**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**Факультет комп'ютерних наук та кібернетики**

**Кафедра математичної інформатики**

**Дисципліна «Проблеми кодування та захисту інформації»**

# ЗВІТ

з виконання практичної роботи №1

**на тему:**

**“Написати програму Автентифікації по Лемпорту”**

**Виконав аспірант**

Волохович Ігор

**Київ – 2025**

## ЗМІСТ

1. Вступ 2
2. Реалізація 2
3. Тести та валідація програми 5
**4. Висновок 6**

# 1. Вступ

У сучасному світі цифрових технологій безпека інформаційних систем стає критичною для забезпечення конфіденційності даних та запобігання несанкціонованому доступу. Автентифікація користувачів є однією з ключових складових цього процесу. Серед численних методів, що використовуються для перевірки особистості, особливо виділяється алгоритм Лемпорта, який базується на концепції одноразових паролів. Цей підхід, запропонований ще у 1980-х роках, демонструє високу стійкість до атак повторного використання облікових даних, оскільки кожен пароль застосовується лише один раз. Даний вступ присвячено розгляду основних принципів роботи алгоритму Лемпорта, його перевагам у порівнянні з традиційними методами автентифікації, а також викликам, які можуть виникати під час його впровадження у програмне забезпечення.

# 2. Реалізація

Код доступний за посиланням: [https://github.com/antomys/Lamport.Authentication](https://github.com/antomys/Lamport.Authentication)

```
public sealed class LamportAuthenticator
{
   // The current hash stored on the server.
   // This represents xn = H^n(secret), where H is the hash function.
   private byte[] _currentHash;

   /*
    * Constructor: Generates the hash chain.
    * Formula:
    *   Let secret = S and H(x) = SHA256(x)
    *   x0 = S
    *   xi = H(xi-1) for i = 1, 2, ..., n
    * The server stores xn (i.e., H applied n times to the secret).
    */
   public LamportAuthenticator(string secret, long iterations)
   {
       // Display the formula and steps using Spectre.Console markup.
       AnsiConsole.MarkupLine("[bold green]* Formula:*[/] " +
                              "[yellow]Let secret = S and H(x) = SHA256(x)[/]");
       AnsiConsole.MarkupLine("[bold green]* Calculation:*[/] " +
                              "[yellow]x0 = S, xi = H(xi-1) for i = 1, 2, ...,
n[/]");
       AnsiConsole.MarkupLine("[bold green]* Note:*[/] " +
```

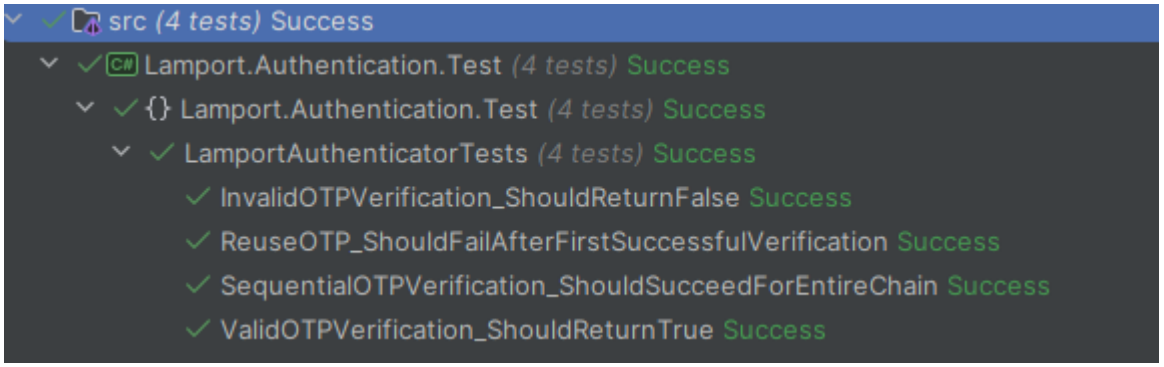
```
                            "[yellow]The server stores xn (i.e., H applied n times
to the secret).[/]");

        // Start with the secret as the initial value (x0).
        byte[] hash = Encoding.UTF8.GetBytes(secret);

        // Compute xn by applying H iteratively n times.
        for (int i = 0; i < iterations; i++)
        {
            hash = SHA256.HashData(hash);
        }
        // currentHash = xn
        _currentHash = hash;
    }

    // Returns the stored hash in hexadecimal form.
    // This value (xn) is registered on the server.
    public string GetCurrentHash()
    {
        return ByteArrayToHexString(_currentHash);
    }

    /*
     * VerifyOtp: Validates the one-time password (OTP) provided by the client.
     *
     * Algorithm steps:
     * 1. The client sends xn-1 (one-time password).
     * 2. The server computes H(xn-1) which should equal the stored xn.
     * 3. If H(xn-1) == xn, then update currentHash to xn-1.
     *
     * Formula:
     *   Provided OTP = xn-1, and the server checks:
     *     H(xn-1) ?= xn
     * If the equality holds, the OTP is valid.
     */
    public bool VerifyOtp(string providedOtp)
    {
        // Display the algorithm steps using Spectre.Console.
        AnsiConsole.MarkupLine("[bold blue]* Algorithm steps:*[/]");
        AnsiConsole.MarkupLine("[blue]1.[/] The client sends [italic]xn-1[/] (one-time
password).");
        AnsiConsole.MarkupLine("[blue]2.[/] The server computes [italic]H(xn-1)[/],
which should equal the stored [italic]xn[/].");
        AnsiConsole.MarkupLine("[blue]3.[/] If [italic]H(xn-1) == xn[/], update
current hash to [italic]xn-1[/].");

        // Convert the provided hexadecimal OTP into a byte array.
        byte[] otpBytes = HexStringToByteArray(providedOtp);
        AnsiConsole.MarkupLine($"[green]* Converting provided OTP to byte array:*[/]
[yellow]{ByteArrayToHexString(otpBytes)}[/]");
```

```
        // Compute H(providedOTP) i.e., H(xn-1)
        byte[] computedHash = SHA256.HashData(otpBytes);
        AnsiConsole.MarkupLine($"[green]* Computed H(providedOTP):*[/]
[yellow]{ByteArrayToHexString(computedHash)}[/]");

        // Check if H(xn-1) equals the stored xn.
        AnsiConsole.MarkupLine("[green]* Verifying:*[/] Checking if computed hash
equals the stored hash.");
        if (CompareByteArrays(computedHash, _currentHash))
        {
            // Update stored hash for next authentication round:
            // Now, _currentHash becomes xn-1.
            _currentHash = otpBytes;
            AnsiConsole.MarkupLine("[green]* Verification successful:*[/] OTP is
valid.");
            return true;
        }
        AnsiConsole.MarkupLine("[red]* Verification failed:*[/] OTP does not match.");
        return false;
    }

    // Helper method: Compare two byte arrays for equality.
    private static bool CompareByteArrays(byte[] a, byte[] b)
    {
        if (a.Length != b.Length)
        {
            return false;
        }
        for (int i = 0; i < a.Length; i++)
        {
            if (a[i] != b[i])
                return false;
        }
        return true;
    }

    // Helper method: Convert a byte array to a hexadecimal string.
    private static string ByteArrayToHexString(Span<byte> bytes)
    {
        var sb = new StringBuilder(bytes.Length * 2);
        foreach (byte b in bytes)
        {
            sb.Append(b.ToString("x2"));
        }
        return sb.ToString();
    }

    // Overload to accept a byte[] directly.
    private static string ByteArrayToHexString(byte[] bytes)
    {
        return ByteArrayToHexString(bytes.AsSpan());
```

```
    }

    // Helper method: Convert a hexadecimal string back to a byte array.
    private static byte[] HexStringToByteArray(string hex)
    {
        int numberChars = hex.Length;
        byte[] bytes = new byte[numberChars / 2];
        var bytesSpan = bytes.AsSpan();
        for (int i = 0; i < numberChars; i += 2)
        {
            bytesSpan[i / 2] = Convert.ToByte(hex.Substring(i, 2), 16);
        }
        return bytes;
    }
}
```

## 3. Тести та валідація програми

```
 Step 1: Define the secret and the number of hash iterations (n).

secret: This is a secret
number of hash iterations (n): 10101010

 Step 2: Server generates the hash chain.
 Computes: xn = H^n(secret) and stores it.

* Formula:* Let secret = S and H(x) = SHA256(x)
* Calculation:* x0 = S, xi = H(xi-1) for i = 1, 2, ., n
* Note:* The server stores xn (i.e., H applied n times to the secret).
Server stored hash (xn): 569d558a595bfb2345092435b7aef65f0c86a707beb168cc4d7333e74be23627

 Step 3: Client prepares the one-time password (OTP).
 The client computes xn-1 = H^(n-1)(secret) using one fewer iteration.
 This value will be used as the OTP.

Client generated OTP (xn-1): 8c359cbd7cbe7552f10735d2f6dedc8c6dcdee02105fd049a1e9500cea8ba2e9

 Step 4: Client sends the OTP to the server.
 Server verifies by computing H(OTP) and checking: H(xn-1) ?= xn
* Algorithm steps:*
1. The client sends xn-1 (one-time password).
2. The server computes H(xn-1), which should equal the stored xn.
3. If H(xn-1) == xn, update current hash to xn-1.
* Converting provided OTP to byte array:* 8c359cbd7cbe7552f10735d2f6dedc8c6dcdee02105fd049a1e9500cea8ba2e9
* Computed H(providedOTP):* 569d558a595bfb2345092435b7aef65f0c86a707beb168cc4d7333e74be23627
* Verifying:* Checking if computed hash equals the stored hash.
* Verification successful:* OTP is valid.
OTP verified: True

 Step 1: Define the secret and the number of hash iterations (n).
```




# 4. Висновок

Реалізація алгоритму автентифікації за Лемпортом за допомогою мови C# доводить ефективність застосування одноразових паролів у сучасних системах безпеки. Написані юніт-тести успішно демонструють працездатність та надійність реалізованого коду, підтверджуючи, що алгоритм коректно обробляє всі передбачені сценарії автентифікації. Це дозволяє впевнено інтегрувати розроблене рішення у різноманітні проекти для забезпечення високого рівня захисту від несанкціонованого доступу та атак повторного використання облікових даних. Загалом, проведена робота свідчить про потенціал алгоритму Лемпорта в реальних умовах експлуатації та відкриває перспективи для подальшого вдосконалення системи автентифікації.