ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра ЕОМ

КВАЛІФІКАЦІЙНА РОБОТА НА ТЕМУ:

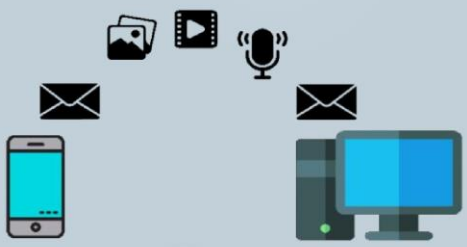
«ВЕБ-ЗАСТОСУНОК ДЛЯ ПРИХОВАНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ»

Здобувач:
Софія НЕСТЕРЕНКО
КІУКІ-21-5

Керівник:
ас. Олег Журило

Актуальність прихованої комунікації

- Підвищені вимоги до конфіденційності
- Зростання цензури та стеження в інтернеті
- Уразливість традиційних месенджерів
- Недостатність звичайного шифрування



Аналіз існуючих рішень

Протокол	Переваги	Недоліки	Тип архітектури	Шифрування
Signal	- Forward secrecy - Незалежний аудит - Відкритий код - Можливість передачі файлів	- Централізована інфраструктура - Прив'язка до номера телефону - Складність реалізації	Централізована	Наскрізне (E2EE) за замовчуванням (Double Ratchet, 3-DH)
Matrix	- Децентралізація - Відкритий стандарт - Можливість передачі файлів - Анонімність	- E2EE не за замовчуванням - Складність налаштування - Фрагментація мережі	Децентралізована, федеративна	Опціональне E2EE (Olm/Megolm)
XMPP	- Відкритий стандарт - Розширюваність (XEPs) - Можливість передачі файлів - Анонімність	- E2EE залежить від розширень - Вразливості безпеки - Несумісність розширень	Децентралізована, федеративна	E2EE через розширення (OMEMO)
IRC	- Простота - Низькі вимоги до ресурсів - Децентралізація - Анонімність	- Обмежені функції - Ризики приватності - Немає передачі файлів	Децентралізована	Без вбудованого E2EE
MTProto	- Висока швидкість - Багаторівневе шифрування - Оптимізація для мобільних пристроїв - Forward secrecy	- Пропрієтарний - Без незалежного аудиту - Централізована інфраструктура	Централізована	E2EE в секретних чатах (AES, RSA, TLS)

3

Постановка задачі

Створення месенджеру, що забезпечує прихований обмін інформацією з використанням стеганографії та має наступний функціонал:

- Приховування текстових повідомлень в зображеннях
- Шифрування повідомлень
- Групові чати зі стеганографією
- Одноразові повідомлення
- Доставка повідомлень офлайн-користувачам
- Експорт/імпорт локальних даних

4

Технології та засоби розробки

Технології клієнтської частини

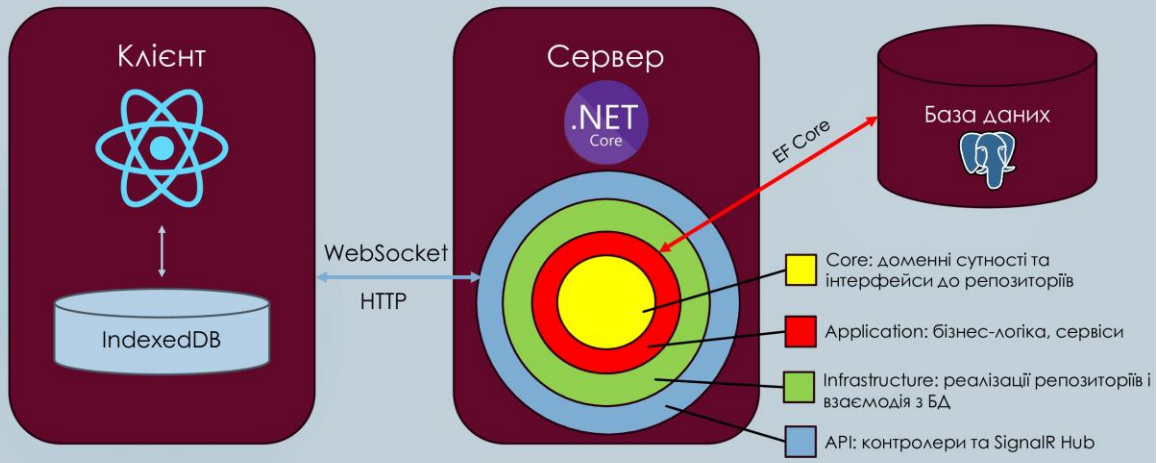
- •React
- •Material UI
- •Dexie.js + IndexedDB
- •AES+RSA

Технології серверної частини

- ASP.NET Core
- Entity Framework Core
- PostgreSQL
- SignalR
- LSB

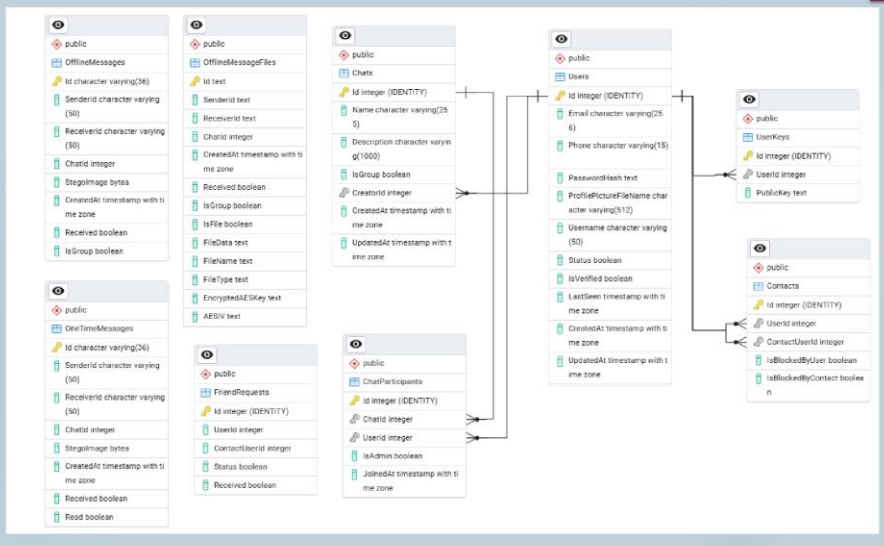
5

Архітектура застосунку



6

Структура бази даних



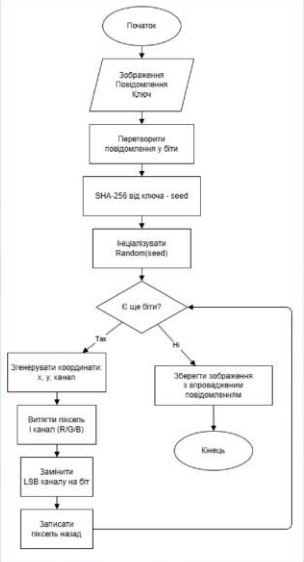
7

Схема шифрування повідомлення

- Повідомлення кодується у байти (UTF-8).
- Генерується випадковий симетричний ключ AES та IV.
- Повідомлення шифрується з використанням AES.
- Відкритий ключ отримувача використовується для шифрування AES-ключа (RSA).
- Отримувач може розшифрувати AES-ключ за допомогою свого приватного RSA-ключа.

8

Схема приховування повідомлення

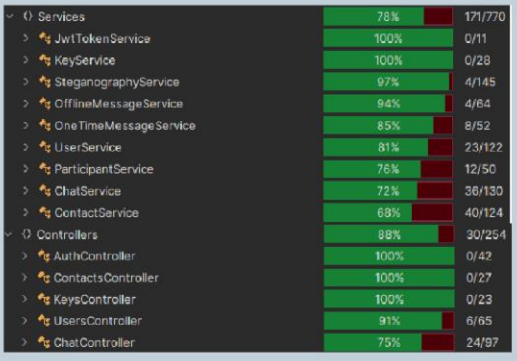


- Повідомлення перетворюється у біти.
- З ключа (senderId+receiverIdOrChatId) обчислюється seed (SHA-256), на його основі ініціалізується генератор псевдовипадкових чисел.
- Для кожного біта випадково генеруються координати (x, y) і канал (R, G або B).
- Молодший біт вибраного каналу замінюється на біт повідомлення.
- Отримане зображення з вбудованим повідомленням зберігається.

9

Юніт-тестування

- Забезпечено понад 60% покриття основної логіки.
- Використано моки та заглушки для ізоляції залежностей.
- Тестування підвищило надійність коду та зменшило кількість помилок на етапі розробки.

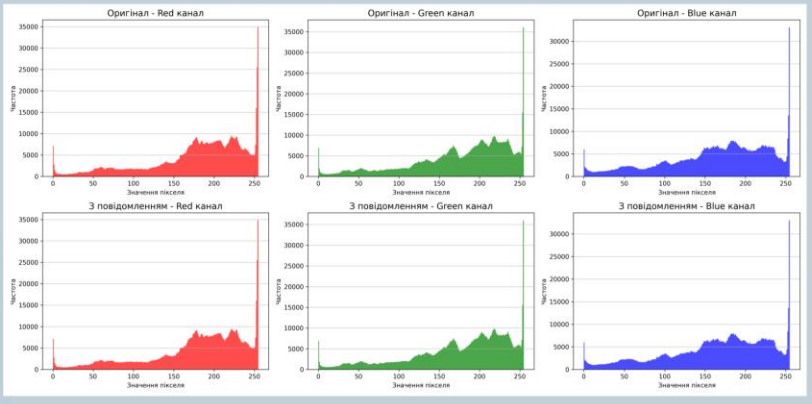


10

Тестування стеганографії

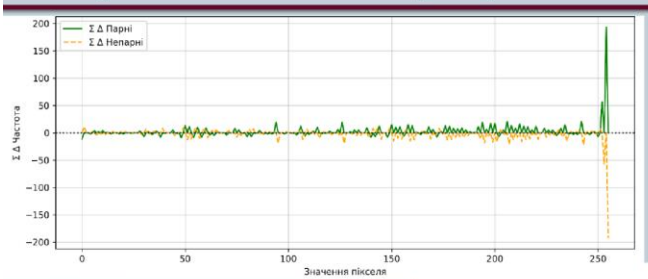
Оригінал

З повідомленням



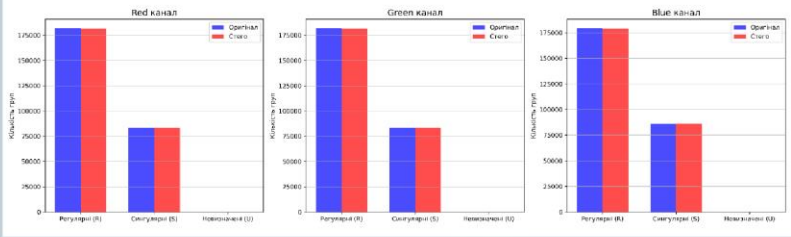
11

Тестування стеганографії



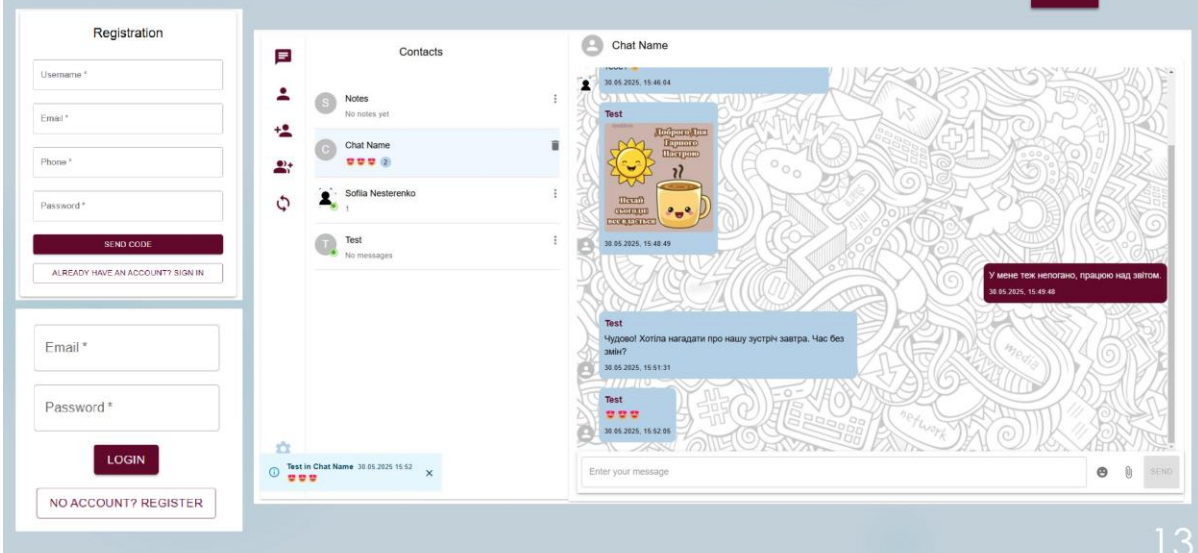
Парний аналіз

RS-аналіз



12

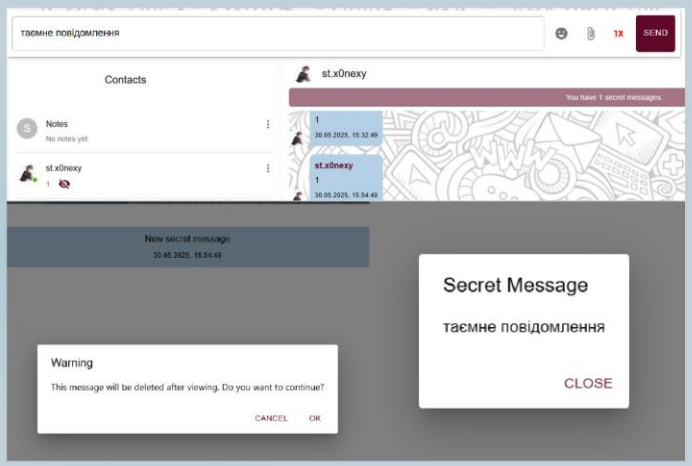
Інтерфейс застосунку



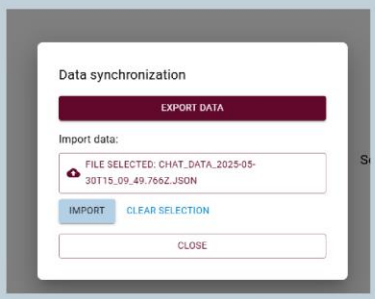
13

Інтерфейс застосунку

Обмін одноразовими повідомленнями



Синхронізація даних



14

Висновки

- Проведено аналіз методів захищеного обміну повідомленнями.
- Розроблено вебзастосунок для прихованої передачі повідомлень.
- Реалізовано комбінацію криптографії та стеганографії.
- Забезпечено можливість конфіденційного та безпечного обміну даними.
- Застосунок демонструє можливість інтеграції стеганографії в онлайн-комунікації.
- За результатами роботи були опубліковані тези на п'ятнадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління».

15

ДОДАТОК Б

Вихідний код сервісу стеганографії

```
using System.Security.Cryptography;
using System.Text;
using Microsoft.Extensions.Logging;
using PixChat.Application.Interfaces.Services;
using System.Drawing;
using System.Drawing.Imaging;
using Microsoft.Extensions.Options;
using PixChat.Application.Config;

namespace PixChat.Application.Services;

public class SteganographyService : ISteganographyService
{
    private readonly ILogger<SteganographyService> _logger;
    private const string EndMarker = "|X7K9P2M|";
    private readonly string _imageFolderPath;

    public SteganographyService(ILogger<SteganographyService>
logger, IOptions<ImageConfig> imageConfig)
    {
        _logger = logger;
        _imageFolderPath = imageConfig.Value.ImageFolderPath;
    }

    public byte[] EmbedMessage(byte[] image, string fullMessage,
string key)
    {
        byte[] messageBytes =
Encoding.UTF8.GetBytes(fullMessage);
        string messageBits = string.Join("",
messageBytes.Select(b => Convert.ToString(b, 2).PadLeft(8,
'0')));

        using (var memoryStream = new MemoryStream(image))
        using (var bitmap = new Bitmap(memoryStream))
        {
            int availableBits = bitmap.Width * bitmap.Height *
3;
            if (messageBits.Length > availableBits)
            {
                throw new ArgumentException($"Message is too
large to embed. Message bits: {messageBits.Length}, Available
bits: {availableBits}");
            }

            var pixelIndices =
```

```
GenerateRandomPixelIndices(bitmap.Width, bitmap.Height,
messageBits.Length, key);

            int bitIndex = 0;
            foreach (var (x, y, channel) in pixelIndices)
            {
                Color pixel = bitmap.GetPixel(x, y);
                int r = pixel.R;
                int g = pixel.G;
                int b = pixel.B;

                if (channel == 0) // R
                    r = (r & 0xFE) | (messageBits[bitIndex] ==
'1' ? 1 : 0);
                else if (channel == 1) // G
                    g = (g & 0xFE) | (messageBits[bitIndex] ==
'1' ? 1 : 0);
                else // B
                    b = (b & 0xFE) | (messageBits[bitIndex] ==
'1' ? 1 : 0);

                bitmap.SetPixel(x, y, Color.FromArgb(r, g, b));
                bitIndex++;
            }

            using (var outputStream = new MemoryStream())
            {
                bitmap.Save(outputStream, ImageFormat.Png);
                return outputStream.ToArray();
            }
        }
    }

    public (byte[] message, string encryptionKey, int
messageLength, DateTime timestamp,
        string encryptedAESKey, byte[] aesIV)
ExtractFullMessage(byte[] image, string key)
    {
        using (var memoryStream = new MemoryStream(image))
        using (var bitmap = new Bitmap(memoryStream))
        {
            int totalBits = bitmap.Width * bitmap.Height * 3;
            var pixelIndices =
GenerateRandomPixelIndices(bitmap.Width, bitmap.Height,
totalBits, key);

            StringBuilder messageBits = new StringBuilder();
            byte[] markerBytes =
Encoding.UTF8.GetBytes(EndMarker);
            int markerBitsLength = markerBytes.Length * 8;
            int bitsRead = 0;
            bool endMarkerFound = false;
```

```
            foreach (var (x, y, channel) in pixelIndices)
            {
                Color pixel = bitmap.GetPixel(x, y);
                int bit = channel == 0 ? pixel.R & 1 : channel
== 1 ? pixel.G & 1 : pixel.B & 1;
                messageBits.Append(bit);
                bitsRead++;

                if (bitsRead >= markerBitsLength)
                {
                    string currentBits =
messageBits.ToString(bitsRead - markerBitsLength,
markerBitsLength);
                    List<byte> currentBytes = new List<byte>();
                    for (int i = 0; i < currentBits.Length; i +=
8)
                    {
                        if (i + 8 > currentBits.Length) break;
                        string byteString =
currentBits.Substring(i, 8);

currentBytes.Add(Convert.ToByte(byteString, 2));
                    }
                    string currentString =
Encoding.UTF8.GetString(currentBytes.ToArray());
                    if (currentString == EndMarker)
                    {
                        endMarkerFound = true;
                        break;
                    }
                }
            }

            if (!endMarkerFound)
            {
                var ex = new FormatException("End marker not
found in image data.");
                _logger.LogError(ex, "End marker not found in
image data.");
                throw ex;
            }

            messageBits.Length = messageBits.Length -
markerBitsLength;

            List<byte> messageBytes = new List<byte>();
            for (int i = 0; i < messageBits.Length; i += 8)
            {
                if (i + 8 > messageBits.Length ) break;
                string byteString = messageBits.ToString(i, 8);
                messageBytes.Add(Convert.ToByte(byteString, 2));
            }
```

```
            string fullMessage =
Encoding.UTF8.GetString(messageBytes.ToArray());
            _logger.LogInformation($"Extracted full message:
{fullMessage}");
            string[] parts = fullMessage.Split('|');

            if (parts.Length != 5)
            {
                var ex = new FormatException($"Invalid
steganography data format. Expected 5 parts, got
{parts.Length}.");
                _logger.LogError(ex, $"Invalid steganography
data format. Expected 5 parts, got {parts.Length}. Full message:
'{fullMessage}'");
                throw ex;
            }


            int messageLength = int.Parse(parts[0]);
            byte[] message = Convert.FromBase64String(parts[1]);
            DateTime timestamp;
            string encryptedAESKey = parts[3];
            byte[] aesIV = Convert.FromBase64String(parts[4]);

            try
            {
                timestamp = DateTime.Parse(parts[2], null,
System.Globalization.DateTimeStyles.RoundtripKind);
            }
            catch (FormatException ex)
            {
                _logger.LogError($"Failed to parse timestamp:
'{parts[2]}'. Full message: '{fullMessage}'", ex);
                throw;
            }

            if (messageLength != message.Length)
            {
                _logger.LogWarning($"Message length mismatch:
expected {messageLength}, got {message.Length}");
            }

            return (message, null, messageLength, timestamp,
encryptedAESKey, aesIV);
        }
    }

    public byte[] GetRandomImage()
    {
        var files = Directory.GetFiles(_imageFolderPath,
"*.png");
        if (files.Length == 0)
        {
```

```
            throw new FileNotFoundException("No images found in
the folder.");
        }
        return File.ReadAllBytes(files[new
Random().Next(files.Length)]);
    }

    private List<(int x, int y, int channel)>
GenerateRandomPixelIndices(int width, int height, int
messageBitLength, string key)
    {
        var random = new Random(GenerateSeedFromKey(key));
        HashSet<(int x, int y, int channel)> indices = new
HashSet<(int x, int y, int channel)>();
        int totalAvailableBits = width * height * 3;

        if (messageBitLength > totalAvailableBits)
        {
            throw new ArgumentException($"Message bits
({messageBitLength}) exceed available pixel bits
({totalAvailableBits}).");
        }

        while (indices.Count < messageBitLength)
        {
            int x = random.Next(0, width);
            int y = random.Next(0, height);
            int channel = random.Next(0, 3);
            indices.Add((x, y, channel));
        }

        return indices.ToList();
    }

    private int GenerateSeedFromKey(string key)
    {
        using (SHA256 sha256 = SHA256.Create())
        {
            byte[] hashBytes =
sha256.ComputeHash(Encoding.UTF8.GetBytes(key));
            return BitConverter.ToInt32(hashBytes, 0);
        }
    }
}
```