



# **CSS Assignment 1**

## **Security Tool: SafePass**

By:

Sanjana Joshi-1611083, Div B

Vridhi Patel-1611106, Div B

Pankti Thakkar-1611114, Div B

---

## Abstract

This report talks about **SafePass** which is a security tool based on the existing tool called **KeePass** which gives resolution to the problem of memorizing different passwords. Its purpose is to keep all of the user's passwords, data, email accounts, usernames and URLs stored in a very secure, encrypted database, protected by a Master Password. The system is very small so it can be easily transferred from one computer to another. It provides several functionalities on the already encrypted data and the new ones to be inserted. The database produced, is protected by a Master Password only known by its inventor with no backup if lost.

## Introduction

Today everything on the web needs to be secure with passwords, hence the need to remember many passwords arises. Your passwords are the most common way to prove your identity when using websites, email accounts and your computer itself (via User Accounts). The use of strong passwords is therefore essential in order to protect your security and identity. The best security in the world is useless if a malicious person has a legitimate user's user name and password. It is advisable to have separate passwords for each user account failing which it becomes easy for the hacker to get access to all the user accounts easily.

KeePass is a free open source password manager, which helps you to manage your passwords in a secure way. All the passwords are kept in one database, which is locked with one master key. So the user has to remember one single master password to unlock the whole database. The databases are encrypted using the best and most secure encryption algorithm currently known, that is **AES**. AES is found at least six times faster than triple DES. To date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches. Therefore KeePass is considered a strong security tool. In this assignment, we have created a tool SafePass which is similar in functionality to KeePass.

---

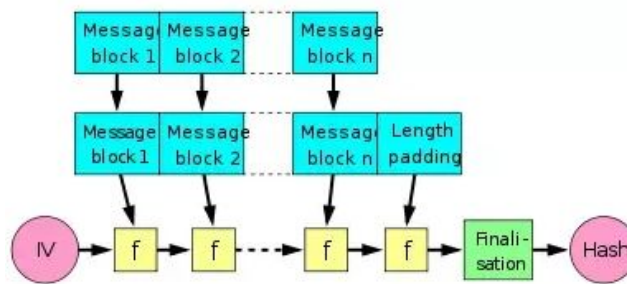
## MD5 Hashing Algorithm:

The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message.

The MD5 message-digest hashing algorithm processes data in 512-bit blocks, broken down into 16 words composed of 32 bits each. The output from MD5 is a 128-bit message digest value.

Computation of the MD5 digest value is performed in separate stages that process each 512-bit block of data along with the value computed in the preceding stage:

- The first stage begins with the message digest values initialized using consecutive hexadecimal numerical values
- Each stage includes four message-digest passes which manipulate values in the current data block and values processed from the previous block
- The final value computed from the last block becomes the MD5 digest for that block



In the diagram, the one-way compression function is denoted by  $f$  and transforms two fixed-length inputs to an output of the same size as one of the inputs. The algorithm starts with an initial value, the initialization vector (IV). The IV is a fixed value (algorithm or implementation-specific).

For each message block, the compression (or compacting) function  $f$  takes the result so far, combines it with the message block, and produces an intermediate result. The last block is padded with zeros as needed and bits representing the length of the entire message are appended.

To harden the hash further the last result is then sometimes fed through a finalization function. The finalization function can have several purposes such as compressing a

---

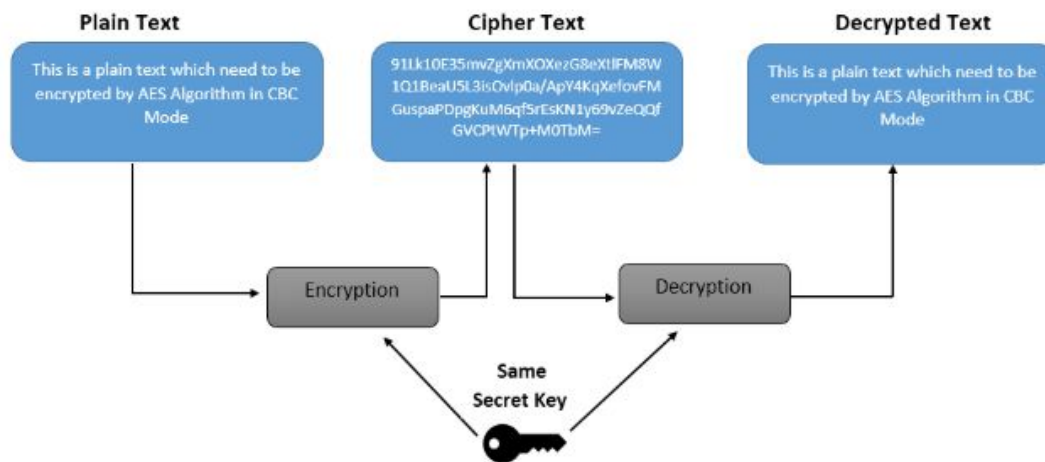
bigger internal state (the last result) into a smaller output hash size or to guarantee a better mixing and avalanche effect on the bits in the hash sum.

## Advanced Encryption Standard (AES):

AES (Advanced Encryption Standard) is a strong encryption and decryption algorithm and more secure than its predecessors DES (Data Encryption Standard) and 3DES (Triple-DES). Since AES Encryption is a Symmetric algorithm we will be using the same Secret Key for both encryption and decryption.

The features of AES are as follows –

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java



AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix.

---

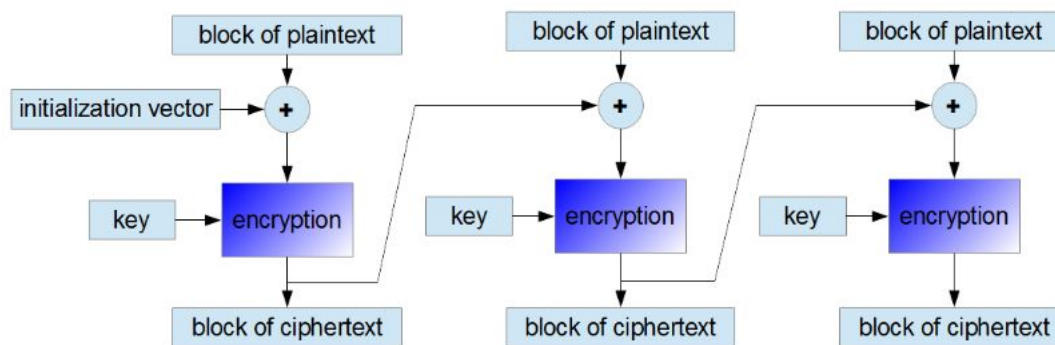
Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

## CBC Mode of Execution

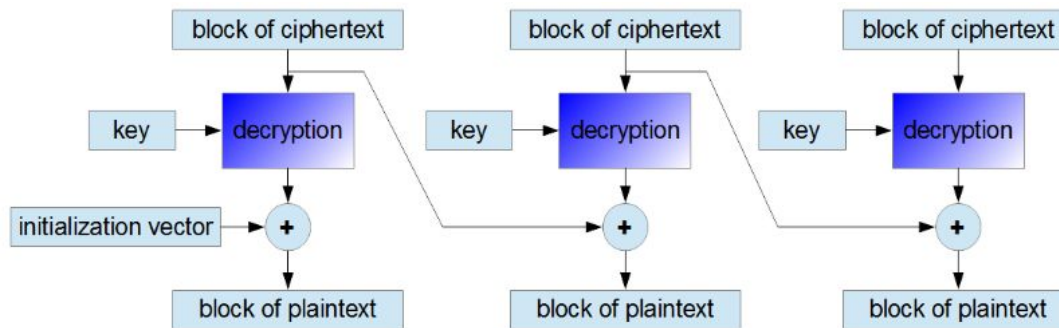
This mode is about adding XOR each plaintext block to the ciphertext block that was previously produced. The result is then encrypted using the cipher algorithm in the usual way. As a result, every subsequent ciphertext block depends on the previous one. The first plaintext block is added XOR to a random initialization vector (commonly referred to as IV). The vector has the same size as a plaintext block.

Encryption in CBC mode can only be performed by using one thread. Despite this disadvantage, this is a very popular way of using block ciphers. CBC mode is used in many applications.

During decryption of a ciphertext block, one should add XOR the output data received from the decryption algorithm to the previous ciphertext block. Because the receiver knows all the ciphertext blocks just after obtaining the encrypted message, he can decrypt the message using many threads simultaneously.



Encryption in the CBC mode



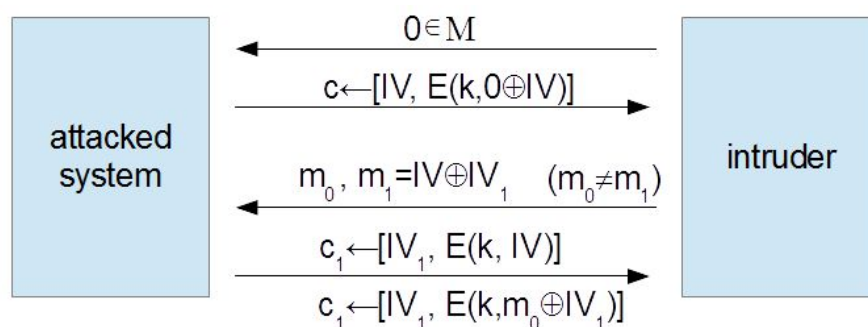
Decryption in the CBC mode

If one bit of a plaintext message is damaged (for example because of some earlier transmission error), all subsequent ciphertext blocks will be damaged and it will be never possible to decrypt the ciphertext received from this plaintext. As opposed to that, if one ciphertext bit is damaged, only two received plaintext blocks will be damaged. It might be possible to recover the data.

A message that is to be encrypted using the CBC mode, should be extended till the size that is equal to an integer multiple of a single block length (similarly, as in the case of using the ECB mode).

## Security of the CBC mode

The initialization vector IV should be created randomly by the sender. During transmission it should be concatenated with ciphertext blocks, to allow decryption of the message by the receiver. If an intruder could predict what vector would be used, then the encryption would not be resistant to chosen-plaintext attacks:



In the example presented above, if the intruder is able to predict that the vector IV1 will be used by the attacked system to produce the response c1, they can guess which one of the two encrypted messages m0 or m1 is carried by the response c1. This situation

---

breaks the rule that the intruder shouldn't be able to distinguish between two ciphertexts even if they have chosen both plaintexts. Therefore, the attacked system is vulnerable to chosen-plaintext attacks.

If the vector IV is generated based on non-random data, for example, the user password, it should be encrypted before use. One should use a separate secret key for this activity. The initialization vector IV should be changed after using the secret key a number of times. It can be shown that even properly created IV used too many times, makes the system vulnerable to chosen-plaintext attacks. For AES cipher it is estimated to be 248 blocks.

## Implementation Details

The project was developed using **Apache NetBeans 11.1**. The front-end was developed using Java Swing. Java Swing is a lightweight Graphical User Interface (GUI) toolkit that includes a rich set of widgets. It includes packages that let you make GUI components for your Java applications, and it is platform-independent.

The Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform-dependent GUI toolkit. But unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent and are referred to as “lightweight” components.

The algorithm used for hashing is MD-5 and for encryption of the data, the AES algorithm in CBC mode is used. In the next section, we will first understand the working of the MD-5 hashing algorithm. Later on, we will see the working of the AES algorithm.

For implementing this Java Cryptography is used. The Java Cryptography API enables one to encrypt and decrypt data in Java, as well as manage keys, sign and authenticate messages, calculate cryptographic hashes and much more.

The packages we've used are -

- java.security
  - MessageDigest - To implement the functionality of the MD-5 message-digest algorithm
  - SecureRandom - To generate cryptographically strong pseudo-random numbers for initialization vectors used in encryption
- java.security.spec

- 
- AlgorithmParameterSpec - To provide the specification of cryptographic parameters
  - java.crypto
    - Cipher - To encrypt and decrypt data
  - java.crypto.spec
    - SecretKeySpec - To specify the secret key used for encryption and decryption
    - IvParameterSpec - To specify an initialization vector (IV)

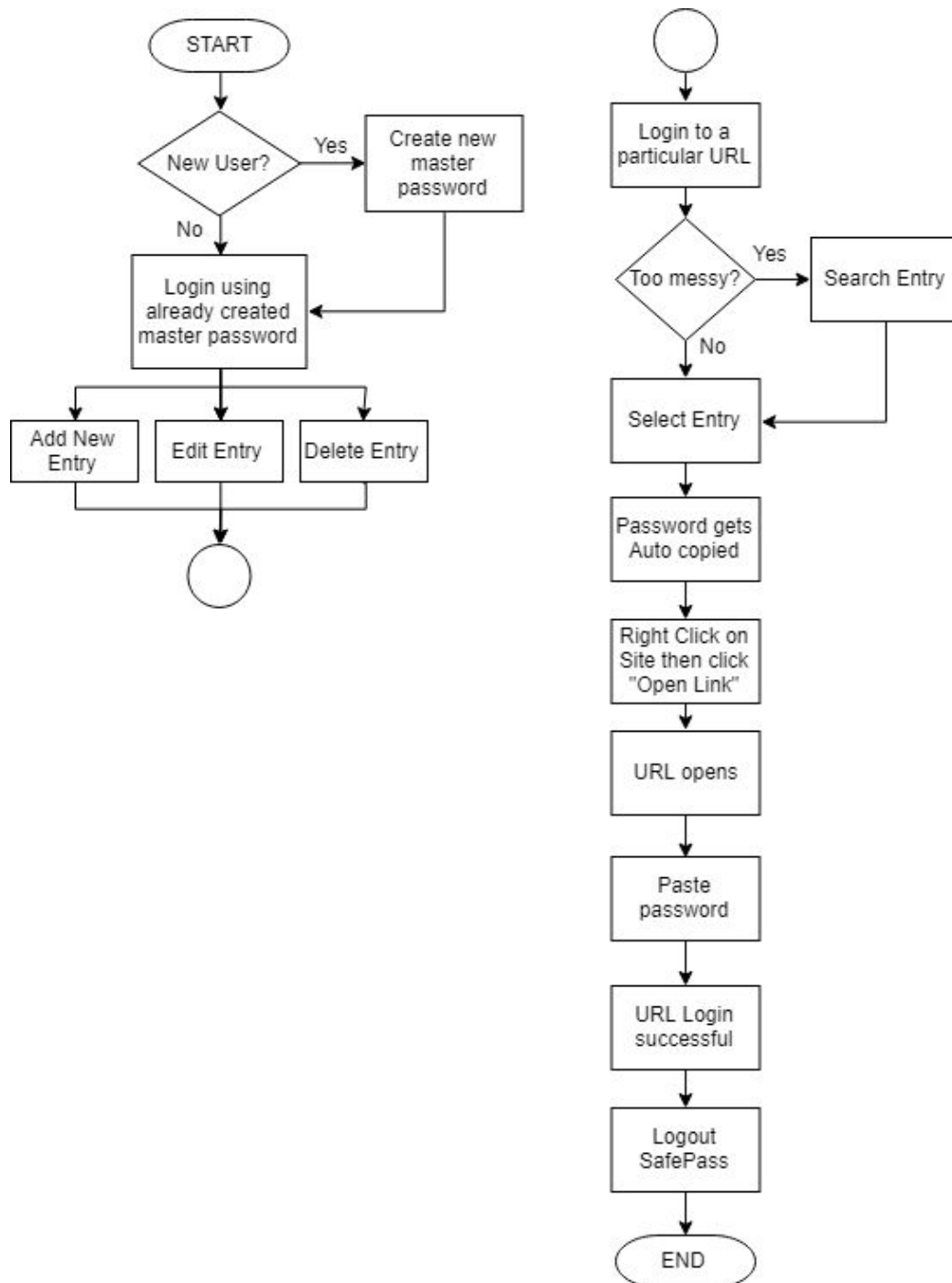
## Working

- New user runs the SafePass.jar file for the first time
- AppDialog.newPasswd() called
  - Call to InputDialog.show() to display the dialog box and user prompted to enter the master password for the first time
- The entered master password is encrypted using md5 and is returned (as key) by AppDialog.newPasswd()
- creatDB() called
  - creates an xml file
  - Root tag <sfpdb> </sfpdb> added
  - Crypto.encryptPassword() called wherein the key is passed and it is encrypted and IV and encrypted data are further encoded with Base64
  - This is the content for <passtest> </passtest> tags
- Apps.passwd() called to login
  - Md5 hash of entered password is calculated and key returned
  - A new instance of DataTable is created and load() method called
    - Call to checkPassword() where it splits encoded iv and data, decodes them (base64 decoding) and then further decryption of AES takes place using iv and key and if it matches, then user authenticated
  - Data from file is loaded using reload() method
- For adding new entry, dialog box displayed
  - User enters the data in respective fields
  - addEntry() method called and a new instance of Entry class is created and toElement() called in which all the fields are encrypted and file updated accordingly



- This data is then transformed to xml schema using Transformer class
- Similarly, to edit an entry, dialog box is displayed and entered data is encrypted and saved to xml
- For deleting, tag contents of that entry are accessed and are removed from database using removeEntry() method

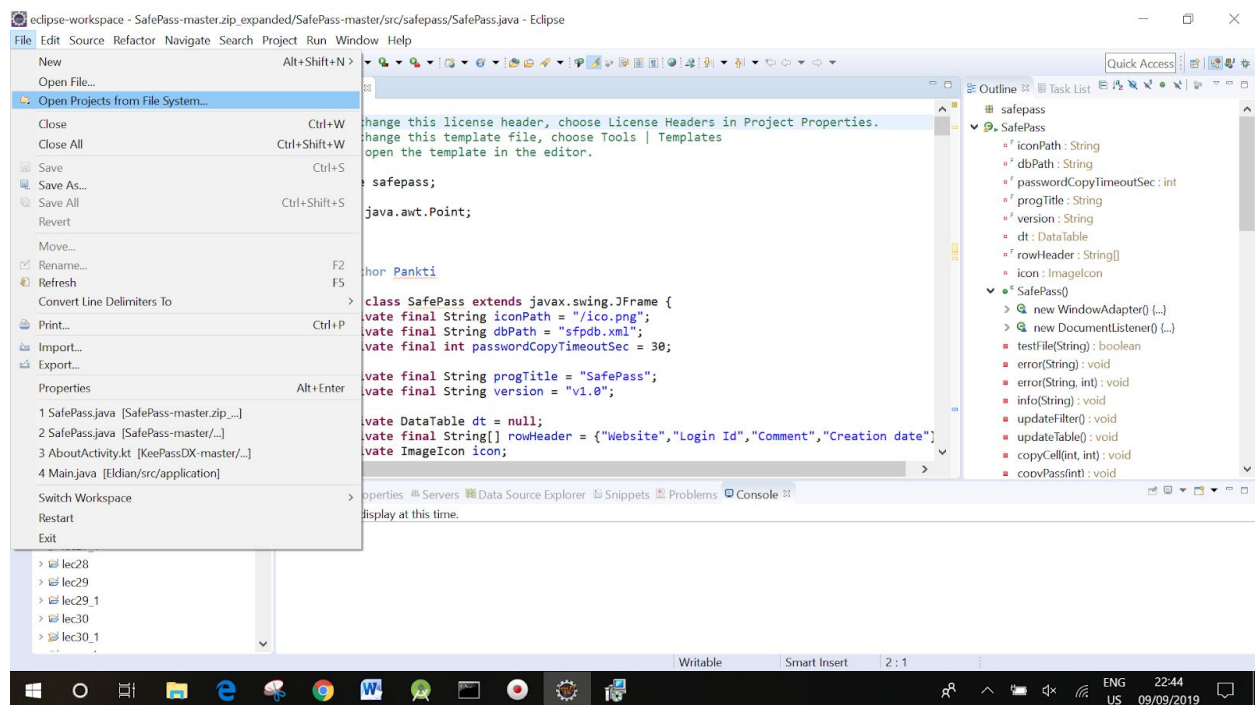
## Flow Diagram of Overall Working

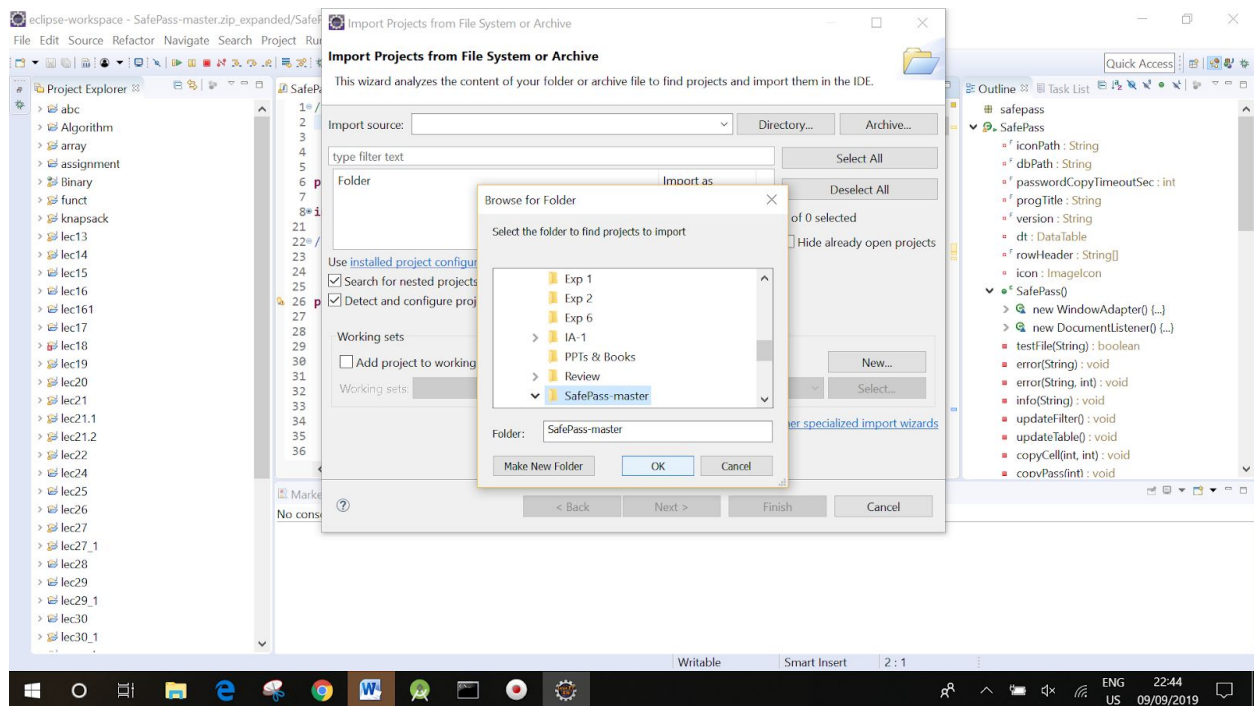
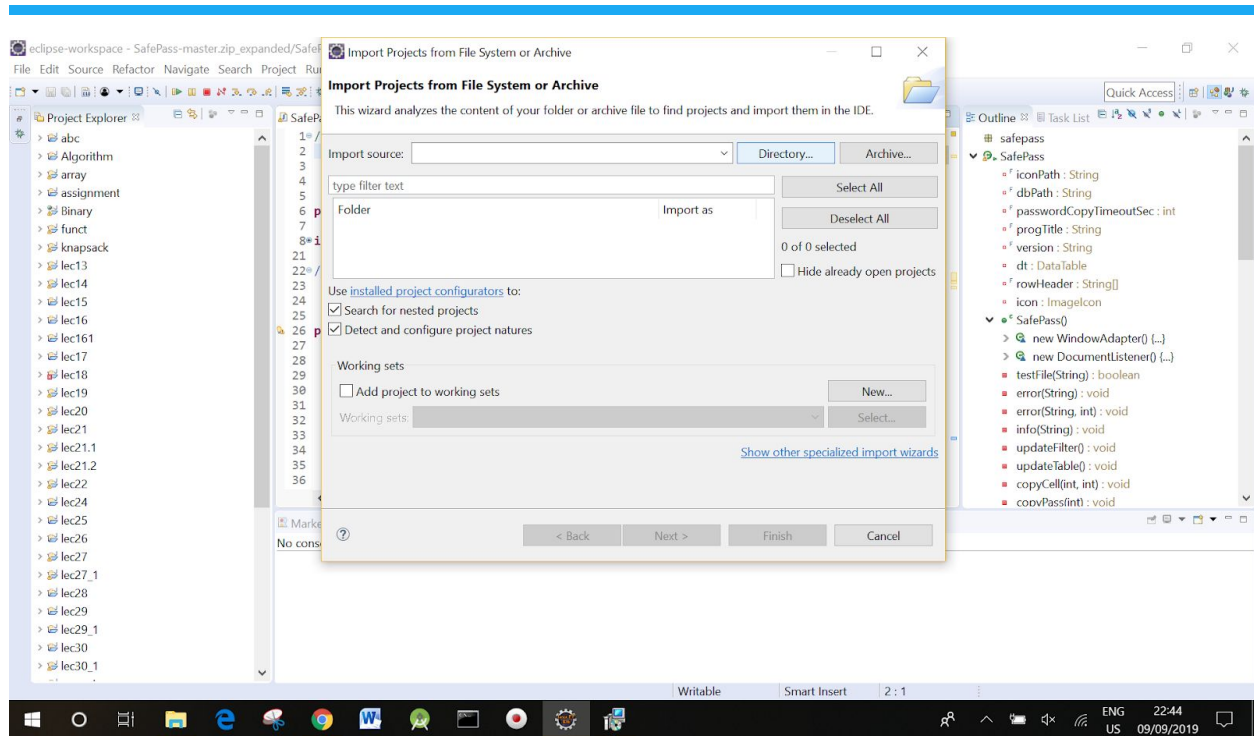


# Setup

## Usage

- Go to the [project link](#), download SafePass.jar from /dist and run it with your JVM (for users having Java 9 and above)
- Otherwise, download the project zip file from [here](#), extract it to a folder and execute the project using eclipse as follows:
  1. Download Eclipse:  
Visit: <https://www.eclipse.org/downloads/>
  2. Open the project in eclipse





### 3. Run the SafePass.java file

Note : On the first run, the program creates database file sfpdb.xml in the current directory.


## Hints

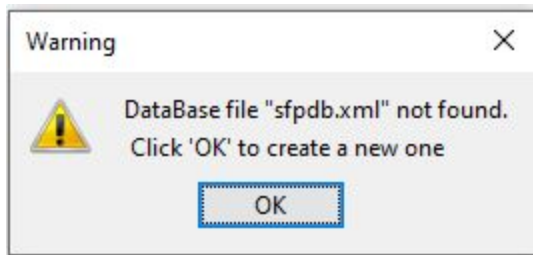
- Double click puts the password into the clipboard
- The clipboard is automatically cleared in 30 seconds after copying the password or when the program is being closed
- Search acts on "Website", "Login Id" and "Comment" fields
- "Website", "Login Id" and "Comment" fields are being encrypted too



## Screenshots of Execution

New User Login

On the first run, the XML file is created

Name	Date modified	Type	Size
 SafePass	08-09-2019 17:19	Executable Jar File	70 KB

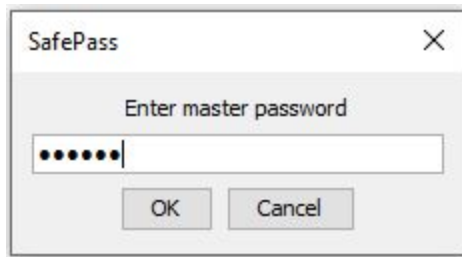


Name	Date modified	Type	Size
 SafePass	08-09-2019 17:19	Executable Jar File	70 KB
 sfpdb	09-09-2019 01:06	XML Document	2 KB



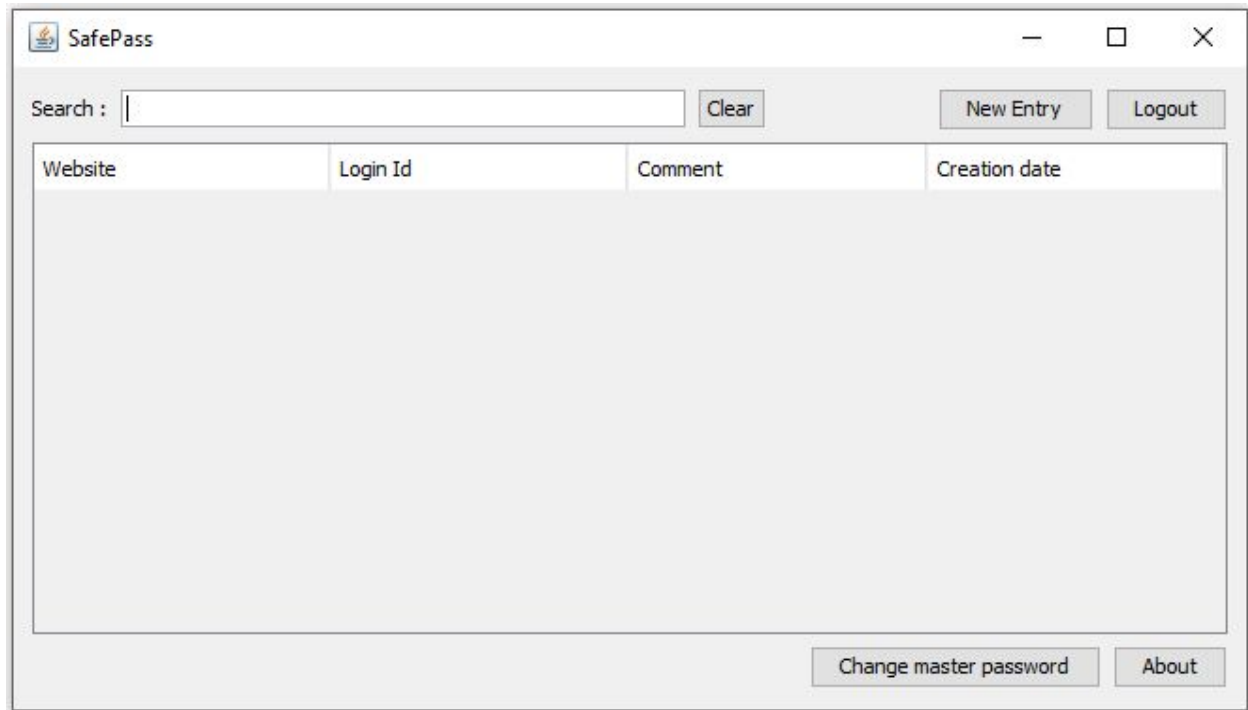
Existing User Login:

---



A small dialog box titled "SafePass" with a close button (X) in the top right corner. The text "Enter master password" is centered above a password input field. The input field contains seven black dots, indicating a masked password. Below the input field are two buttons: "OK" and "Cancel".

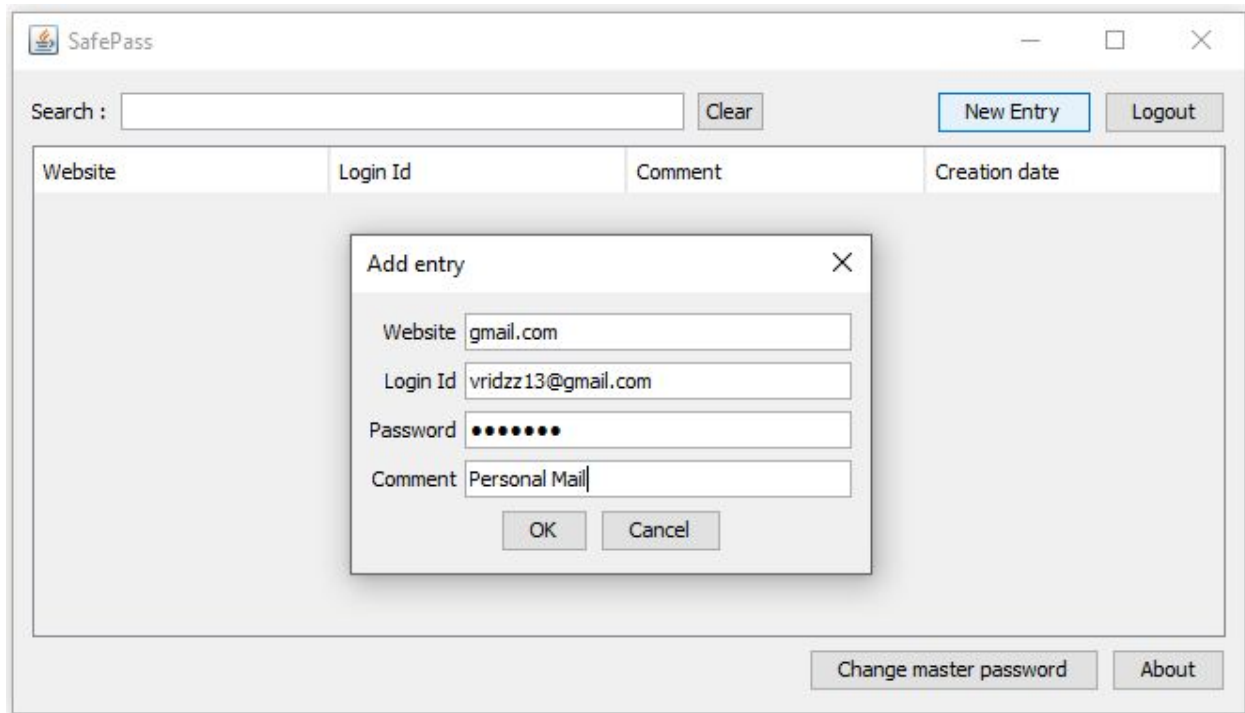
User Interface:



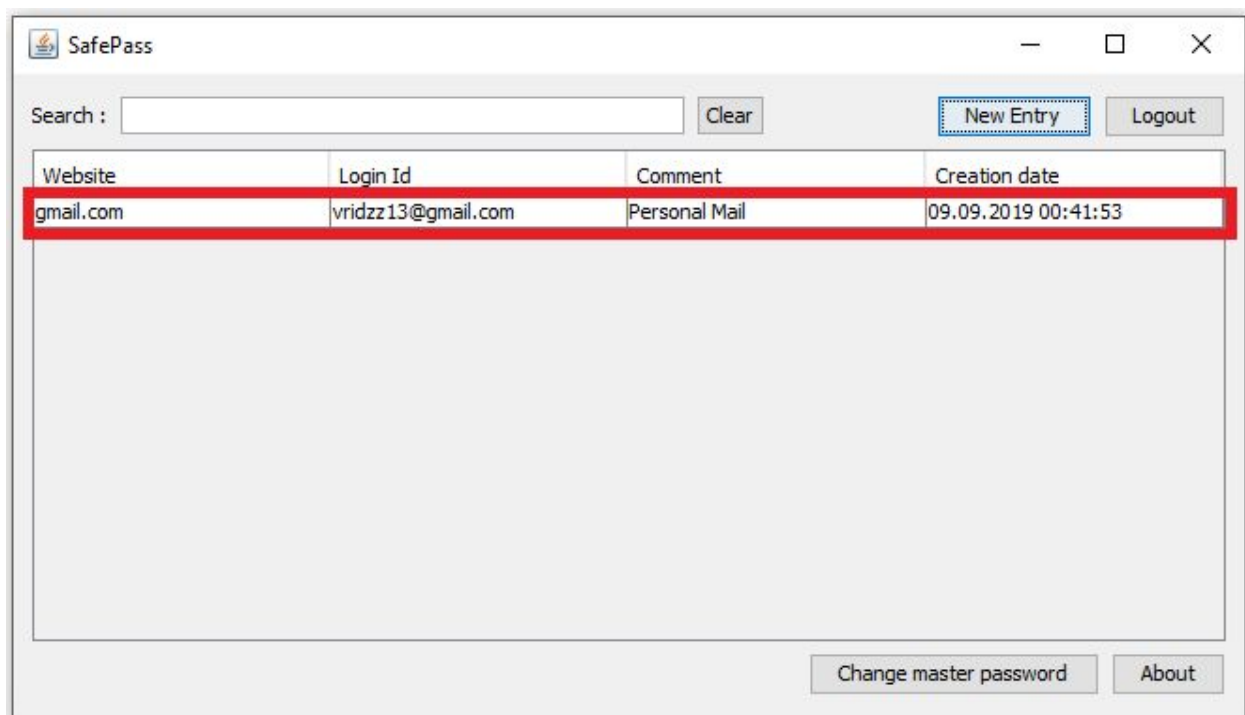
The main application window titled "SafePass" with standard window controls (minimize, maximize, close) in the top right. The interface includes a search bar with the label "Search :", a "Clear" button, and two buttons "New Entry" and "Logout". Below these is a table with four columns: "Website", "Login Id", "Comment", and "Creation date". The table body is currently empty. At the bottom right, there are two buttons: "Change master password" and "About".

Website	Login Id	Comment	Creation date
---------	----------	---------	---------------

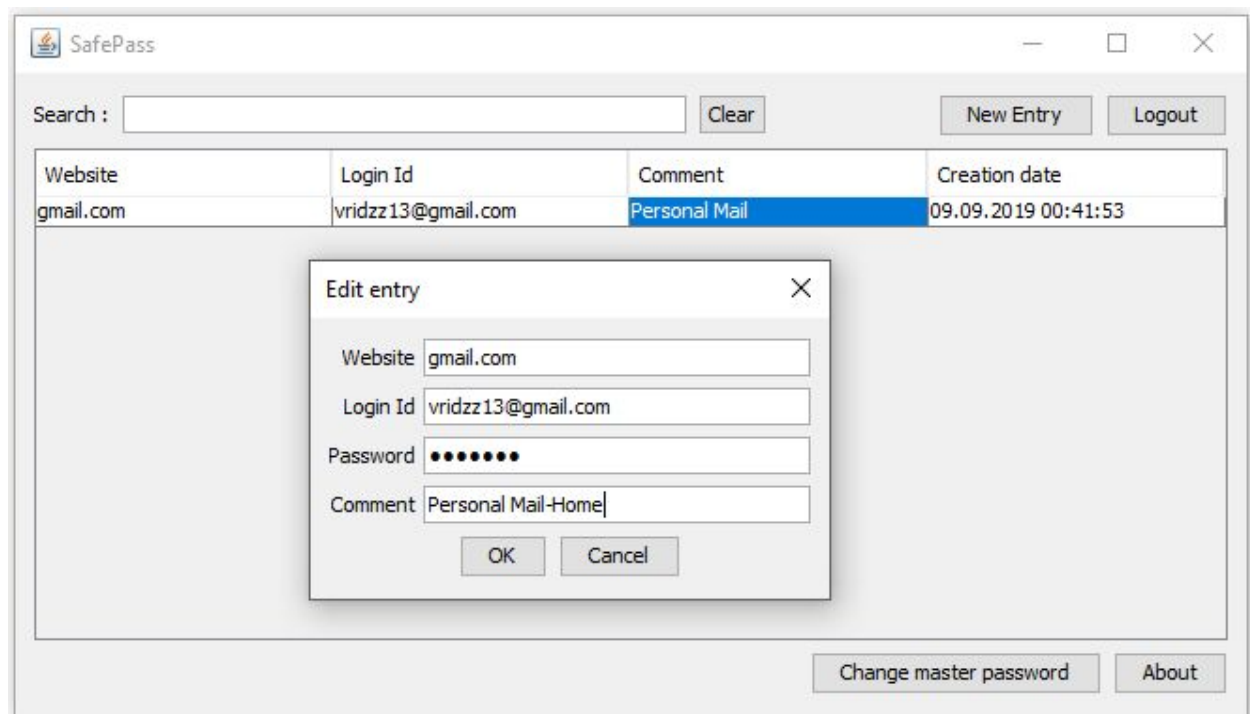
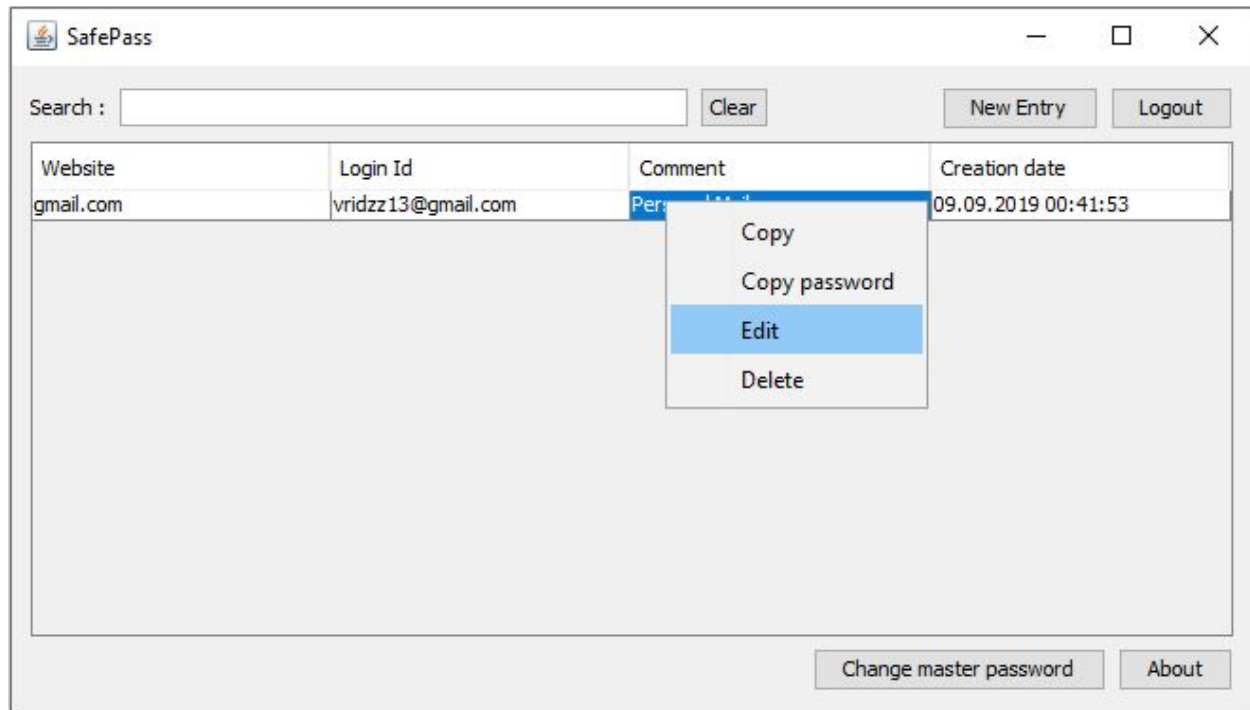
Add Entry:

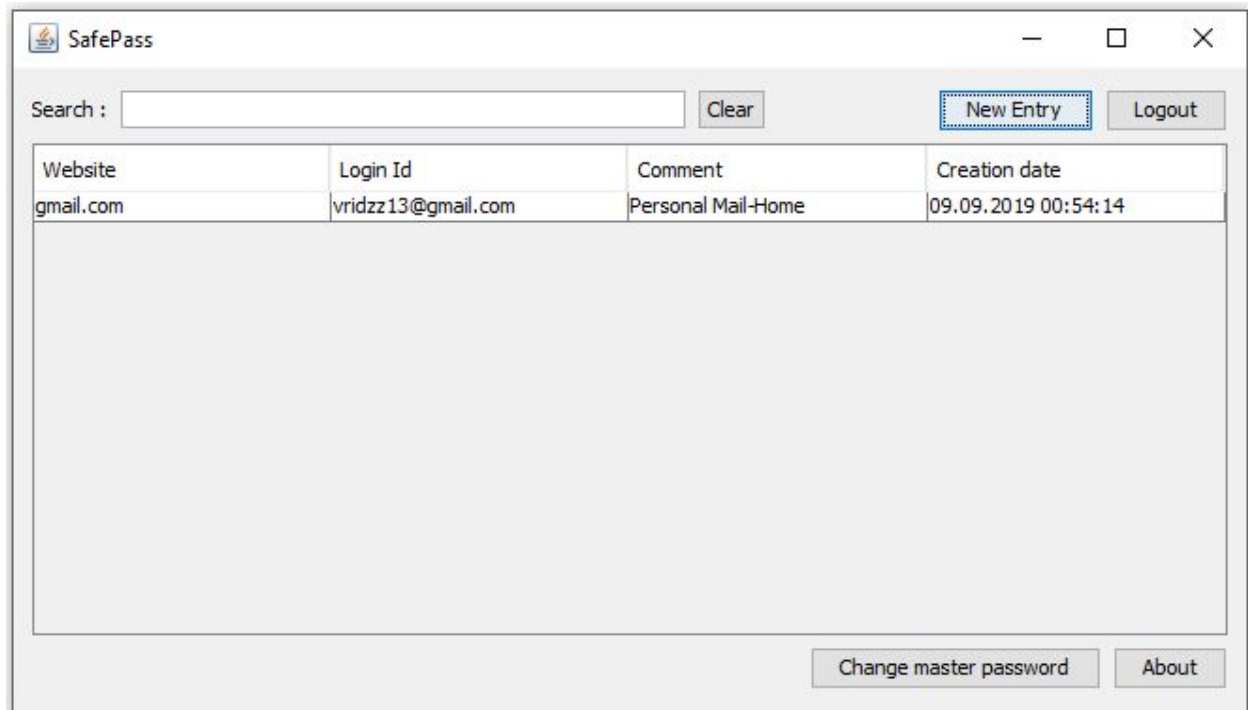


A new entry is added



Edit Entry:

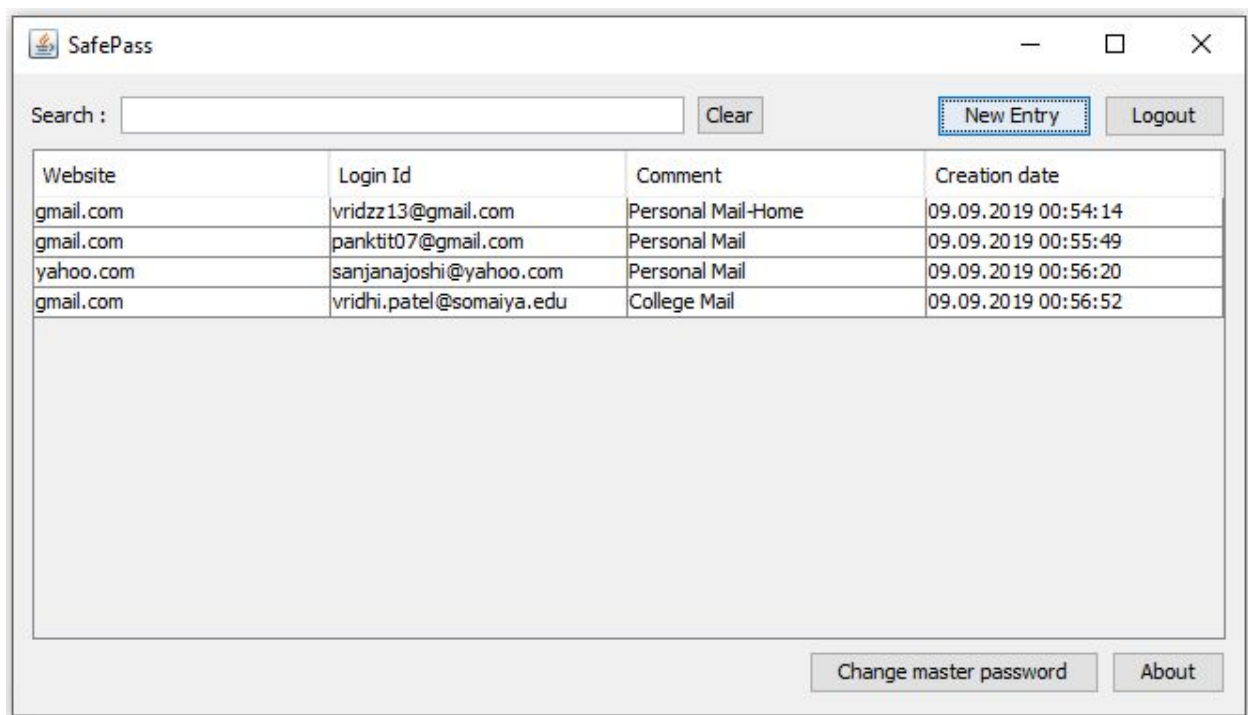




The screenshot shows the SafePass application window. At the top, there is a search bar with the text "Search : " and a "Clear" button. To the right of the search bar are two buttons: "New Entry" (highlighted with a blue border) and "Logout". Below the search bar is a table with four columns: "Website", "Login Id", "Comment", and "Creation date". The table contains one row of data: "gmail.com", "vridzz13@gmail.com", "Personal Mail-Home", and "09.09.2019 00:54:14". Below the table is a large empty rectangular area. At the bottom right of the window are two buttons: "Change master password" and "About".

Website	Login Id	Comment	Creation date
gmail.com	vridzz13@gmail.com	Personal Mail-Home	09.09.2019 00:54:14

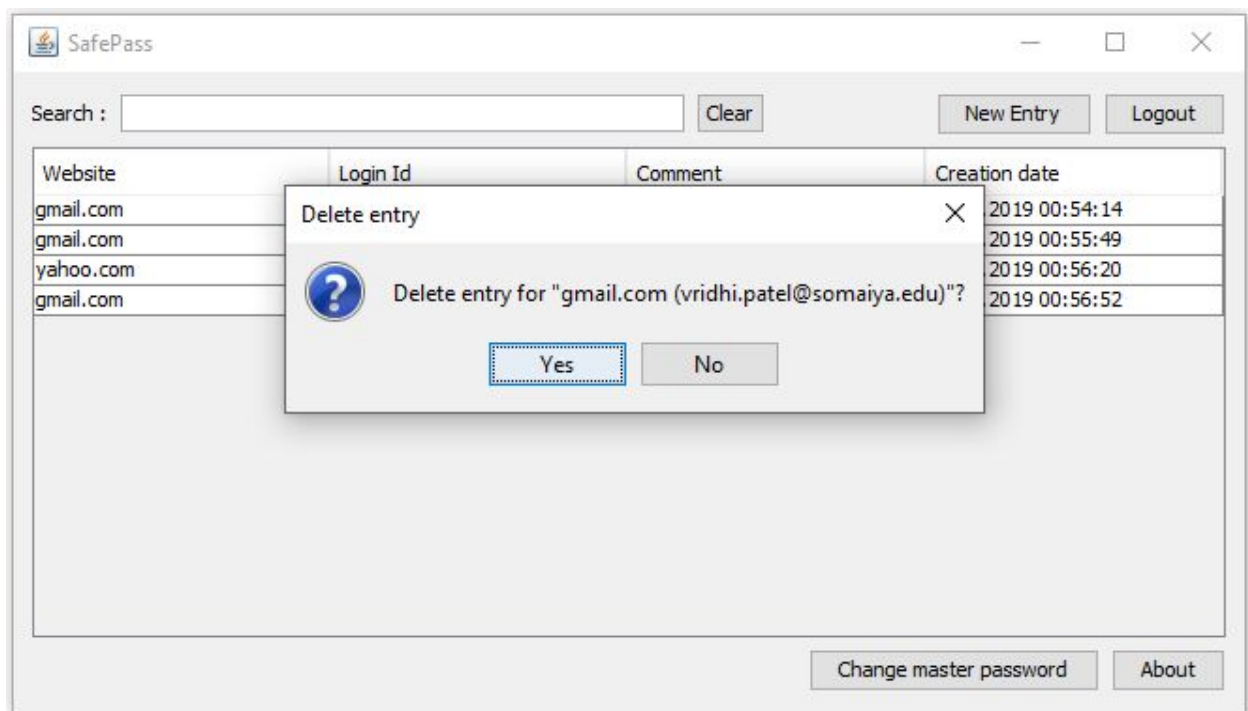
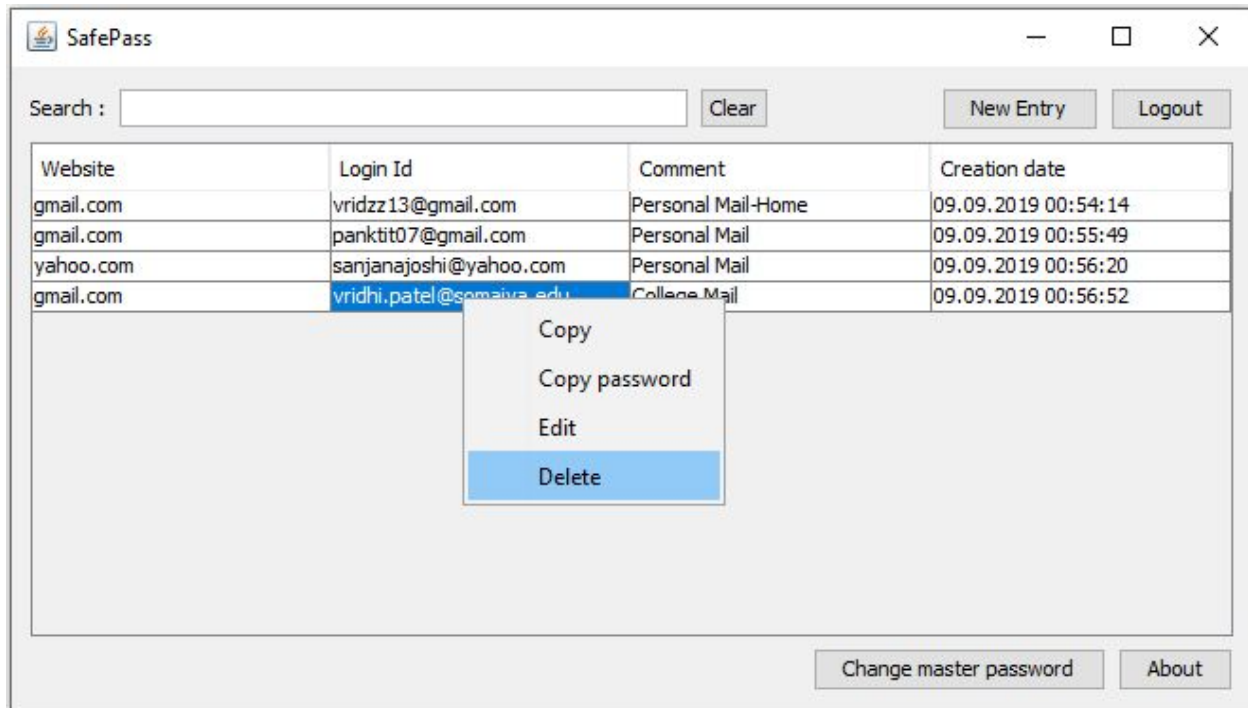
Delete Entry:



The screenshot shows the SafePass application window with the same layout as the first image. The table now contains four rows of data:

Website	Login Id	Comment	Creation date
gmail.com	vridzz13@gmail.com	Personal Mail-Home	09.09.2019 00:54:14
gmail.com	panktit07@gmail.com	Personal Mail	09.09.2019 00:55:49
yahoo.com	sanjanajoshi@yahoo.com	Personal Mail	09.09.2019 00:56:20
gmail.com	vridhi.patel@somaiya.edu	College Mail	09.09.2019 00:56:52





The screenshot shows the SafePass application window. At the top, there is a search bar with the text "Search : " and a "Clear" button. To the right of the search bar are two buttons: "New Entry" and "Logout". Below the search bar is a table with four columns: "Website", "Login Id", "Comment", and "Creation date". The table contains three rows of data. Below the table is a large empty rectangular area. At the bottom right of the window are two buttons: "Change master password" and "About".

Website	Login Id	Comment	Creation date
gmail.com	vridzz13@gmail.com	Personal Mail-Home	09.09.2019 00:54:14
gmail.com	panktit07@gmail.com	Personal Mail	09.09.2019 00:55:49
yahoo.com	sanjanajoshi@yahoo.com	Personal Mail	09.09.2019 00:56:20

Search:

This screenshot is similar to the first one, but the search bar now contains the text "gmail.com". The table below the search bar now only displays two rows, corresponding to the entries for "gmail.com". The rest of the interface, including the "Clear", "New Entry", "Logout", "Change master password", and "About" buttons, remains the same.

Website	Login Id	Comment	Creation date
gmail.com	vridzz13@gmail.com	Personal Mail-Home	09.09.2019 00:54:14
gmail.com	panktit07@gmail.com	Personal Mail	09.09.2019 00:55:49

Change Master password:

SafePass

Search :

Website	Login Id	Comment	Creation date
gmail.com	vridzz13@gmail.com	Personal Mail-Home	09.09.2019 00:54:14
gmail.com	panktit07@gmail.com	Personal Mail	09.09.2019 00:55:49
yahoo.com			9.2019 00:56:20

Change master password

Old password

New password


Confirm new password

SafePass

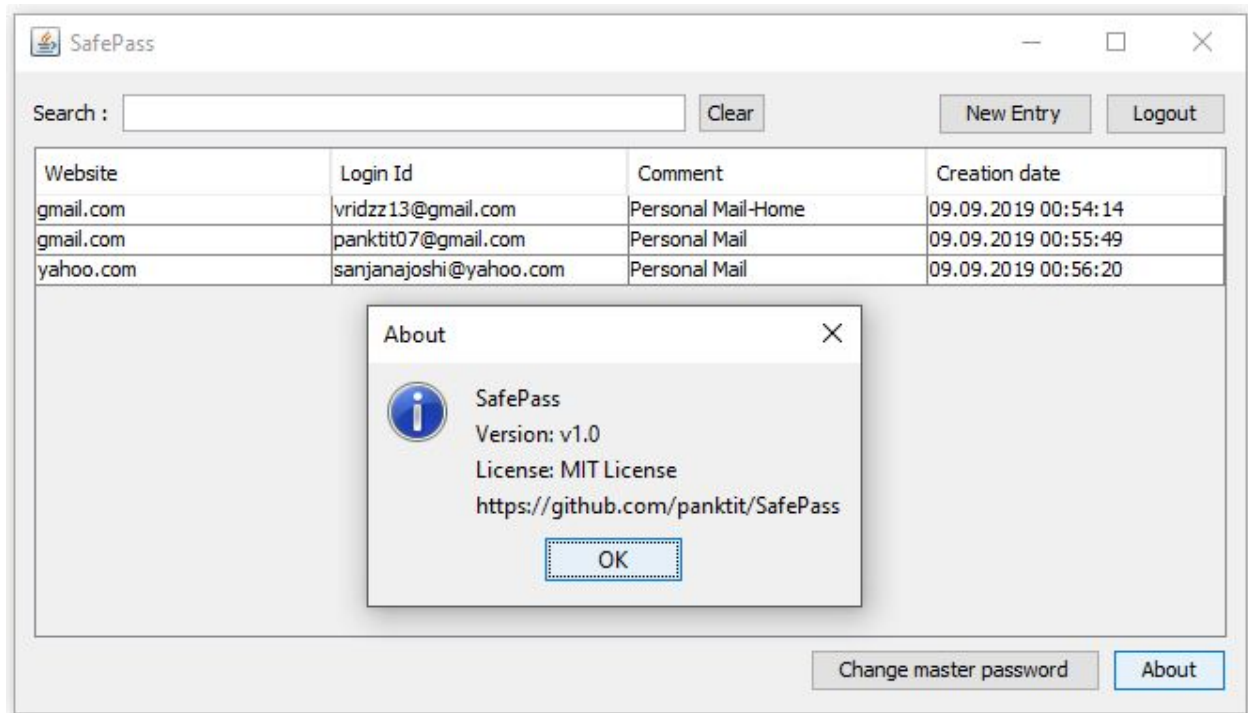
Search :

Website	Login Id	Comment	Creation date
gmail.com	vridzz13@gmail.com	Personal Mail-Home	09.09.2019 00:54:14
gmail.com	panktit07@gmail.com	Personal Mail	09.09.2019 00:55:49
yahoo.com	sanjanajoshi@yahoo.com	Personal Mail	09.09.2019 00:56:20

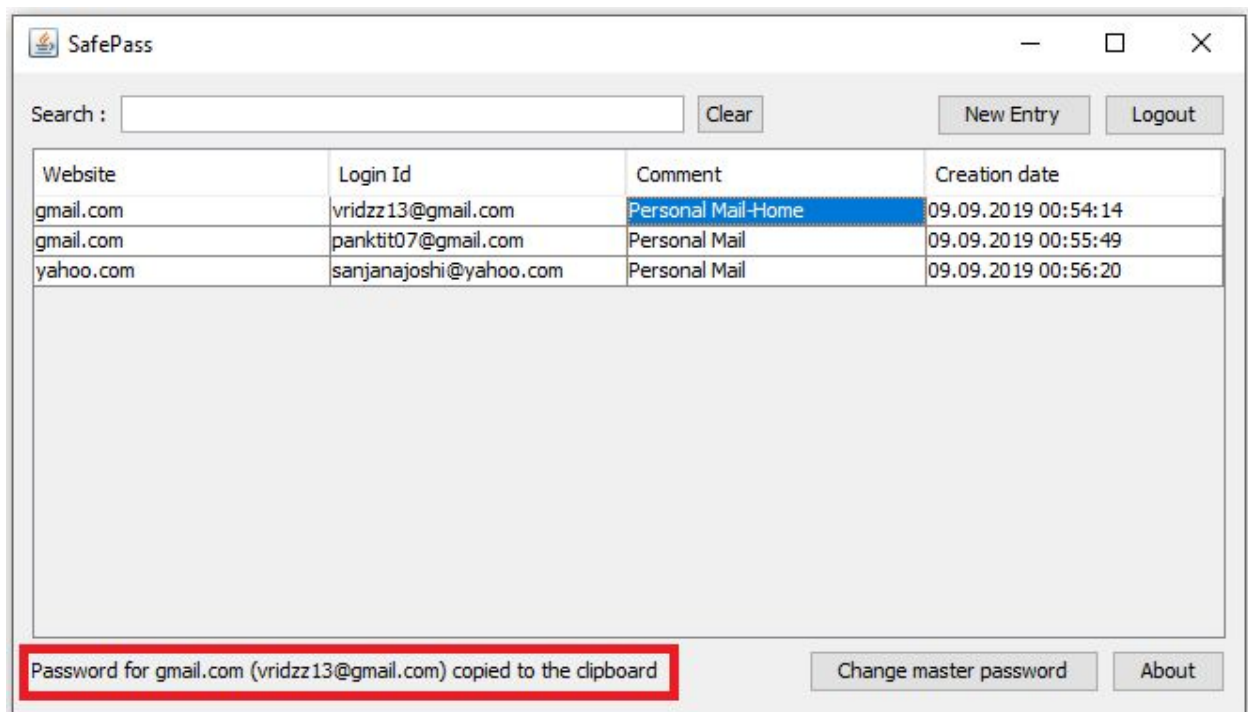
Info

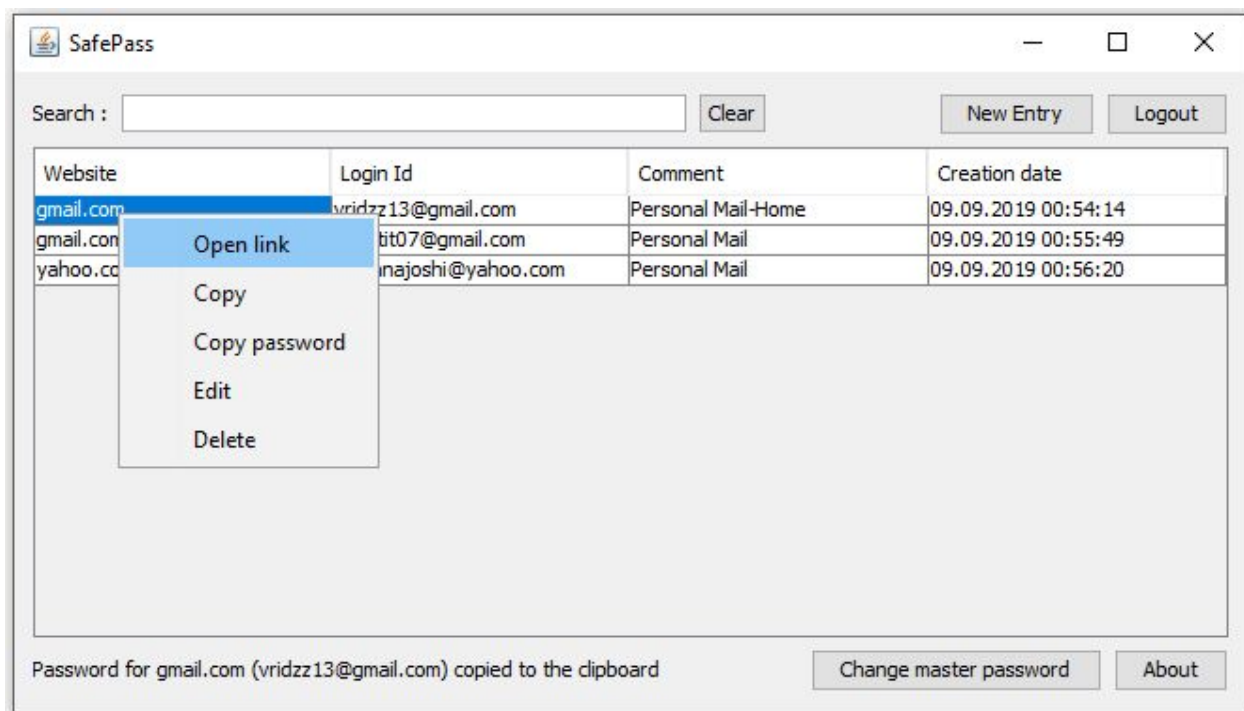
 Master password changed successfully

About:



Login in URL:





The link opens, enter your Login Id and paste the copied password to log in.

Google  
Sign in  
to continue to Gmail

Email or phone  
vridzz13@gmail.com

[Forgot email?](#)

Not your computer? Use Guest mode to sign in privately.  
[Learn more](#)

[Create account](#) [Next](#)

Google  
Welcome

vridzz13@gmail.com

Enter your password  
.....

[Forgot password?](#) [Next](#)