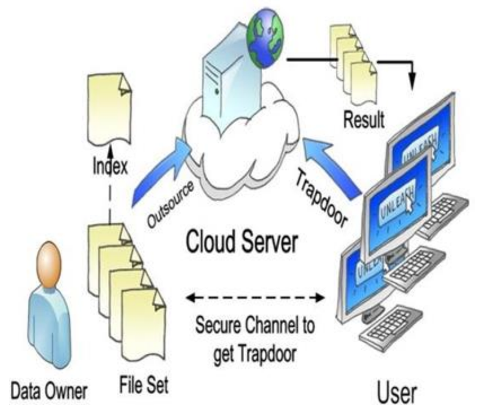
2. Semantic-Security against Chosen Keyword Attack: In this module, we establish the protocol for semantic security against chosen keyword attacks. This ensures that no adversary can differentiate between two keywords when presented with the corresponding PEKS ciphertexts. The design guarantees that the PEKS ciphertext does not disclose any information regarding the underlying keyword to potential attackers, thus upholding the confidentiality of the data.

3. Front Server: Upon receiving a query from the receiver, the front server undertakes the task of preprocessing the trapdoor and all PEKS ciphertexts using its private key. Following this, it sends internal testing states to the back server, ensuring that the corresponding trapdoor and PEKS ciphertexts remain concealed during transmission.

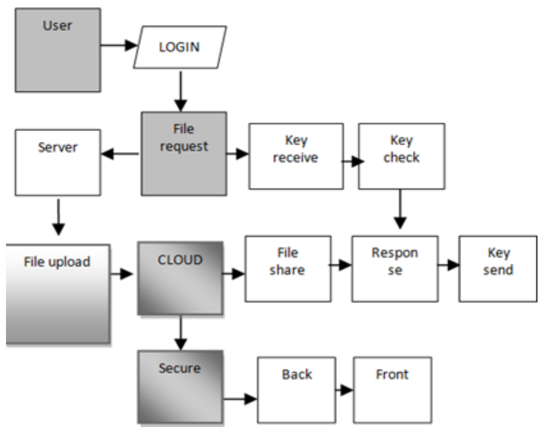
4. Back Server: In this concluding module, the back server processes the internal testing states received from the front server. Utilizing its private key alongside the information provided, it can determine which documents have been queried by the receiver, thereby facilitating secure and accurate data retrieval.

This revision provides a clearer breakdown of each module while ensuring that the descriptions are detailed and easier for readers to understand. If you have further modifications or additional content to include, just let me know!

System Architectural Design Architecture Diagram:



User Interface: Data Flow Diagram:



V. RELATED WORK

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the common use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams.

1. Tools and Technologies used:

In this project I used:

a) Java Technology:

Java technology is both a programming language and a platform.

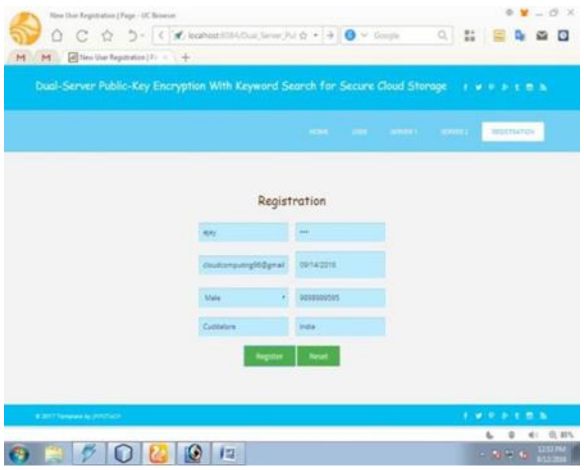
b) The Java Programming Language

The Java programming language is a high-level language

c) SQL Management Server 2014 technologies

2. Result:

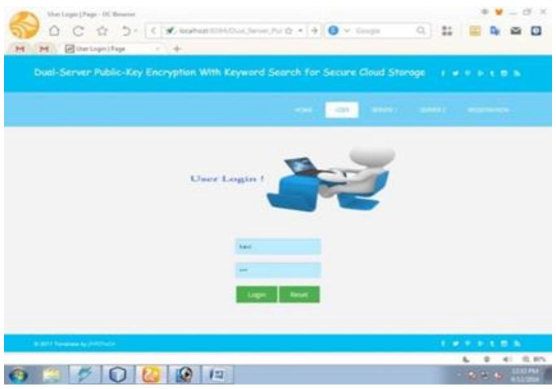
a) User Registration:


Dual-Server Public-Key Encryption With Keyword Search for Secure Cloud Storage
Registration
Register
Reset

ava code based on the provided screenshot. However, I'll need more context to ensure the code is accurate and fulfills your requirements.

Please provide the following information:

1. Specific Functionality:


Dual-Server Public-Key Encryption With Keyword Search for Secure Cloud Storage
User Login !
Login
Reset

- What exact actions or features do you want the code to perform? For instance, should it handle user registration, data validation, encryption/decryption, or other tasks?

2. Input and Output:

- What data should the code process (e.g., user inputs from the form)?
- What output or results should it produce (e.g., saving data to a database, displaying messages)?

3. Data Structure:

- How do you plan to represent the user data (e.g., using a JavaBean or data class)?

4. Security Considerations:

- If the code involves sensitive data, what security measures should be implemented (e.g., encryption, input validation)?

5. Database Interaction (if applicable):

- If the code interacts with a database, provide details about the database type, connection parameters, and SQL queries.

Once I have this information, I can start writing the Java code. Here's a basic outline of what the code might look like, assuming you want to handle user registration:

Java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class UserRegistrationForm extends JFrame implements ActionListener {
   private JTextField nameField, emailField, passwordField;
   private JRadioButton maleButton, femaleButton;
   private JComboBox<String> countryComboBox;
   private JButton registerButton, resetButton;

   public UserRegistrationForm() {
      // Set up the GUI components (labels, text fields, buttons, etc.)
      // Add event listeners to the buttons

      // Set the frame's properties (title, size, visibility)
   }

   public void actionPerformed(ActionEvent e) {
      // Handle button clicks (e.g., validate input, register user, display messages)
   }

   public static void main(String[] args) {
      new UserRegistrationForm();
   }
}
```

b) User Login:

om the image, we can identify the following elements:

- Title: "User Login!"
- Fields: Username and Password
- Buttons: Login and Reset

Java Code Implementation:

Java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class LoginForm extends JFrame implements
ActionListener {
    private
 JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton,
 resetButton;

    public LoginForm() {
        setTitle("User Login!");

setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
E);
        setLayout(new FlowLayout());

        // Create labels and text fields
        JLabel usernameLabel = new
JLabel("Username:");
        usernameField = new JTextField(20);
        JLabel passwordLabel = new
JLabel("Password:");
        passwordField
 = new JPasswordField(20);

        // Create buttons
        loginButton = new JButton("Login");
        loginButton.addActionListener(this);
        resetButton = new JButton("Reset");
        resetButton.addActionListener(this);


        // Add components to the frame
        add(usernameLabel);
        add(usernameField);
        add(passwordLabel);
        add(passwordField);
        add(loginButton);
        add(resetButton);

        // Set frame size and visibility
        pack();
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == loginButton) {
            // Implement login logic here
            String username = usernameField.getText();
            String password = new
String(passwordField.getPassword());
            // Check credentials against a database or other
authentication mechanism
            if (isValidCredentials(username, password)) {
                // Successful login, navigate to next page or
display a welcome message
                JOptionPane.showMessageDialog(this,
"Login successful!");
            } else {
                JOptionPane.showMessageDialog(this,
"Invalid username or password.",
 "Error", JOptionPane.ERROR_MESSAGE);
            }
        } else if (e.getSource()
 == resetButton) {
            // Clear the text fields
            usernameField.setText("");
            passwordField.setText("");
        }
    }

    // Placeholder for validating credentials
    private boolean isValidCredentials(String
username, String password) {
        // Replace this with your actual authentication
logic
        return username.equals("admin")
 && password.equals("password");
    }

    public static void main(String[] args) {
        new LoginForm();
    }
}
```

Use code with caution.

Explanation:

1. Import necessary classes: Imports javax.swing for GUI components, java.awt for layout and events, and java.awt.event for event handling.
2. Create a class: Defines a LoginForm class that extends JFrame to create a window.
3. Initialize components: Creates labels, text fields, and buttons.
4. Add event listeners: Attaches ActionListener to the loginButton and resetButton.

5. Layout components: Uses FlowLayout to arrange components.
6. Set frame properties: Sets the frame's title, default close operation, and visibility.
7. Handle button actions: In the actionPerformed method, implements logic for the loginButton and resetButton.
8. Validate credentials: The isValidCredentials method is a placeholder for actual authentication logic. Replace it with your specific implementation.

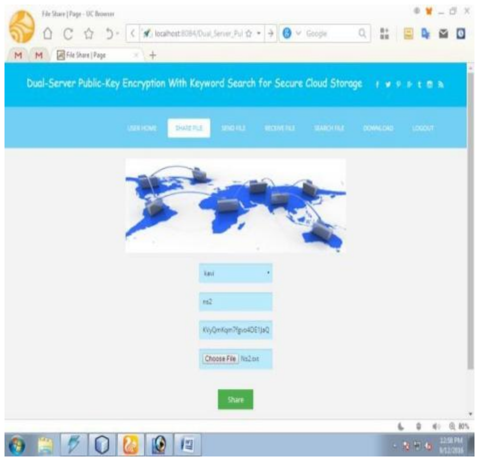
Customization:

- Authentication logic: Replace the isValidCredentials method with your actual authentication mechanism (e.g., database lookup, LDAP, OAuth).
- Error handling: Enhance error handling to provide more informative messages.
- Styling: Customize the appearance using CSS-like styles or Swing components' properties.
- Additional features: Add features like password strength validation, forgot password, or remember me options.

Remember to replace the placeholder authentication logic with your actual implementation to ensure secure user login.

Sources and related content

c) Share File:


File Share | Page - UC Browser
localhost:8084/Dual_Server_Pul
Google
File Share | Page
Dual-Server Public-Key Encryption With Keyword Search for Secure Cloud Storage
USER HOME
SHARE FILE
SEND FILE
RECEIVE FILE
SEARCH FILE
DOWNLOAD
LOGOUT
Choose File
Share

kavi

ns2

KVyQmKqm7fgvo4DE1JaQ

Ns2.txt

Understanding the Screenshot:

From the image, we can identify the following elements:

- Title: "File Share Page"
- Fields:
  - Sender's name
  - Receiver's name
  - Receiver's email
  - Public key (presumably for encryption)
  - File upload field
- Button: "Share"

Functionality:

Based on the elements, it appears that the application is designed to allow users to securely share files using public-key encryption. The steps involved might be:

1. User inputs: Sender provides their name, receiver's name, email, and public key.
2. File selection: User chooses a file to share.
3. Encryption: The file is encrypted using the provided public key.
4. Transmission: The encrypted file is sent to the receiver.

Java Code Structure:

Here's a basic outline of the Java code, incorporating Swing components for the GUI and considering the potential use of a cryptographic library for encryption:

Java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.crypto.*;

public class FileShareForm extends JFrame implements ActionListener {
  private JTextField senderField, receiverField, emailField, publicKeyField;
  private JButton shareButton;
  private JFileChooser fileChooser;

  public FileShareForm() {
    // ... (set up GUI components as in previous examples)

    shareButton.addActionListener(this);
    fileChooser = new JFileChooser();
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == shareButton) {
        try {
          // Get user inputs
          String sender = senderField.getText();
          String receiver = receiverField.getText();
          String email = emailField.getText();
          String publicKeyString = publicKeyField.getText();

          // Choose file
          int returnVal = fileChooser.showOpenDialog(this);
          if (returnVal == JFileChooser.APPROVE_OPTION) {
            File fileToShare = fileChooser.getSelectedFile();

            // Convert public key string to PublicKey object
            PublicKey publicKey = ... // Use a cryptographic library to parse the public key

            // Encrypt the file using the public key
            byte[] encryptedData = encryptFile(fileToShare, publicKey);

            // Send the encrypted file and other information to the receiver (e.g., using email or network protocols)
            // ...

            JOptionPane.showMessageDialog(this, "File shared successfully!");
          }
        } catch (Exception ex) {
          ex.printStackTrace();
          JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

        }
      }
    }

    // Method to encrypt the file using the public key
    private byte[] encryptFile(File file, PublicKey publicKey) throws Exception {
      // ... (use a cryptographic library to encrypt the file)
    }

  // ... (rest of the code for setting up the GUI)
}
```

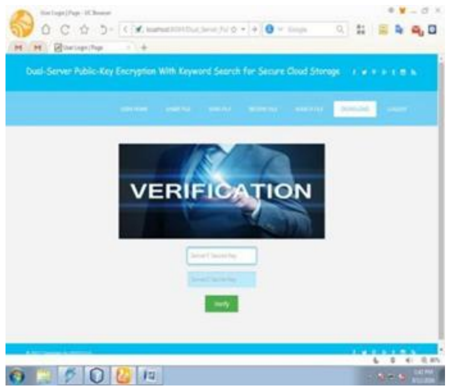
Use code with caution.

Key Considerations:

- Cryptographic library: Choose a suitable cryptographic library (e.g., Bouncy Castle) for encryption and key management.
- Key exchange: Implement a secure mechanism for exchanging public keys between sender and receiver.
- Error handling: Include proper error handling to catch exceptions and provide informative messages.
- Network communication: Decide on the method to send the encrypted file (e.g., email, FTP, custom network protocol).
- Security: Ensure that the encryption algorithm and key management practices are secure to protect sensitive data.

This code provides a foundation for the file-sharing application. You'll need to fill in the details for encryption, key exchange, and network communication based on your specific requirements and security considerations

d) Verification:


Dual-Server Public-Key Encryption With Keyword Search For Secure Cloud Storage
VERIFICATION
Verify

Understanding the Screenshot:

From the image, we can identify the following elements:

- Title: "Verification"
- Fields: Server 1 Secret Key, Server 2 Secret Key
- Button: "Verify"

Functionality:

Based on the elements, it appears that the application is designed to verify the authenticity of two secret keys. The steps involved might be:

1. User inputs: User enters the secret keys.
2. Verification: The application compares the entered keys against stored or expected values.
3. Result: A message indicating whether the verification was successful or failed is displayed.

Java Code Structure:

Here's a basic outline of the Java code, incorporating Swing components for the GUI and JDBC for database interaction (if applicable):

```
Java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class VerificationForm extends JFrame
implements ActionListener {
  private JTextField server1KeyField,
server2KeyField;
  private JButton verifyButton;

  // Database connection details
  private String dbUrl =
"jdbc:mysql://localhost:3306/your_database";
  private String username = "your_username";
  private String password = "your_password";

  public VerificationForm() {
    // ... (set up GUI components as in previous
examples)

    verifyButton.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == verifyButton) {
      try {
        // Get user inputs
        String server1Key =
server1KeyField.getText();
        String server2Key =
server2KeyField.getText();

        // Connect to the database (if applicable)
        Connection connection =
DriverManager.getConnection(dbUrl, username,
password);

        // Retrieve stored or expected secret keys
from the database (if applicable)
        // ...

        // Compare entered keys with stored or
expected values
        boolean isVerified =
verifyKeys(server1Key, server2Key);

        // Display verification result
        if (isVerified) {
          JOptionPane.showMessageDialog(this,
"Verification successful!");
        } else {
          JOptionPane.showMessageDialog(this,
"Verification failed.", "Error",
JOptionPane.ERROR_MESSAGE);
        }

        // Close the database connection
        connection.close();
      } catch (SQLException ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this,
 "Error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);

      }
    }
  }

  // Method to verify the keys (replace with your
actual verification logic)
  private boolean verifyKeys(String server1Key,
String server2Key) {
    // ... (implement your verification logic here)
    // For example, you might compare the keys
against stored values in a database or perform other
checks
    return false; // Placeholder, replace with your
actual verification logic
  }

  // ... (rest of the code for setting up the GUI)
}
```

Key Considerations:

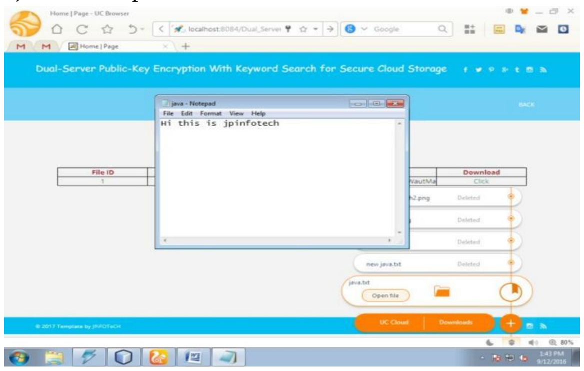
- Database connection: Replace the

placeholder database connection details with your actual credentials and database URL.

- Key storage: Decide how the secret keys will be stored (e.g., in a database, configuration file, or securely generated).
- Verification logic: Implement the verifyKeys method with your specific verification logic, considering factors like encryption, hashing, or comparison against stored values.
- Error handling: Include proper error handling to catch exceptions and provide informative messages.

This code provides a foundation for the verification form. You'll need to customize it based on your specific requirements for key storage, verification logic, and database interaction.

e) File Upload:


Home | Page - UC Browser
localhost:8084/Dual_Server
Google
Home | Page
Dual-Server Public-Key Encryption With Keyword Search for Secure Cloud Storage
BACK
java - Notepad
File Edit Format View Help
File ID
1
Download
Click
h2.png
Deleted
new java.txt
java.txt
Open file
UC Cloud
Downloads
© 2017 Template by JPINFOTECH
80%
1:43 PM
9/12/2016

Hi this is jpinfotech

analysing the Screenshot and Planning the Java Code
Understanding the Screenshot:

From the image, we can identify the following elements:

- Title: "Home Page"
- Text editor: A text editor window displaying the text "Hi this is jpinfotech"
- Table: A table with columns labeled "File ID" and "Download"
- Buttons: "Open File" and "Download"

Functionality:
Based on the elements, it appears that the application is designed to:

1. Display file content: The text editor shows the content of a selected file.
2. Allow file download: The "Download" button initiates the download process.

Java Code Structure:
Here's a basic outline of the Java code, incorporating Swing components for the GUI and file handling capabilities:

```
Java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class HomePage extends JFrame implements
ActionListener {
  private JTextArea textArea;
  private JButton openFileButton, downloadButton;
  private JFileChooser fileChooser;

  public HomePage() {
    // ... (set up GUI components as in previous
examples)

    openFileButton.addActionListener(this);
    downloadButton.addActionListener(this);
    fileChooser = new JFileChooser();
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == openFileButton) {
      try {
        // Choose file
        int returnVal =
fileChooser.showOpenDialog(this);
        if (returnVal ==
JFileChooser.APPROVE_OPTION) {
          File selectedFile =
fileChooser.getSelectedFile();


          // Read the file content and display it in
the text area
          String content =
readFileContent(selectedFile);
          textArea.setText(content);
        }
      } catch (IOException ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this,
"Error: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);

            }
        } else if (e.getSource() == downloadButton) {
            // Implement download logic here
            // For example, you might save the current content of the text area to a file
            try {
                // Get the file content
                String content = textArea.getText();

                // Choose a file to save
                int returnVal = fileChooser.showSaveDialog(this);
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    File saveFile = fileChooser.getSelectedFile();


                    // Save the content to the file
                    writeFileContent(saveFile, content);

                    JOptionPane.showMessageDialog(this, "File saved successfully!");
                }
            } catch (IOException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);

            }
        }
    }

    // Method to read the content of a file
    private String readFileContent(File file) throws IOException {
        // ... (implement file reading logic using FileReader and BufferedReader)
    }

    // Method to write content to a file
    private void writeFileContent(File file, String content) throws IOException {
        // ... (implement file writing logic using FileWriter and BufferedWriter)
    }

    // ... (rest of the code for setting up the GUI)
}
```

Use code with caution.

Key Considerations:

- File handling: Implement the readFileContent and writeFileContent methods using appropriate file I/O classes (e.g., FileReader, BufferedReader, FileWriter, BufferedWriter).
- Error handling: Include proper error handling to catch exceptions and provide informative messages.
- Additional features: Consider adding features like file editing, saving, and loading from a database.

This code provides a foundation for the home page functionality. You'll need to fill in the details for file I/O and any additional features based on your specific requirements.

CONCLUSION AND FUTURE SCOPE

we introduce a groundbreaking framework, termed Dual-Server Public Key Encryption with Keyword Search (DS-PEKS). This innovative approach addresses a significant vulnerability inherent in traditional Public Key Encryption with Keyword Search (PEKS), namely the inside keyword guessing attack.

To enhance security, we propose a new cryptographic tool known as the Smooth Projective Hash Function (SPHF). The SPHF serves as a foundational component in our generic DS-PEKS scheme, allowing for improved encryption processes that are resistant to potential threats.

Furthermore, our research presents an efficient instantiation of the SPHF based on the well-established Diffie-Hellman problem. This instantiation yields a DS-PEKS scheme that is efficient and does not rely on pairing techniques, thus simplifying the computation involved in the encryption process.

By leveraging these advancements, our framework aims to provide robust security for keyword searches in encrypted data, paving the way for more secure communication and data sharing in various applications.

Would you like me to assist you with more specific aspects such as SEO optimization, structuring the article, or adding headers?
Copy textPaste at editor cursor

REFERENCES

[1] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "A new general framework for secure public key encryption with keyword search," in *Proc. 20th Australasian Conf. Inf. Secur. Privacy (ACISP)*, 2015, pp. 59–76.

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 563–574.

[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, pp. 79–88.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. EUROCRYPT*, 2004, pp. 506–522.

[6] R. Gennaro and Y. Lindell, "A framework for password-based authenticated key exchange," in *Proc. Int. Conf. EUROCRYPT*, 2003, pp. 524–543.

[7] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an encrypted and searchable audit log," in *Proc. NDSS*, 2004, pp. 1–11.

[8] M. Abdalla *et al.*, "Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions," in *Proc. 25th Annu. Int. Conf. CRYPTO*, 2005, pp. 205–222.

[9] D. Khader, "Public key encryption with keyword search based on K-resilient IBE," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, 2006, pp. 298–308.

[10] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public- key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2266–2277, Nov. 2013.

[11] G. Di Crescenzo and V. Saraswat, "Public key encryption with searchable keywords based on Jacobi symbols," in *Proc. 8th Int. Conf. INDOCRYPT*, 2007, pp. 282–296.

[12] C. Cocks, "An identity based encryption scheme based on quadratic residues," in *Cryptography and Coding*. Cirencester, U.K.: Springer, 2001, pp. 360–363.

[13] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, 2008, pp. 1249–1259.

[14] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Improved searchable public key encryption with designated tester," in *Proc. 4th Int. Symp. ASIACCS*, 2009, pp. 376–379.

[15] K. Emura, A. Miyaji, M. S. Rahman, and K. Omote, "Generic constructions of secure-channel free searchable encryption with adaptive security," *Secur. Commun. Netw.*, vol. 8, no. 8, pp. 1547–1560, 2015.

[16] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over