**Encrypted File Vault**

*Submitted by*

**Sabyasachi Biswas**

32301218053

**Saikat Maji**

32301218050

**Subhajit Bhuin**

32301218022

**Sudin Chakraborty**

32301218014

**Sandip Roy**

32301218048

Under the guidance of

**Dr. Ratnakirti Roy**

**Assistant Professor**

**Dr. B.C. Roy Engineering College**

**Academy of Professional Courses**

**Durgapur**

**INTRODUCTION**

**Encrypted File Vault** is a standalone multi-user system that enables users to securely store their files in a vault to prevent unauthorized access to those files. Users can log into the vault, add files, encrypt files using multiple algorithms, view files, and export files. Encrypted File Vault has a very simple and interactive GUI that helps users to encrypt files easily without thinking too much about underlying complexities. Users may signup using the register option available in the vault and each user can use the vault independent of other users.

Files may be encrypted using multiple algorithms to choose from. The user doesn’t have to think about keys for encryption as everything is handled by the vault.

**OBJECTIVE**

The objective of the project is to create a multi-user system that has a simple and interactive GUI that enables users to encrypt files using a specific algorithm that may be chosen from an available range of algorithms.

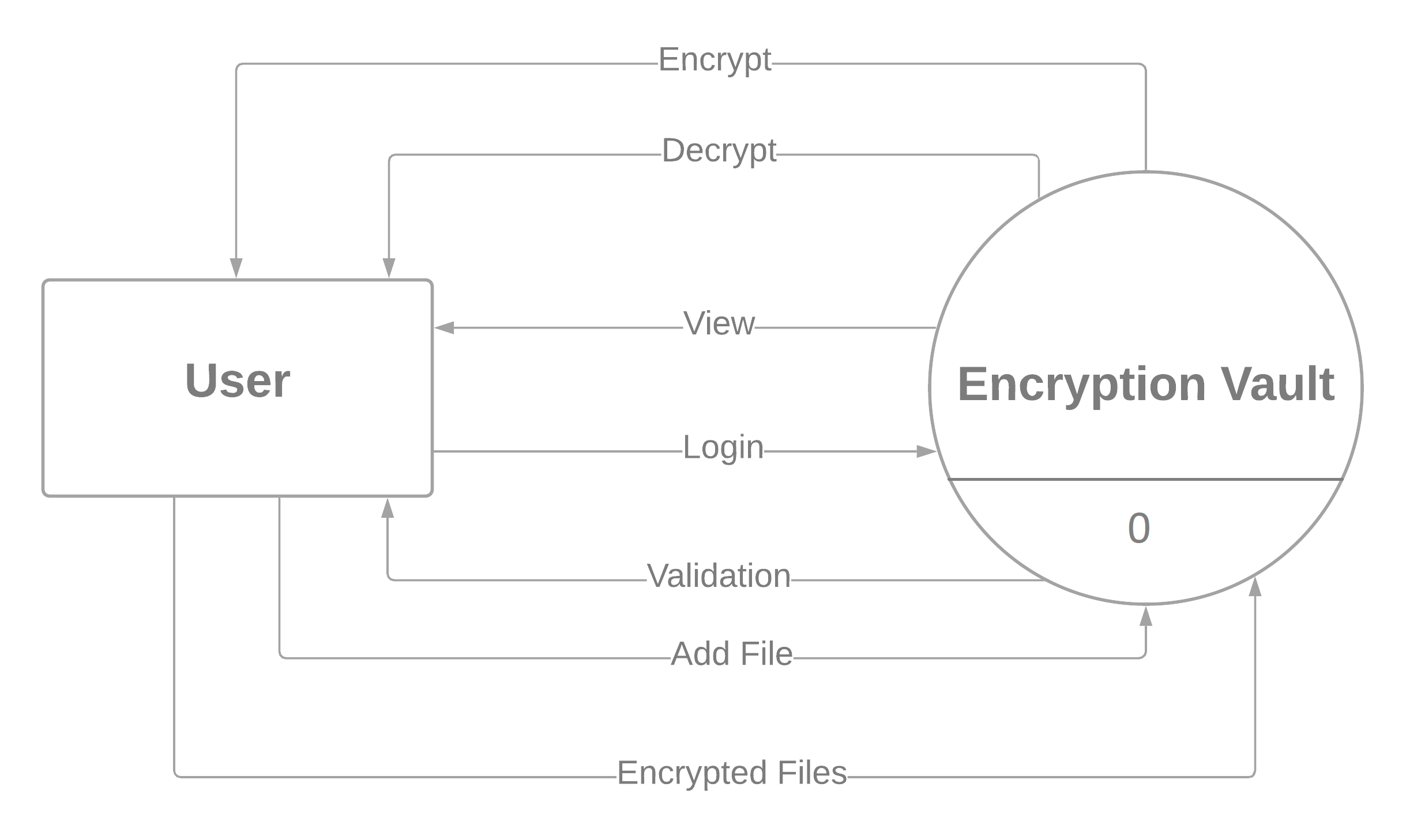
Underlying functionalities such as key generation or file operations are hidden from the end-user to make the UI very simple. Once a user registers using the register button on the login window, a new user account is created where they can use the vault. Appropriate measures are taken to ensure that when any unauthorized user does not log into the vault. This is achieved by storing users’ login information and performing validation checks during the time of login. Passwords are hashed and stored so that in case of any data breach user information is not leaked. Upon successful login, users will have the option to add files to the vault. These files can be encrypted by an algorithm chosen by the user, decrypted, viewed, and exported.

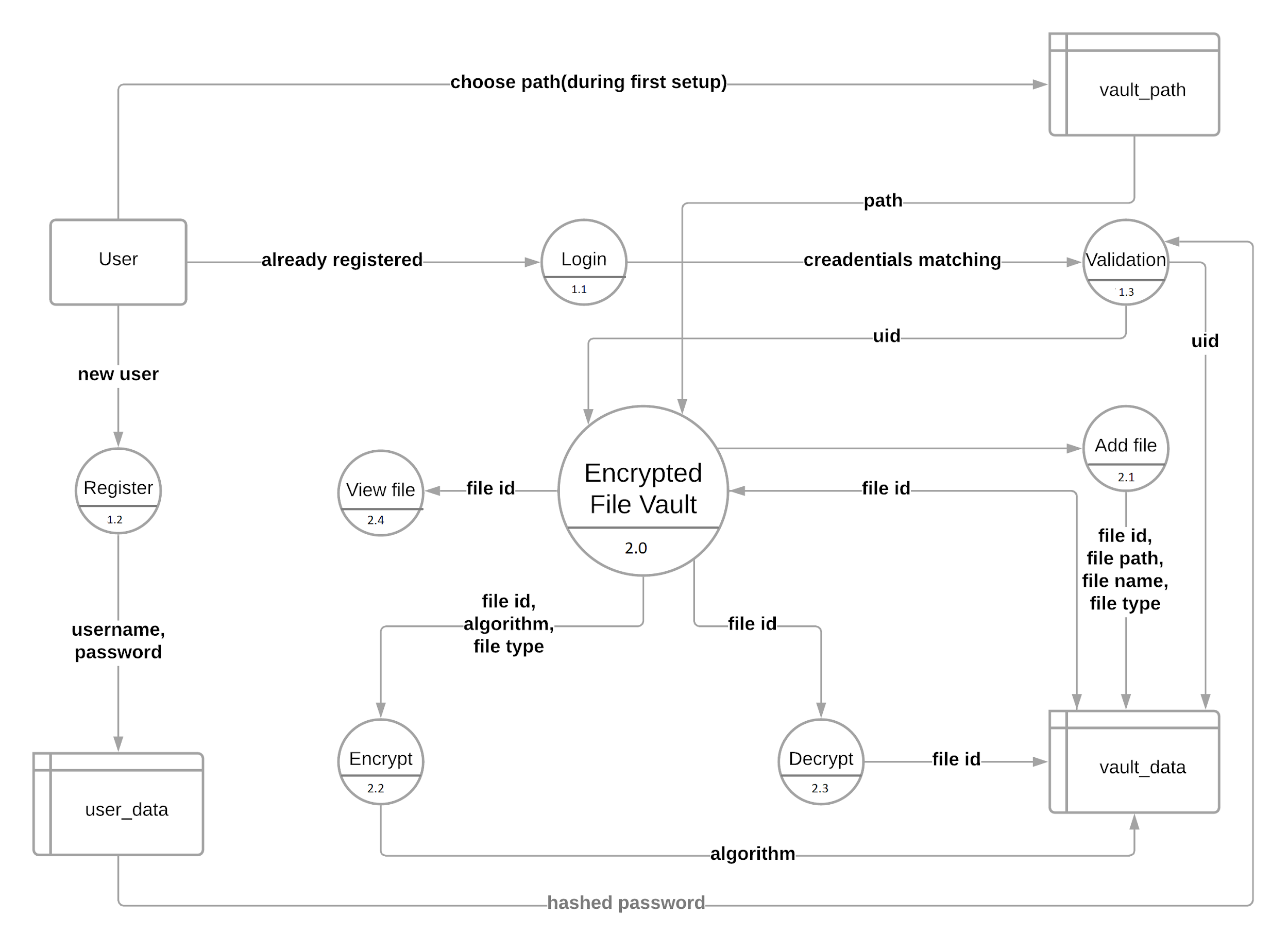
**PROJECT CATEGORY**

* **Cryptography**
* **RDBMS**
* **OOPs**

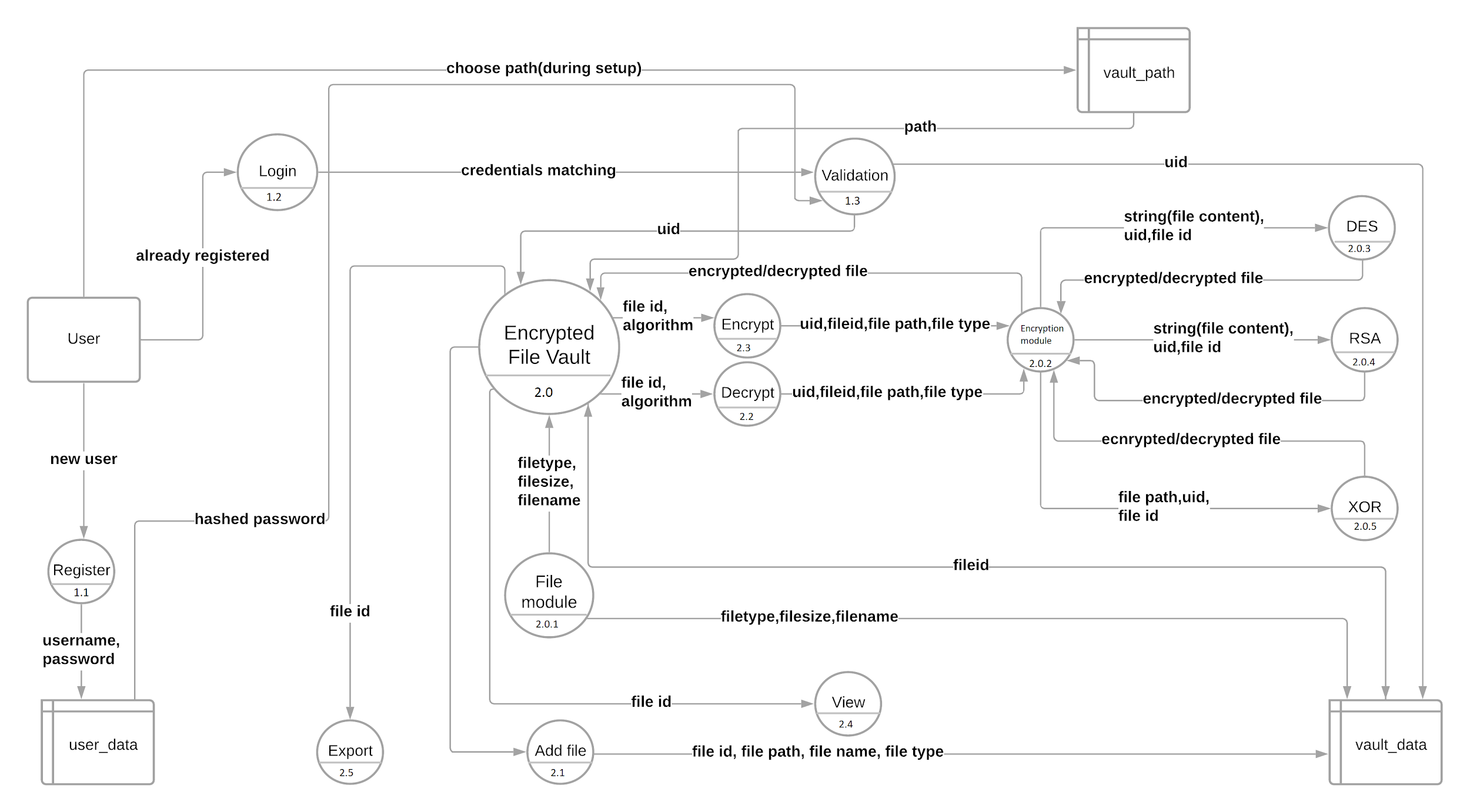
**ANALYSIS**

**Data Flow Diagram** Level 0

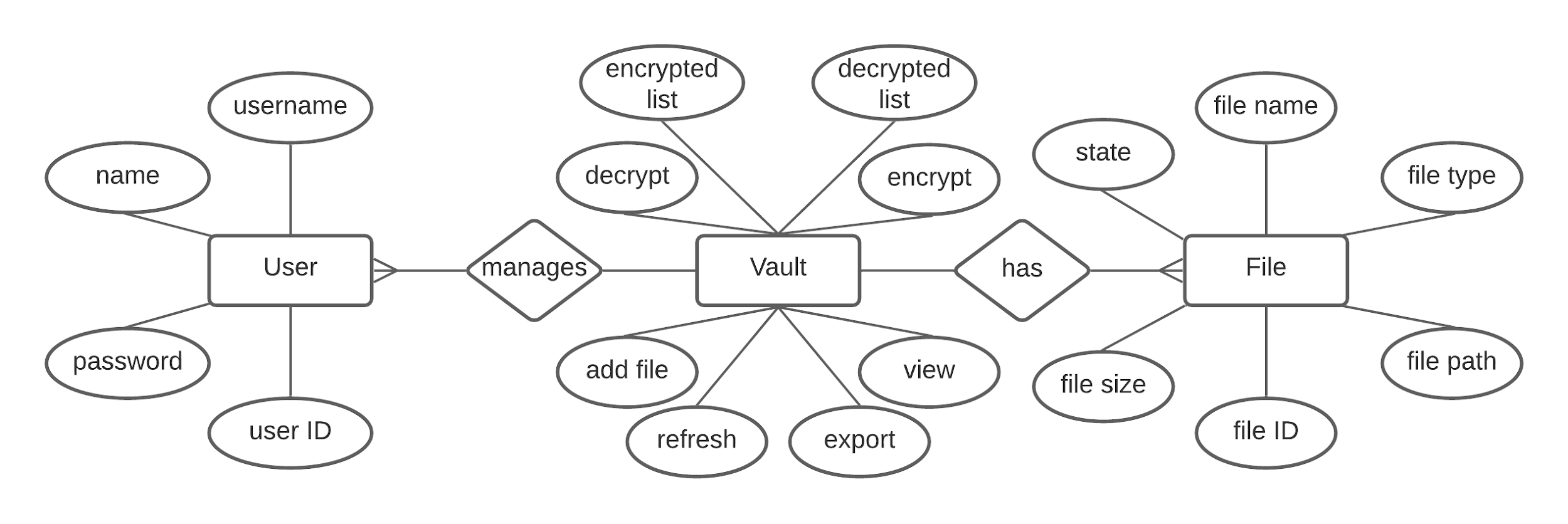
****

**Data Flow Diagram** Level 1

**Data Flow Diagram** Level 2



**Entity Relationship Diagram**



**STRUCTURE**

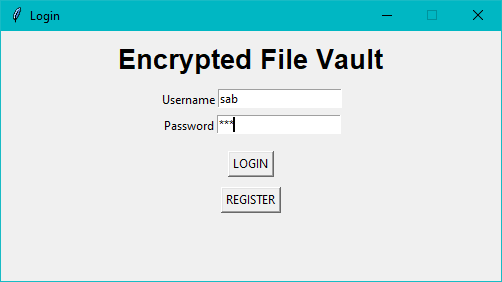
**MODULES**

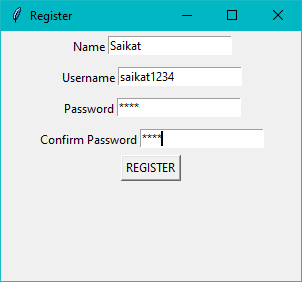
**1. login\_module**

The login module consists of the functionalities that allow users to log in to the vault. New users may register using their name and username, they are also prompted to choose a password. After successful registration, users may log in using their username and password. The hashed passwords are stored so that in case of any data breach user information is not leaked, which also makes the vault more secure. A validation check is performed to check if a user has entered the correct username and password and to keep away unauthorized users. Upon a successful validation check, the login module passes the control to the vault window along with the user ID for further reference. The user ID is unique for each user that registers to the vault, making it a true multi-user system.

The main elements of this module include:

* **Login**: For logging in users to the vault. The vault performs a validation check to authorize users.
* **Register**: New users may register here. The vault stores the user’s name, username, and hashed form of the password chosen by the user.





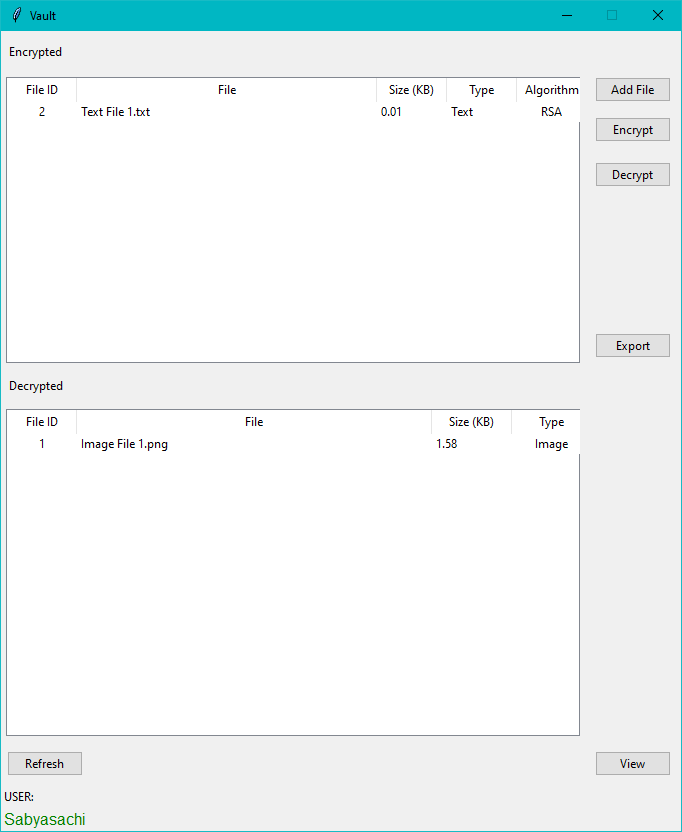
**2. hashing\_module**

The hashing module works in conjunction with the login module to hash passwords and store them in the database. Any system that requires a password needs a way to store the password in such a way that any unauthorized person does not get access to that password from the database. Hence storing the passwords as plain text is a huge security threat to both the user and the system. Hence the passwords are stored in hashed form so that even if someone does get access to the database they will not be able to understand the password as it is in a hashed form. To achieve this a third-party python library called bcrypt is used to hash the passwords during the time of user registration. When the user logs in to the vault, at the time of validation check the password entered by the user is hashed and matched against the one stored in the database. The concept of salting is also used so that passwords cannot be guessed or retrieved using rainbow tables or brute force attacks making the vault evermore secure.

**3. vault\_layout**

The vault layout is the core element of Encrypted File Vault. This module contains the GUI elements that the end-user interacts with during the time of use. Upon successful validation, the login module passes the user ID to this module. Several elements are present which include :

* **Add File**: This button allows the user to add new files to the vault. The users may add text files and images.
* **Encrypt**: This button opens up a small window that allows users to choose from three encryption algorithms available to choose from XOR, RSA, or DES, to encrypt their files.
* **Decrypt**: This button is used to decrypt the encrypted files should the user choose so.
* **View**: Using this button users may view their files when they are in a decrypted state. Concepts of image processing and file handling are used to enable this feature.
* **Export**: Should the users feel the need to take some files out of the vault, this button enables them to do so by using operating system commands.
* **Refresh**: Whenever new files are added or operations are performed, refreshing the vault UI is needed. The vault auto-refreshes to make the user experience much smoother, but the user can refresh manually, giving them much more control of the vault.
* **Encrypted treeview**: This a treeview that provides a visual representation of the files in the vault that has been encrypted. It shows the following attributes of the files: File ID, Filename, Filesize in KB, Filetype namely Image or Text, and Algorithm used to encrypt the file.
* **Decrypted treeview**: Also a treeview that provides a visual representation of the files in the vault that are in a decrypted state. When files are added to the vault they show in this treeview. It shows the following attributes of the files: File ID, Filename, Filesize in KB, and Filetype namely Image or Text.
* **Active User**: This shows the currently active user.



**4. filetype\_module**

This module performs several operations related to the files in the vault, like checking the file type namely Images or Text, getting the size of the file, and getting the filename. Whenever a user adds a new file is added to the vault the file is inspected with this module to fetch crucial information related to it so that the vault can perform its functions properly.

**5. encryption\_module**

When the user encrypts or decrypts a file the vault passes the control over to this module. The encryption module takes the file data, user ID, file ID, and algorithm and performs necessary actions before passing control over to the respective encryption algorithm modules for the files to be encrypted or decrypted. These actions include reading the file, writing to a file, or encoding the text so that the encryption modules can understand the data provided to them.

**6. rsa\_text**

If the user chooses to encrypt their file using the RSA algorithm this module comes into play. The encryption module reads the file and passes a string containing the contents of the file. The rsa module then encrypts the file using a key that is generated by the system using user data. Note that the user does not have to think about the key generation making the user experience hassle-free. Once encrypted the string is returned to the encryption module which writes the data back to the file. When the user wishes to decrypt the file the encryption module passes the key and a string containing the file contents. The rsa module decrypts the string using the key provided.

**7. des\_text**

If the user chooses to encrypt their file using the DES algorithm this module is called upon. The encryption module reads the file and passes a string containing the contents of the file. The des module then encrypts the file using a key that is generated by the system using user data. Note that the user does not have to think about the key generation making the user experience hassle-free. Once encrypted the string is returned to the encryption module which writes the data back to the file. When the user wishes to decrypt the file the encryption module passes the key and a string containing the file contents. The des module decrypts the string using the key provided.

**8. xor\_module**

If the user chooses to encrypt the file using XOR encryption, the encryption module provides the xor module with the file path and a key that is generated using user data. This is a simple encryption done by performing xor operation on the byte data of the file using the key. During decryption, the filetype module provides the file path and the key and the xor module decrypts the file using the key.

**9. setup**

When the Encrypted File Vault is installed for the first time, necessary files need to be created like the database that stores all of the information and third-party libraries for the proper functioning of the vault. The user is also prompted to choose a directory for the vault where all files will be stored. Setup needs to be run only once before the first launch of Encrypted File Vault.

**DATA STRUCTURES**

**1. login\_module**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| username | string | Accept the username provided by the user for validation check. |
| uid | integer | A unique number assigned to each user for identification and reference |
| password | string | Accepts a password from the user and either store it in hashed form or perform a validation check of the hashed password |
| name | string | Stores the user’s name during new user registration |

**2. hashing\_module**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| hash | string | Hash of the password chosen by the user during the time of registration |

**3. vault\_layout**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| fileid | integer | A number assigned to each file to uniquely identify it and refer to later |
| path | string | Stores the path for the files that are in the vault |
| algorithm | string | The algorithm chosen to encrypt is stored |

**4. filetype\_module**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| filetype | string | Identifies the type of file namely Image or Text |
| filesize | string | Stores the size of the file |
| filename | string | Splits filename from file path for storage |

**5. encryption\_module**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| fileid | integer | The number used to identify the file to be encrypted or decrypted |
| uid | integer | User ID of the current user used for key generation and reference |
| path | string | Path of the file to be encrypted or decrypted |
| algorithm | string | Algorithm for encryption or decryption |

**6. rsa\_text**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| key\_e | integer | The public key used for encryption |
| key\_d | integer | The private key used for decryption |
| value\_n | integer | Value of n for encryption and decryption |
| cypher\_list | string list | List for storing the ciphertext |
| message\_list | string list | List for storing the plaintext |
| ASCII\_values | string list | Stores the ASCII values of file contents |

**7. des\_text**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| key | string | Stores the key used for either encryption or decryption |
| plaintext | string | Plaintext before encryption or after decryption |
| ciphertext | string | Ciphertext after encryption |

**8. xor\_module**

|  |  |  |
| --- | --- | --- |
| **Variable name** | **datatype** | **Description** |
| file | string | Path to the file to be encrypted or decrypted |
| key | integer | Key used for encryption or decryption |
| file\_array | byte array | Byte array that stores contents of the file in bytes |

**PROCESS LOGIC**

**1. login\_module**

When new users register their name, username, and password is stored in the vault’s database. The passwords are hashed before storing to prevent user data leakage. The vault auto generates a user id for each user which is used for identifying users when they log in. At the time of login, the user needs to enter their username and password. The vault then matches the username and password with the database. If a match is found the user is allowed to enter the vault otherwise the user is asked to enter a correct username and password.

**2. hashing\_module**

The password that a new user chooses is hashed and stored in the database. During login, the user’s entered password is hashed and matched with the hash stored in the database. If the match is found it returns true.

**3. vault\_layout**

The vault provides a GUI that the user directly interacts with. It has certain GUI elements to facilitate the same. Two treeviews show encrypted and decrypted files and information related to those files that are present in the vault which is fetched from the database using a connection and running queries. Buttons are available to allow users to interact with the vault. The Add File button opens up a file dialog box asking the user to choose a file that is to be added to the vault. When a new file is added to the vault, by default it shows up in the decrypted treeview. Using the Encrypt button the user may encrypt the files that have been added. Pressing the encrypt button opens up a new window asking the user to choose the encryption algorithm. The decryption button decrypts the file using the algorithm that was used to encrypt it. View button uses file reading and image processing to view the files. The export button moves the file from the vault folder to another directory that is chosen by the user.

**4. filetype\_module**

Using several operating system functions file information is fetched like filetype namely image or text, file name.

**5. encryption\_module**

When the user wants to encrypt or decrypt their files, the file’s path, file id, user id, and file type is fetched from the database and encrypted or decrypted. Files are opened and the contents are passed to the respective algorithm modules for encryption along with user id and file id for key generation. Similarly during decryption, the file content along with file id and user id is passed to the respective module for decryption.

**6. rsa\_text**

Two functions are available to encrypt and decrypt. The encrypt function accepts a string and generates a key to encrypt the string, then returns the string and the decrypt function accepts a string, gets a key used for the encryption, and returns the string after decryption.

**7. des\_text**

Encrypt and decrypt functions are available. The encrypt function accepts a string and generates a key to encrypt the string, then returns the string and the decrypt function accepts a string, gets a key used for the encryption, and returns the string after decryption.

**8. xor\_module**

Here the encrypt function opens a file in byte mode and performs XOR operation of the byte data using a key and writes back the byte data to the file, whereas the decrypt function performs XOR operation using the key which was used for decryption to perform decryption similarly.

**9. setup**

Before the first run, the setup file is executed. Necessary libraries are installed and the database is created along with its tables.

**TESTING**

Testing is the process of running a system to find errors. Testing enhances the integrity of a system by detecting deviations in design and errors in the system. Testing aims at detecting error-prone areas. This helps in the prevention of errors in a system. Testing also adds value to the product by conforming to the user requirements. The main purpose of testing is to detect errors and error-prone areas in a system. Testing must be thorough and well-planned. A partially tested system is as bad as an untested system. And the price of an untested and under-tested system is high.

The implementation is the final and important phase. It involves user-training, system testing to ensure the successful running of the proposed system. The user tests the system and changes are made according to their needs. The testing involves the testing of the developed system using various kinds of data. While testing, errors are noted and correctness is the mode.

OBJECTIVES OF TESTING:

The objectives of testing are:

●Testing is the process of executing a program with the intent of finding errors.

●A Successful test case uncovers an as-yet-undiscovered error.

System testing is a stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as per the user’s need before the live operation commences. As stated before, testing is vital to the success of a system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. A series of tests are performed before the system is ready for the user acceptance test.

TESTING METHODS

System testing is the stage of implementation. This is to check whether the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. The candidate system is subject to a variety of tests: online response, volume, stress, recovery, security, and usability tests. A series of tests are performed for the proposed system is ready for user acceptance testing.

The Testing Steps are:

●Unit Testing

Unit testing focuses efforts on the smallest unit of software design. This is known as module testing. The modules are tested separately. The test is carried out during the programming stage itself. In this step, each module is found to be working satisfactorily as regards the expected output from the module.

●Integration Testing

Data can be lost across an interface. One module may harm another, sub-functions, when combined, may not be linked in the desired manner in major functions. Integration testing is a systematic approach for constructing the program structure, while at the same time conducting the test to uncover errors associated with the interface. The objective is to take unit-tested modules and build a program structure. All the modules are combined and tested as a whole.

●Validation

At the culmination of the integration testing, the Software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software tests begin in validation testing. Validation testing can be defined in many ways, but a simple definition is that the validation succeeds when The software function is in a manner that is expected by the customer. After a validation test has been conducted, one of the three possible conditions exists.

a ) The function or performance characteristics conform to specification and are accepted.

b ) A deviation from the specification is uncovered and a deficiency list is created.

c ) Proposed system under consideration has been tested by using a validation test and found to be working satisfactorily.

●Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce their quired output in a specific format. The output format on the screen is found to be correct. The format was designed in the system design time according to the user needs. For the hard copy also; the output comes as per the specified requirements by the user. Hence output testing did not result in any correction for the system.

●User Acceptance Testing

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes whenever required. This is done regarding the following point:

a) Input Screen Design

b) Output Screen Design

c) Format of reports and other outputs.

Security mechanisms

The system is provided with authentication without which no user can pass. So only legitimate users are allowed to use the application. If the legitimate users share the authentication information then the system is open to outsiders.

**DATABASE STRUCTURE**

Only one database is present user\_data containing the following tables:

1.user

|  |  |  |
| --- | --- | --- |
| **Field** | **Datatype** | **Description** |
| uid | integer | Unique ID for each user |
| name | string | Name of user |
| username | string | Username of the user |
| pwd | string | The hashed password of the user |

2.vault\_path

|  |  |  |
| --- | --- | --- |
| **Field** | **Datatype** | **Description** |
| path | string | Stores a static path to where files are stored in the vault |

3.vault\_data

|  |  |  |
| --- | --- | --- |
| **Field** | **Datatype** | **Description** |
| fileid | integer | The unique number assigned to each file for identification |
| uid | integer | Current user’s user ID |
| state | string | The current state of a file namely encrypted or decrypted |
| algo | string | Encryption algorithm used to encrypt a file |
| filename | string | Name of the file |
| filesize | integer | Size of file in bytes |
| path | string | Path to the file |
| filetype | string | Type of file namely Image or Text |

**REQUIREMENTS**

* Development tools
  + Python 3 (Development programming language)
  + VS Code (Code Editor)
  + SQLite (Relational Database)
  + Tkinter (Python GUI library)
  + Third-party libraries
    - bcrypt (Hashing library)
    - Pillow (Image processing library)
* Development hardware and operating system
  + CPU : AMD Ryzen 5 / Intel core i5
  + RAM : 16 GB
  + Diskspace : 5 GB
  + OS : Windows 10
* User software / tools and operating system requirements
  + Python 3
  + Active internet connection (only for setup)
* User hardware and operating system
  + OS : Windows 10
  + CPU : AMD Ryzen 3 / Intel core i3 or higher
  + RAM : 4 GB or higher
  + Diskspace : 2 GB or higher

**FUTURE SCOPE**

Encrypted File Vault could be deployed on the cloud to enable users to access their files over the internet. Support for video and audio files could be added to enable users to have a fully secure multimedia vault.

Advanced features like file transfer could be added so that users can transfer files. Better GUI would enhance the user experience.