\*\*Практическое задание по информационной безопасности\*\*

\*\*Курс: 3\*\*

\*\*Язык: C#\*\*

\*\*Тема: Безопасное хранение и обработка паролей с защитой от утечек данных\*\*

---

### \*\*Цель задания\*\*

Разработать консольное приложение, которое безопасно хранит учетные записи пользователей, шифрует пароли в памяти и на диске, а также защищает данные от атак через побочные каналы.

---

### \*\*Задачи\*\*

1. \*\*Безопасный ввод пароля\*\*

- Реализуйте ввод пароля с маскировкой (без отображения символов в консоли).

- Используйте `SecureString` для временного хранения пароля в памяти.

2. \*\*Шифрование данных\*\*

- Зашифруйте учетные данные (логин, пароль) с помощью \*\*AES-256-CTR\*\*.

- Ключ шифрования должен генерироваться из мастер-пароля пользователя через \*\*Argon2id\*\* с солью.

- Соль и nonce (для CTR) сохраняйте вместе с зашифрованными данными.

3. \*\*Безопасное хранение\*\*

- Сохраняйте зашифрованные данные в файл в формате JSON.

- Убедитесь, что пароли не остаются в памяти после шифрования (используйте `Array.Clear` для очистки буферов).

4. \*\*Проверка подлинности\*\*

- При расшифровке данных выполните проверку мастер-пароля через \*\*постоянное сравнение во времени\*\* (`CryptographicOperations.FixedTimeEquals`).

- В случае несовпадения пароля или повреждения данных — завершайте работу с информативной ошибкой.

---

### \*\*Требования к реализации\*\*

- \*\*Безопасность\*\*:

- Запретите кэширование паролей в памяти (например, отключите `GC.AddMemoryPressure`).

- Используйте \*\*эфемерные ключи\*\* (удаляйте их сразу после использования).

- Генерируйте случайные значения (соль, nonce) с помощью `Cryptography.RandomNumberGenerator`.

- \*\*Обработка ошибок\*\*:

- Отслеживайте попытки доступа к зашифрованным данным без ввода пароля.

- Обрабатывайте исключения (например, повреждение файла, неверный пароль).

- \*\*Без админских прав\*\*:

- Все файлы сохраняйте в папку `%APPDATA%` текущего пользователя.

- Используйте только стандартные библиотеки .NET (например, `System.Security.Cryptography`).

---

### \*\*Примерный алгоритм работы\*\*

1. \*\*Добавление учетной записи\*\*:

- Пользователь вводит мастер-пароль.

- Генерация ключа: Argon2id + случайная соль.

- Шифрование логина/пароля учетной записи с помощью AES-CTR.

- Сохранение соли, nonce и зашифрованных данных в JSON-файл.

2. \*\*Получение учетной записи\*\*:

- Пользователь вводит мастер-пароль.

- Чтение данных из файла.

- Проверка пароля через Argon2id и постоянное сравнение.

- Расшифровка данных, если проверка успешна.

---

### \*\*Проверка задания\*\*

1. Убедитесь, что пароли не хранятся в открытом виде в памяти (используйте инструменты вроде \*\*Process Hacker\*\* для проверки).

2. Попробуйте подобрать мастер-пароль методом атаки по времени — сравнение должно занимать одинаковое время.

3. Проверьте, что случайные значения (соль, nonce) уникальны для каждой записи.

---

### \*\*Подсказки\*\*

- Для реализации Argon2id используйте библиотеку [Isopoh.Cryptography.Argon2](https://www.nuget.org/packages/Isopoh.Cryptography.Argon2/) (не требует админских прав).

- Для работы с `SecureString` преобразуйте его в `Span<byte>` без кэширования.

- Пример структуры JSON:

```json

{

"salt": "base64string",

"nonce": "base64string",

"encryptedData": "base64string"

}

```

---

### \*\*Оценка\*\*

- Безопасный ввод и обработка паролей — 30%.

- Корректность шифрования/расшифровки — 30%.

- Защита от атак по времени и утечек данных — 40%.

---

Это задание требует понимания криптографических принципов, управления памятью и защиты от побочных каналов. Удачи!