# SecureByte Password Manager

## Overview

The SecureByte Password Manager is a secure application designed to manage and store passwords. It uses advanced encryption techniques to ensure the safety of your data. The application is built using Streamlit for the user interface and SQLite for database management. It also includes various algorithms for password strength analysis and similarity checks.

## Features

* **Create, Load, and Delete Databases**: Manage multiple password databases with ease.
* **Add, View, Edit, and Delete Password Entries**: Perform CRUD operations on password entries.
* **Password Encryption**: Encrypt passwords using AES-CBC encryption.
* **Password Strength Analysis**: Analyze password strength using the knapsack algorithm and zxcvbn library.
* **Password Similarity Check**: Compare passwords using the Longest Common Subsequence (LCS) algorithm.

## Project Structure

SecureByte Password Manager/ ├── password\_manager.py ├── password\_databases/ │ ├── <database\_name>.db │ ├── <database\_name>.db.meta │ └── passwords.json ├── salt.key └── README.md

## Encryption

### Key Derivation

The key for encryption is derived using the PBKDF2 algorithm with a salt. The salt is either generated or loaded from the

salt.key

file.

defderive\_key(password: str, salt: bytes) -> bytes:return PBKDF2(password, salt, dkLen=32, count=1000000)

### AES-CBC Encryption

Passwords are encrypted using AES in CBC mode. The encryption process includes padding the data to match the AES block size.

defencrypt(raw: bytes, key: bytes) -> bytes: iv = os.urandom(AES.block\_size) cipher = AES.new(key, AES.MODE\_CBC, iv) padded\_data = pad(raw) return iv + cipher.encrypt(padded\_data) defdecrypt(enc: bytes, key: bytes) -> bytes: iv = enc[:AES.block\_size] cipher = AES.new(key, AES.MODE\_CBC, iv) return unpad(cipher.decrypt(enc[AES.block\_size:]))

## Algorithms

### Longest Common Subsequence (LCS)

The LCS algorithm is used to compare passwords and determine their similarity.

deflcs(X, Y): m = len(X) n = len(Y) L = [[None] \* (n + 1) for i inrange(m + 1)] for i inrange(m + 1): for j inrange(n + 1): if i == 0or j == 0: L[i][j] = 0elif X[i - 1] == Y[j - 1]: L[i][j] = L[i - 1][j - 1] + 1else: L[i][j] = max(L[i - 1][j], L[i][j - 1]) return L[m][n]

### Knapsack Algorithm

The knapsack algorithm is used to analyze password strength based on various characteristics.

defknapsack(characteristics, max\_weight): n = len(characteristics) items = list(characteristics.items()) dp = [[0for \_ inrange(max\_weight + 1)] for \_ inrange(n + 1)] for i inrange(1, n + 1): for w inrange(1, max\_weight + 1): char\_name, char\_weight = items[i - 1] if char\_weight <= w: dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - char\_weight] + char\_weight) else: dp[i][w] = dp[i - 1][w] selected\_chars = [] w = max\_weight for i inrange(n, 0, -1): if dp[i][w] != dp[i - 1][w]: selected\_chars.append(items[i - 1][0]) w -= items[i - 1][1] return selected\_chars, dp[n][max\_weight]

## User Interface

The user interface is built using Streamlit. It includes various functions to manage databases and password entries.

### Manage Databases

defmanage\_database\_ui(): st.sidebar.title("SecureByte - Manage Password Databases") databases, metadata = show\_databases() st.sidebar.header("Your Databases") for db in databases: category = metadata.get(db, {}).get('category', 'Uncategorized') st.sidebar.write(f"\*\*{db}\*\* ({category})") st.sidebar.header("Manage Databases") create\_new = st.sidebar.checkbox("Create New Database") delete\_existing = st.sidebar.checkbox("Delete a Database") if create\_new: st.subheader("Create a New Database") name = st.text\_input("Database Name") category = st.text\_input("Category (optional)") password = st.text\_input("Set Password", type="password") confirm\_password = st.text\_input("Confirm Password", type="password") if st.button("Create Database"): ifnot name ornot password: st.error("Name and password are required.") elif password != confirm\_password: st.error("Passwords do not match.") else: try: create\_database(name, category, password) st.success(f"Database '{name}' created successfully!") except Exception as e: st.error(f"Error: {e}") if delete\_existing: st.subheader("Delete a Database") db\_to\_delete = st.selectbox("Select Database to Delete", databases) password = st.text\_input("Enter Password to Confirm Deletion", type="password") if st.button("Delete Database"): try: delete\_database(db\_to\_delete, password) st.success(f"Database '{db\_to\_delete}' deleted successfully!") except Exception as e: st.error(f"Error: {e}")

### Manage Entries

defmanage\_entries\_ui(): file\_path = st.session\_state["file\_path"] key = st.session\_state["key"] conn, \_ = load\_database(file\_path, key) st.header(f"Manage Password Entries in '{st.session\_state['selected\_db']}'") option = st.radio("Choose an option", ["Add Entry", "View Entries", "Edit Entry", "Delete Entry", "Check Similarities"]) if option == "Add Entry": st.subheader("Add a New Password Entry") with st.form(key='add\_entry\_form'): name = st.text\_input("Entry Name (required)", key="add\_entry\_name") with st.expander("Add a Description (optional)"): description = st.text\_area("Description", key="add\_entry\_description") password = st.text\_input("Password (required)", type="password", key="add\_entry\_password") submit\_button = st.form\_submit\_button("Add Entry") if submit\_button: ifnot name ornot password: st.error("Name and Password are required!") else: try: encrypted\_password = base64.b64encode(encrypt(password.encode(), key)).decode('utf-8') conn.execute( "INSERT INTO passwords (name, description, encrypted\_password) VALUES (?, ?, ?)", (name, description, encrypted\_password), ) save\_database(conn) st.success(f"Entry '{name}' added successfully!") st.rerun() except Exception as e: st.error(f"Error saving entry: {e}") elif option == "View Entries": st.subheader("View All Password Entries") cursor = conn.execute("SELECT id, name, description, encrypted\_password FROM passwords") rows = cursor.fetchall() if rows: for row in rows: decrypted\_password = decrypt(base64.b64decode(row[3]), key).decode('utf-8') with st.expander(f"\*\*{row[1]}\*\*"): st.write(f"\*\*Description\*\*: {row[2]}") st.write(f"\*\*Password\*\*: {decrypted\_password}") if st.button("Copy Password", key=f"copy\_{row[0]}"): st.session\_state[f"copied\_password\_{row[0]}"] = decrypted\_password st.success(f"Password for '{row[1]}' copied to clipboard!") else: st.info("No entries found in the database.") elif option == "Edit Entry": st.subheader("Edit a Password Entry") cursor = conn.execute("SELECT id, name, description, encrypted\_password FROM passwords") rows = cursor.fetchall() if rows: entries = {row[1]: row for row in rows} selected\_entry = st.selectbox("Select Entry to Edit", list(entries.keys()), key="edit\_entry\_select") entry = entries[selected\_entry] with st.form(key='edit\_entry\_form'): name = st.text\_input("Entry Name (required)", value=entry[1], key="edit\_entry\_name") description = st.text\_area("Description", value=entry[2], key="edit\_entry\_description") password = st.text\_input("Password (required)", type="password", value=decrypt(base64.b64decode(entry[3]), key).decode('utf-8'), key="edit\_entry\_password") submit\_button = st.form\_submit\_button("Update Entry") if submit\_button: ifnot name ornot password: st.error("Name and Password are required!") else: try: encrypted\_password = base64.b64encode(encrypt(password.encode(), key)).decode('utf-8') conn.execute( "UPDATE passwords SET name = ?, description = ?, encrypted\_password = ? WHERE id = ?", (name, description, encrypted\_password, entry[0]), ) save\_database(conn) st.success(f"Entry '{name}' updated successfully!") st.rerun() except Exception as e: st.error(f"Error updating entry: {e}") else: st.info("No entries to edit.") elif option == "Delete Entry": st.subheader("Delete a Password Entry") cursor = conn.execute("SELECT id, name FROM passwords") rows = cursor.fetchall() if rows: entries = {row[1]: row[0] for row in rows} selected\_entry = st.selectbox("Select Entry to Delete", list(entries.keys()), key="delete\_entry\_select") if st.button("Delete Entry", key="delete\_entry\_button"): conn.execute("DELETE FROM passwords WHERE id = ?", (entries[selected\_entry],)) save\_database(conn) st.success(f"Entry '{selected\_entry}' deleted successfully!") else: st.info("No entries to delete.") elif option == "Check Similarities": check\_similarities\_ui() save\_database(conn) if st.button("Close Database", key="close\_database\_button"): st.session\_state["db\_unlocked"] = False st.session\_state.pop("file\_path", None) st.session\_state.pop("key", None) st.session\_state.pop("selected\_db", None) st.rerun()

## Running the Application

To run the application, execute the following command:

streamlit run password\_manager.py

## Dependencies

The application requires the following Python packages:

* Streamlit
* SQLite
* PyCryptodome
* Plotly
* zxcvbn

Install the dependencies using the following command:

pip install streamlit sqlite3 pycryptodome plotly zxcvbn

## Future Enhancements

* **Two-Factor Authentication**: Add an extra layer of security by implementing two-factor authentication.
* **Password Generation**: Include a feature to generate strong passwords.
* **Cloud Storage**: Allow users to store their databases in the cloud for easy access across devices.
* **User Management**: Implement user accounts and permissions for shared databases.
* **Backup and Restore**: Provide options to backup and restore databases.License

This project is licensed under the MIT License. See the

LICENSE

file for details.

Similar code found with 2 license types