УДК 004

Шифрования пароля на языке программирования C#

Ульянов Егор Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается и описывается разработка простого метода шифрования пароля при помощи хеширования и соления. Программа будет разрабатываться на языке программирования C# в среде разработки Visual Studio. Практическим результатом является разработанный метод.
Ключевые слова: шифрования, C#, хеширование, соление, среда разработки Visual Studio

Password encryption in the C# programming language

Ulianov Egor Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses and describes the development of a simple password encryption method using hashing and salting. The program will be developed in the C# programming language in the Visual Studio development environment. The practical result is the developed method.
Keywords: encryption, C#, hashing, salting, Visual Studio development environment

В сфере кибербезопасности защита паролей пользователей имеет первостепенное значение для защиты конфиденциальной информации. Хеширование и соление — это фундаментальные методы, используемые для повышения безопасности хранимых паролей. В C# разработчики могут использовать эти методы для защиты своих систем аутентификации от несанкционированного доступа и утечки данных.

Цель данной статьи создать простую звуковую панель, с базовыми функциями выбора и воспроизведение мелодий.

В результате своей работы Н. Н. Додобоев, О. И. Кукарцева, Я. А. Тынченко рассмотрели вопросы появления различных языков программирования (в частности C#), определения особенностей этих языков, а также составления основных видов и классификаций языков программирования [1]. З.С.Магомадова рассмотрела языки программирования высокого уровня, особенности, недостатки и сложности в изучении, а также описала несколько легких алгоритмов [2]. В статье Ф.В.

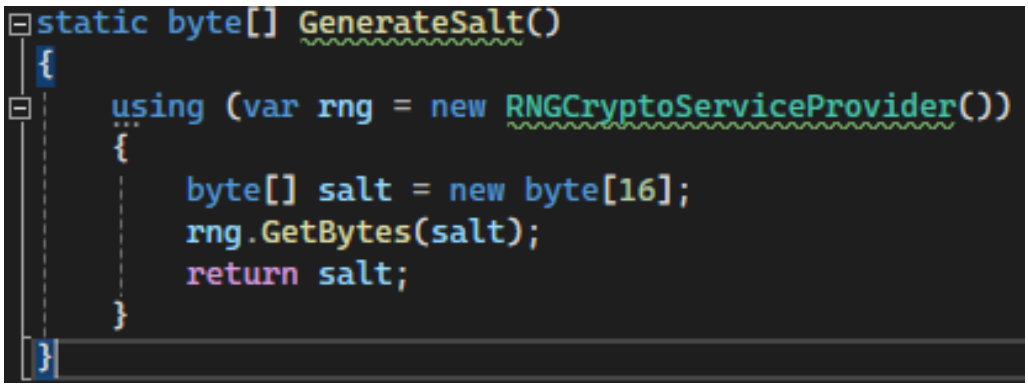
Патюченко, И.С. Слащев, А.В. Клименко, Л.А. Трегубенко были рассмотрены два подхода для создания программ на базе windows, обоснование выбора одного из них [3]. Бужинская Н.В., Масленников В.О. свою работу посвятили реализации криптографических методов в программировании, для популяризации криптографии. В статье описаны несколько несложных, но притом стойких к расшифровке методов шифрования информации.

Когда пользователь создает учетную запись или обновляет свой пароль, в игру вступает хеширование. Хеширование — это процесс преобразования открытого текста пароля в необратимую строку символов фиксированной длины. В C# разработчики часто используют для этой цели криптографические хэш-функции, такие как SHA-256 или bcrypt. Полученный хэш уникален для каждого пароля, что делает невозможным для злоумышленников обратить процесс вспять и получить исходный пароль см. рисунок 1.

```
using System.Reflection;
using System.Security.Cryptography;
using System.Text;

Ссылок: 4
static string HashPassword(string password, byte[] salt)
{
    using (var sha256 = new SHA256Managed())
    {
        byte[] passwordBytes = Encoding.UTF8.GetBytes(password);
        byte[] saltedPassword = new byte[passwordBytes.Length + salt.Length];


        Buffer.BlockCopy(passwordBytes, 0, saltedPassword, 0, passwordBytes.Length);
        Buffer.BlockCopy(salt, 0, saltedPassword, passwordBytes.Length, salt.Length);


        byte[] hashedBytes = sha256.ComputeHash(saltedPassword);


        byte[] hashedPasswordWithSalt = new byte[hashedBytes.Length + salt.Length];
        Buffer.BlockCopy(salt, 0, hashedPasswordWithSalt, 0, salt.Length);
        Buffer.BlockCopy(hashedBytes, 0, hashedPasswordWithSalt, salt.Length, hashedBytes.Length);

        return Convert.ToBase64String(hashedPasswordWithSalt);
    }
}
```

Рис. 1. Метод HashPassword

Хеширование само по себе, хотя и эффективно, может быть уязвимо для таких атак, как «атаки радужных таблиц». Здесь на помощь приходит «соление» паролей. Соль — это случайное значение, уникальное для каждого пользователя. Он объединяется с паролем перед хешированием, что добавляет дополнительный уровень сложности. Даже если у двух пользователей одинаковый пароль, их хеши будут различаться из-за уникальных солей см. рисунок 2.



```
static byte[] GenerateSalt()
{
    using (var rng = new RNGCryptoServiceProvider())
    {
        byte[] salt = new byte[16];
        rng.GetBytes(salt);
        return salt;
    }
}
```

Рис. 2. Метод GenerateSalt

В языке программирования C# полученный хешированный пароль и соль можно сохранить в базе данных. Получение и проверка паролей во время входа в систему включает в себя получение соли, объединение с введенным паролем, хеширование результата и сравнение с сохраненным хешем см. рисунок 3-5. Опишем основную логику программы:

1. UserDTO: Это класс Data Transfer Object (DTO), который используется для передачи данных пользователя между слоями приложения. Содержит свойства UserName, MobileNo, Password и ConfirmPassword.
2. IHashingPassword: Интерфейс, определяющий два метода: CreateUser для создания пользователя и UserVerify для проверки пользователя. Эти методы асинхронные, что указывает на использование операций ввода-вывода или других задач, которые могут выполняться в фоновом режиме.
3. HashingPassword: Класс, реализующий интерфейс IHashingPassword. Содержит следующие методы:
    - CreateUser: Принимает объект UserDTO и создает нового пользователя. Пароль хэшируется с использованием соли, и результат сохраняется в базе данных.
    - UserVerify: Принимает объект UserDTO и проверяет, совпадает ли введенный пароль с хэшированным паролем, хранящимся в базе данных.
4. DbContextCom: Предполагается, что это контекст базы данных Entity Framework, который используется для взаимодействия с базой данных.
5. Процесс хэширования и верификации:
    - При создании пользователя генерируется соль, пароль хэшируется с солью, и результаты сохраняются в базе данных.
    - При верификации пользователя хэш введенного пароля сравнивается с хэшированным паролем из базы данных.
6. Модель Usertest: Это модель, которая представляет таблицу пользователей в базе данных. Содержит поля, соответствующие свойствам UserDTO, а также дополнительные поля, такие как Email, IsActive, LastActiondatetime и Salt.

7. Асинхронность: Методы CreateUser и UserVerify являются асинхронными, что позволяет приложению оставаться отзывчивым во время выполнения операций базы данных.

```
using System.Reflection;
using System.Text;

Ссылок: 4
public class UserDTO
{
    Ссылок: 1
    public string UserName { get; set; }
    Ссылок: 2
    public string MobileNo { get; set; }
    Ссылок: 0
    public string Password { get; set; }
    Ссылок: 2
    public string ConfirmPassword { get; set; }
}

Ссылок: 1
public interface IHashingPassword
{
    Ссылок: 1
    public Task<string> CreateUser(UserDTO create);
    Ссылок: 1
    public Task<string> UserVerify(UserDTO verify);
}

Ссылок: 1
public class HashingPassword : IHashingPassword
{
    private readonly DbContextCom _dbContext;
    Ссылок: 0
    public HashingPassword(DbContextCom dbContext)
    {
        _dbContext = dbContext ?? throw new ArgumentNullException(nameof(dbContext));
    }
```

Рис. 3. Основные методы шифрования

```
public async Task<string> CreateUser(UserDTO create)
{
    string password = create.ConfirmPassword;

    byte[] saltBytes = GenerateSalt();
    // Hash the password with the salt
    string hashedPassword = HashPassword(password, saltBytes);
    string base64Salt = Convert.ToBase64String(saltBytes);

    byte[] retrievedSaltBytes = Convert.FromBase64String(base64Salt);

    var user = new Models.Usertest
    {
        ConfirmPassword = hashedPassword,
        Email = "",
        IsActive = true,
        LastActiondatetime = DateTime.Now,
        Mobile = create.MobileNo,
        Password = base64Salt,
        UserName = create.UserName,
        Salt = retrievedSaltBytes
    };
    _dbContext.Usertests.AddAsync(user);
    await _dbContext.SaveChangesAsync();

    return "User added successfully";
}
```

Рис. 4. Продолжение методов

```
public async Task<string> UserVerify(UserDTO verify)
{


    var user = _dbContext.Usertests.Where(x => x.Mobile == verify.MobileNo).Select(x => x).FirstOrDefault();

    string storedHashedPassword = user.ConfirmPassword;
    byte[] storedSaltBytes = user.Salt;
    string enteredPassword = verify.ConfirmPassword;


    byte[] enteredPasswordBytes = Encoding.UTF8.GetBytes(enteredPassword);


    byte[] saltedPassword = new byte[enteredPasswordBytes.Length + storedSaltBytes.Length];
    Buffer.BlockCopy(enteredPasswordBytes, 0, saltedPassword, 0, enteredPasswordBytes.Length);
    Buffer.BlockCopy(storedSaltBytes, 0, saltedPassword, enteredPasswordBytes.Length, storedSaltBytes.Length);


    string enteredPasswordHash = HashPassword(enteredPassword, storedSaltBytes);


    if (enteredPasswordHash == storedHashedPassword)
    {
        return "Password is correct.";
    }
    else
    {
        return "Password is incorrect.";
    }
}
```

Рис. 5. Продолжение методов

Хеширование и «соление» паролей в C# — важные методы построения надежных и безопасных систем аутентификации. Внедряя данные методы, разработчики могут значительно снизить риск несанкционированного доступа, гарантируя конфиденциальность учетных данных пользователей в постоянно меняющейся сфере кибербезопасности.

Библиографический список

1. Додобоев Н. Н., Кукарцева О. И., Тынченко Я. А. Современные языки программирования // Современные технологии: актуальные вопросы, достижения и инновации. 2014. №5. С. 81-85.
2. Магомадова З. С. Языки программирования высокого уровня // Разработка и применение наукоёмких технологий в эпоху глобальных трансформаций. 2020. №8. С. 94-96.
3. Патюченко Ф.В., Слащев И.С., Клименко А.В., Трегубенко Л.А. Windows form или windows presentation foundation // Modern science. 2019. №7-2. С. 318-320.
4. Бужинская, Н. В. Реализация криптографических методов шифрования на языке программирования c# // Ростовский научный журнал. 2019. № 2. С. 240-248.