Разработка менеджера паролей с генерацией на основе цепей Маркова

Андрей Сергиенко Б-82
Проект: Markov Password Manager (PwdMan)

5 декабря 2025 г.

Аннотация

Данный пет-проект был создан из-за конкретной практической потребности — необходимости централизованного и безопасного хранения множества учетных данных. Проблема заключалась в фрагментации хранения паролей: часть из них сохранялась в браузере Chrome, часть — в Firefox, отдельные учетные данные хранились в текстовых заметках или даже в избранных сообщениях мессенджеров. Такая ситуация создавала неудобства при использовании нескольких браузеров, а также при удаленном подключении к серверам по протоколу SSH.

Кроме того, существующие решения, такие как генератор паролей в Google Chrome, часто создают сложные, нечитаемые последовательности символов, которые трудно запомнить в случае необходимости ручного ввода. Поэтому мной в качестве альтернативы был применен алгоритм генерации псевдослов на основе цепей Маркова. Этот метод позволяет создавать пароли, которые, сохраняя высокую энтропию, остаются относительно запоминаемыми благодаря имитации структуры естественного языка.

Таким образом, проект сочетает в себе задачи централизованного управления учетными данными и генерации устойчивых к взлому, но при этом мнемонически удобных паролей.

1 Теоретические основы

1.1 Марковские цепи

Марковская цепь — это случайный процесс, в котором следующий шаг зависит только от текущего состояния и не зависит от того, как система пришла в это состояние.

Формально, для дискретной марковской цепи порядка n вероятность появления символа x_{i+1} задаётся так:

$$P(x_{i+1} \mid x_1, x_2, \ldots, x_i) = P(x_{i+1} \mid x_{i-n+1}, \ldots, x_i) \tag{1}$$

При генерации текста состояние — это последние n символов, а переход — выбор следующего символа на основе статистики, собранной из обучающего набора данных.

1.2 Энтропия паролей

Энтропия пароля H измеряется в битах и определяет количество попыток, необходимых для подбора пароля методом грубой силы:

$$H = \log_2(N) \tag{2}$$

где N — количество возможных комбинаций.

Для пароля длиной L символов из алфавита размером A:

$$H = L \cdot \log_2(A) \tag{3}$$

В реализованной системе используется смешанный регистр (прописные и строчные буквы), что обеспечивает оптимальную энтропию при сохранении структурной похожести на естественный язык.

2 Архитектура системы

2.1 Общая структура

Система состоит из четырех основных модулей:

1. **Генератор паролей** (`generator.py`) — реализация марковской модели
2. **Криптографическое хранилище** (`vault.py`) — управление зашифрованной базой данных
3. **Интерфейс командной строки** (`cli.py`) — пользовательский интерфейс
4. **Модуль конфигурации** (`__init__.py`) — управление путями и константами

2.2 Модуль генерации паролей

Класс `MarkovPasswordGenerator` реализует следующий алгоритм:

1. **Предобработка корпуса**: очистка текста с сохранением только букв кириллицы или латиницы
2. **Построение модели**: создание словаря переходов n-грамм (по умолчанию $n = 2$)
3. **Генерация**: итеративное построение последовательности на основе вероятностей переходов
4. **Постобработка**: случайная капитализация символов с вероятностью 0.3, я не стал добавлять спецсимволы и цифры в пароль, что бы не нарушать читаемость.

Модель сохраняется в бинарном формате (pickle) для быстрой загрузки при последующих запусках, что существенно сокращает время инициализации системы.

2.3 Криптографическое хранилище

Модуль `vault.py` обеспечивает безопасное хранение паролей с использованием следующих технологий:

- **PBKDF2-HMAC-SHA256**: функция формирования ключа на основе мастер-пароля с 480,000 итерациями
- **Fernet (AES-256-CBC)**: симметричное шифрование паролей
- **SQLite**: локальная база данных для хранения зашифрованных записей
- **Случайная соль**: уникальный 16-байтный вектор для каждой инсталляции

Для обеспечения безопасности применяется двухэтапный криптографический процесс, который преобразует простой для запоминания мастер-пароль в надежную систему защиты данных.

На первом этапе пользовательский мастер-пароль, который сам по себе может быть недостаточно сложным, преобразуется в надежный криптографический ключ с помощью функции **PBKDF2-HMAC-SHA256**. Этот процесс решает две ключевые задачи:

1. **Использование соли:** Для преобразования используется случайная 16-байтная соль, уникальная для каждой установки приложения. Соль гарантирует, что даже при одинаковом мастер-пароле у разных пользователей итоговые ключи будут различными, что защищает от атак с использованием заранее вычисленных таблиц (радужных таблиц).
2. **Замедление перебора:** Высокое количество итераций (480,000) намеренно замедляет процесс выработки ключа. Это делает атаку полного перебора (brute-force) крайне затратной по времени и вычислительным ресурсам для злоумышленника, фактически сводя ее на нет.

На втором этапе полученный ключ используется для шифрования самих паролей с помощью симметричного шифра **Fernet**, который основан на стандарте **AES-256-CBC**. Fernet обеспечивает не только конфиденциальность, но и целостность данных:

- **Конфиденциальность:** Алгоритм AES-256 делает зашифрованные данные нечитаемыми без наличия секретного ключа.
- **Целостность:** Помимо шифрования, Fernet добавляет к данным код аутентификации сообщения (HMAC). Этот механизм гарантирует, что данные не были изменены. Если злоумышленник попытается внести изменения в зашифрованную базу данных, цифровая подпись станет недействительной, и приложение откажется расшифровывать поврежденные данные. Это защищает от активных атак, направленных на подмену информации.

База данных **SQLite** выступает в роли структурированного хранилища. Она не обеспечивает шифрование сама по себе, а служит надежным и удобным контейнером для зашифрованных записей, создаваемых алгоритмом Fernet.

Таким образом, представленная криптографическая схема обеспечивает комплексную защиту: конфиденциальность данных за счет сильного шифрования, защиту от подделки, а также устойчивость к взлому благодаря медленному и "соленому"процессу формирования ключа. Компрометация файлов системы не ведет к раскрытию паролей без знания мастер-пароля.

2.4 Управление пакетом через pipx

Приложение распространяется как полноценный Python-пакет, устанавливаемый через `pipx` — инструмент для изолированной установки приложений командной строки.

После установки командой `pipx install -e .` утилита `pwdman` становится доступной глобально в системе, при этом все зависимости изолируются в отдельном виртуальном окружении, что предотвращает конфликты версий библиотек.

Система использует `platformdirs` для определения кросс-платформенных путей хранения данных:

- Linux: `~/.local/share/PasswordManager`
- macOS: `~/Library/Application Support/PasswordManager`
- Windows: `C:\Users\<User>\AppData\Local\...\PasswordManager`

3 Реализация

3.1 Интерфейс командной строки

Система предоставляет следующие команды:

```
pwdman generate -l 16 -c 5        // Генерация паролей
pwdman add facebook user@email    // Добавление пароля
pwdman get facebook -copy         // Извлечение с копированием
pwdman list                       // Список сервисов
pwdman delete facebook            // Удаление записи
pwdman train                      // Переобучение модели
pwdman reset                      // Очистка хранилища
```

Интерфейс реализован с использованием библиотеки `typer`, обеспечивающей автоматическую генерацию справки, валидацию параметров и удобную работу с аргументами командной строки.

3.2 Обработка конфиденциальных данных

Мастер-пароль и пароли сервисов запрашиваются через функцию `getpass.getpass()`, которая обеспечивает скрытый ввод без отображения символов на экране. Копирование в буфер обмена реализовано через `pyperclip`, поддерживающий различные системы управления буфером обмена.

4 Экспериментальное сравнение

4.1 Анализ запоминаемости

Рассмотрим два пароля одинаковой длины (16 символов):

- **Случайный**: `wsdKlndiQycrtmb`
- **Марковский**: `HaverimSbrEtoHa`

Марковский пароль содержит узнаваемые фонетические паттерны (например, `Haver`, `rim`, `Eto`), что облегчает запоминание. При этом энтропия обоих паролей сопоставима ($\approx$ 80 бит), так как случайная капитализация и смешивание алфавитов компенсируют структурную предсказуемость.

5 Обсуждение

5.1 Преимущества подхода

1. **Баланс безопасности и удобства**: генерируемые пароли сочетают высокую энтропию с улучшенной мнемоничностью
2. **Настраиваемость**: возможность изменения обучающего корпуса для адаптации стиля паролей
3. **Автономность**: все операции выполняются локально без сетевого взаимодействия
4. **Кросс-платформенность**: поддержка всех основных операционных систем
5. **Изоляция зависимостей**: использование pipx предотвращает конфликты библиотек

5.2 Ограничения

1. Качество генерации зависит от размера и характеристик обучающего корпуса
2. Отсутствие облачной синхронизации может быть неудобным для пользователей с несколькими устройствами
3. Потеря мастер-пароля приводит к невосстановимой потере всех сохраненных данных

5.3 Рекомендации по использованию

Для обеспечения максимальной безопасности рекомендуется:

- Использовать длинный мастер-пароль (фраза из нескольких слов)
- Создавать резервные копии файлов `passwords.db` и `key.salt`
- Использовать корпус текста объемом не менее 100 КБ для обучения модели
- Генерировать пароли длиной не менее 12 символов

6 Заключение

В работе представлена реализация менеджера паролей, демонстрирующая практическое применение марковских цепей для генерации криптографически стойких, но запоминаемых паролей. Система сочетает современные криптографические технологии с удобным интерфейсом командной строки и обеспечивает безопасное локальное хранение конфиденциальной информации.

Использование `pipx` для распространения приложения упрощает установку и обеспечивает изоляцию зависимостей, что делает систему доступной для широкого круга пользователей. Дальнейшее развитие проекта может включать реализацию графического интерфейса, поддержку двухфакторной аутентификации и механизмы безопасной синхронизации между устройствами.