МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття освітньо-кваліфікаційного рівня «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему «Порогові схеми розподілу секрету»
«Thereshold secret sparing schemes»

Виконав: студент денної форми навчання
напряму підготовки 6.050102 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)

Сівак Євген Олександрович
(прізвище, ім'я, по-батькові)

Керівник канд. фіз.-мат. наук, доц. Савастру О. В.
(науковий ступінь, вчене звання, прізвище та ініціали, підпис)

Рецензент канд. фіз.- мат. наук, доц. Бєлозьоров Г.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2019 р.

Завідувач кафедри

_____ П.Д. Варбанець
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «___» ____ 2019 р.

Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _____
(підпис) (прізвище, ініціали)

Одеса - 2019

АНОТАЦІЯ

В даному дипломному проекті на програмному рівні реалізована порогова схема розподілу секрету Шаміра. Розроблена програма створює безпечний розподіл секрету між деякою групою користувачів. Ініціатор має можливість розподілити секрет між групою користувачів таким чином, щоб забезпечити розподіл фрагментів секрету у вільному порядку та кількості між усіма користувачами в залежності від потреб групи. Для відновлення початкового секрету використовуються зашифрований секрет та певна мінімальна кількість фрагментів секрету, яку вказує ініціатор при створенні розподілу секрету.

Досліджено існуючі аналогічні схеми розподілу секрету і напрямки їх використання.

АННОТАЦИЯ

В данном дипломном проекте на программном уровне реализована пороговая схема распределения секрета Шамира. Разработаная программа создает безопасное распределение секрета между некоторой группой пользователей. Инициатор имеет возможность распределить секрет между участниками таким образом, чтобы обеспечить распределение фрагментов секрета в свободном порядке и количестве между всеми пользователями в зависимости от потребностей группы. Для восстановления первоначального секрета используются зашифрованный секрет и определенное минимальное количество фрагментов секрета, которую указывает инициатор при создании распределения секрета.

Исследованы существующие аналогичные схемы распределения секрета и направления их использования.

ABSTRACT

In this diploma project Shamir’s thereshold sharing is implemented on the software level. The developed program creates a safe distribution of secret between a certain group of users. The initiator has the ability to distribute secrets between a group of users in such a way as to ensure the distribution of secret secrecy fragments among all users depending on the needs of the group. To restore the initial secrecy, an encrypted secret and a certain minimum number of secrecy fragments, which the initiator indicates when creating a secrecy distribution, is used.

Existing similar schemes of secretion distribution and directions of their use are investigated.

ЗМІСТ

ВСТУП

Те, що інформація має цінність, люди усвідомили дуже давно - недарма листування сильних світу цього здавна була об'єктом пильної уваги їх друзів і недругів. Тоді і виникла потреба захисту цієї інформації від надмірно зацікавлених. В давнину люди намагалися використовувати для вирішення цього завдання найрізноманітніші методи, і одним з них був тайнопис - вміння складати повідомлення таким чином, щоб його зміст був недоступний нікому крім тих, для кого він і був призначений. Є свідчення того, що мистецтво тайнопису зародилося ще в доантичні часи. В наш час інформація набула самостійну комерційну цінність і стала майже звичайним товаром. Її виробляють, зберігають, транспортують, продають і купують, а значить - крадуть і підроблюють, отже, її необхідно захищати. Широке застосування комп'ютерних технологій і постійне збільшення обсягу інформаційних потоків викликає постійне зростання інтересу до криптографії. Останнім часом збільшується роль програмних засобів захисту інформації, які не потребують великих фінансових витрат в порівнянні з апаратними криптосистемами. Сучасні методи шифрування гарантують практично абсолютний захист даних. Проте на сьогоднішній день разом з потребою в максимальній захищеності інформації також зростає і потреба в комфортному використанні інформації тими, для кого вона призначена. Тобто при спробі реалізувати ефективну систему захисту будь яких даних, з’являються й інші фактори, з якими фахівцям з безпеки доводиться рахуватися, створюючи індивідуальні рішення для різних типів проблем. Одним із таких рішень було розділення секрету по частинах. Хоча таке рішення створило іншу проблему, яку ще в 1976 році сформулювали Мартін Геллман та Вітфілд Діффі [1]. Проблема була сформульована наступним чином: Одинадцять вчених працюють над секретним проектом. Вони хочуть заблокувати документи в шафі, щоб відкрити шафу, якщо і тільки якщо присутні шість або більше вчених. Яку мінімальну кількість замків

необхідно? Яку найменшу кількість ключів від замків кожен вчений повинен нести? Насправді нескладно зрозуміти, що кількість замків і ключів потрібних для кожного вченого буде неймовірно висока і вона стане експоненціально гірше, коли число вчених збільшиться. Також складність ефективного поділу секрету полягає у тому, що якщо була скомпрометована одна з частин секрету, є висока ймовірність того, що і секрет в цілому буде скомпрометовано. Саме для цього й було створено спеціальні схеми розподілу секрету, які складні в розкритті навіть у випадку компрометації однієї або кількох частин секрету.

Таким чином, метою даної дипломної роботи є виконання аналізу існуючих алгоритмів розподілу секрету і реалізація розподілення секрету за одним з алгоритмів програмному рівні.

Для досягнення поставленої мети потрібно виконати наступні задачі:

1. Виконати дослідження джерел та підготовку матеріалів про алгоритми розподілу секрету.
2. Проаналізувати та описати існуючі алгоритми розподілення секрету.
3. Провести моделювання та тестування розподілення секрету, спроектованого в середовищі розробки Visual Studio, використовуючи мову C#.
4. Зробити висновок щодо досягнутої безпеки за допомогою реалізованого розподілу секрету та ефективності роботи алгоритму при різній кількості учасників.

ВИСНОВОК

В ході даної роботи були розібрані теоретичні питання щодо видів схем розділення секрету та алгоритмів їх реалізації. Було обрано порогову схему розподілу секрету Шаміра через ефективність алгоритму. При оцінці продуктивності програма, що реалізує порогову схему розподілу секрету за алгоритмом Шаміра ефективно працює при кількості учасників до 200. Тому ефективність схеми розподілу секрету обмежена кількістю учасників. Основна перевага цієї схеми розподілу секрету полягає в більш економних потребах в використанні машинних потужностей.

Розроблена система порогового розподілу секрету може бути ефективно використана для розподілу секрету як між малими, так і великими групами учасників. Окрім того, цю схему можна використати в системі електронного голосування для забезпечення безпеки, достовірності та конфіденційності.

Подальша робота може вестись у напрямку покращення алгоритму порогового розподілу секрету, чи використання іншої схеми, наприклад, з можливістю перевірки долей, яка краще підходить для ектронного голосування тому, що не потребує незлочинного ініціатора розподілу чи перевіреного каналу зв’язку для розподілення долей секрету між учасниками, інакше з’являється загроза безпеки через саботаж зі сторони ініціатора.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Diffie W., Hellman M. E. New Directions in Cryptography // IEEE Transactions on Information Theory. — 1976. — URL: https://www.researchgate.net/publication/3082825_New_Directions_in_Cryptography.
2. А.П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А.В. Черемушкин // Основы криптографии, 2-е издание — 2002. — URL: https://studfiles.net/preview/6311470/
3. Adi Shamir How to share a secret // Massachusetts Institute of Technology, Cambridge — Nov. 1976. — URL: https://dl.acm.org/citation.cfm?doid=359168.359176
4. Pomerance C. Advances in Cryptology — CRYPTO ’87. T. 293. — Springer, 1988.
5. Blakley G. R. Safeguarding cryptographic keys // Proceedings of the 1979 AFIPS National Computer Conference. — 1979. — С. 311— 317. —URL: https://www.computer.org/csdl/proceedingsarticle/afips/1979/50870313/12OmNCeK2a1
6. E. Bresson; J. Stern & M. Szydlo (2002), "Threshold ring signatures and applications to ad-hocgroups", Advances in Cryptology: Crypto 2002: 465–480 URL: https://link.springer.com/chapter/10.1007%2F3-540-45708-9_30

ДОДАТОК
ПРОГРАМНА РЕАЛІЗАЦІЯ ОСНОВНИХ МЕТОДІВ

```
//клас, за допомогою якого відбувається шифрування секрету за
алгоритмом RSA (Rijndael)

public class Encryption

{

    public static byte[] Encrypt(byte[] clearData, byte[] Key,
byte[] IV)

    {

        MemoryStream ms = new MemoryStream();

        Rijndael alg = Rijndael.Create();

        alg.Key = Key;

        alg.IV = IV;

        CryptoStream cs = new CryptoStream(ms,

           alg.CreateEncryptor(), CryptoStreamMode.Write);

        cs.Write(clearData, 0, clearData.Length);

        cs.Close();

        byte[] encryptedData = ms.ToArray();


        return encryptedData;

    }

    public static string Encrypt(string clearText, string
Password)

    {

        byte[] clearBytes =

          System.Text.Encoding.Unicode.GetBytes(clearText);

        PasswordDeriveBytes pdb = new
PasswordDeriveBytes(Password,

            new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,
```

```
            0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76});
        byte[] encryptedData = Encrypt(clearBytes,
                 pdb.GetBytes(32), pdb.GetBytes(16));
        return Convert.ToBase64String(encryptedData);
    }
    public static byte[] Encrypt(byte[] clearData, string
Password)
    {
        PasswordDeriveBytes pdb = new
PasswordDeriveBytes(Password,
            new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,
            0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76});
        return Encrypt(clearData, pdb.GetBytes(32),
pdb.GetBytes(16));
    }
    public static void Encrypt(string fileIn,
                string fileOut, string Password)
    {
        FileStream fsIn = new FileStream(fileIn,
            FileMode.Open, FileAccess.Read);
        FileStream fsOut = new FileStream(fileOut,
            FileMode.OpenOrCreate, FileAccess.Write);
        PasswordDeriveBytes pdb = new
PasswordDeriveBytes(Password,
            new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,
            0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76});
        Rijndael alg = Rijndael.Create();
        alg.Key = pdb.GetBytes(32);
        alg.IV = pdb.GetBytes(16);
        CryptoStream cs = new CryptoStream(fsOut,
```

```
            alg.CreateEncryptor(), CryptoStreamMode.Write);
can
        int bufferLen = 4096;
        byte[] buffer = new byte[bufferLen];
        int bytesRead;
        do
        {
            bytesRead = fsIn.Read(buffer, 0, bufferLen);
            cs.Write(buffer, 0, bytesRead);
        } while (bytesRead != 0);
        cs.Close();
        fsIn.Close();
    }
    public static byte[] Decrypt(byte[] cipherData,
                                 byte[] Key, byte[] IV)
    {
        MemoryStream ms = new MemoryStream();
        Rijndael alg = Rijndael.Create();
        alg.Key = Key;
        alg.IV = IV;
        CryptoStream cs = new CryptoStream(ms,
            alg.CreateDecryptor(), CryptoStreamMode.Write);
        cs.Write(cipherData, 0, cipherData.Length);
        cs.Close();
        byte[] decryptedData = ms.ToArray();

        return decryptedData;
    }
```

```
    public static string Decrypt(string cipherText, string
Password)

    {

        byte[] cipherBytes =
Convert.FromBase64String(cipherText);

        PasswordDeriveBytes pdb = new
PasswordDeriveBytes(Password,

            new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,
0x65,

            0x64, 0x76, 0x65, 0x64, 0x65, 0x76});

        byte[] decryptedData = Decrypt(cipherBytes,

            pdb.GetBytes(32), pdb.GetBytes(16));

        return
System.Text.Encoding.Unicode.GetString(decryptedData);

    }

    public static byte[] Decrypt(byte[] cipherData, string
Password)

    {

        PasswordDeriveBytes pdb = new
PasswordDeriveBytes(Password,

            new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,

            0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76});

        return Decrypt(cipherData, pdb.GetBytes(32),
pdb.GetBytes(16));

    }

    public static void Decrypt(string fileIn,

                string fileOut, string Password)

    {

        FileStream fsIn = new FileStream(fileIn,

                    FileMode.Open, FileAccess.Read);

        FileStream fsOut = new FileStream(fileOut,
```

```
                    FileMode.OpenOrCreate, FileAccess.Write);
        PasswordDeriveBytes pdb = new
PasswordDeriveBytes(Password,
            new byte[] {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d,
            0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76});
        Rijndael alg = Rijndael.Create();

        alg.Key = pdb.GetBytes(32);
        alg.IV = pdb.GetBytes(16);
        CryptoStream cs = new CryptoStream(fsOut,
            alg.CreateDecryptor(), CryptoStreamMode.Write);
        int bufferLen = 4096;
        byte[] buffer = new byte[bufferLen];
        int bytesRead;
        do
        {
            bytesRead = fsIn.Read(buffer, 0, bufferLen);
            cs.Write(buffer, 0, bytesRead);
        } while (bytesRead != 0);
        cs.Close();
        fsIn.Close();
    }
}
//клас, який генерує ключ для розподілу секрету
    class KeyGenerator
    {
        private static Random rand = new Random();
        public static byte[] GenerateKey(int bitsLength)
        {
            if (bitsLength % 8 != 0)
```

```
            {
                throw new ArgumentException("The bits length
must be able to be divided by 8.");
            }

            int bytesLength = bitsLength / 8;
            var key = new byte[bytesLength];
            rand.NextBytes(key);

            return key;
        }

        public static ushort[] GenerateDoubleBytesKey(byte[]
arr)
        {
            if (arr.Length % 2 != 0)
            {
                throw new ArgumentException("The array length
must be even.");
            }

            int length = arr.Length;
            var result = new ushort[length / 2];

            ushort el1, el2;
            for (int i = 0; i < length / 2; i++)
            {
                el1 = (ushort)(arr[2 * i]<<8);
                el2 = arr[2 * i + 1];
                result[i] = (ushort)(el1+ el2);
            }
            return result;
        }
        public static string GetHexKey(ushort[] key)
        {
            var newKey = new uint[key.Length];
            for (int i = 0; i < key.Length; i++)
            {
                newKey[i] = key[i];
            }
            return HexConverter.NumbersArrToHexString(newKey, '-
');
        }
    }
}
```

```
//клас, який генерує поліном для розподілення секрету
    class PolynomGenerator
    {
        public static uint lowerXBound = 1;
        public static uint upperXBound = Mod.MOD - 1;

        private static Random rand = new Random();

        public static ushort[] GenerateWholeNumbers(int count)
        {
            var result = new ushort[count];

            for (int i = 0; i < count; i++)
            {
                result[i] = (ushort)(rand.Next(ushort.MinValue,
ushort.MaxValue));
            }

            return result;
        }

        public static ushort[,] GeneratePolynomsMatrix(ushort[]
key, int count, int power)
        {
            var matrix = new ushort[count, power + 1];

            var currentCoefficients = new ushort[power + 1];

            for (int i = 0; i < count; i++)
            {
                currentCoefficients = GenerateWholeNumbers(power
+ 1);
                currentCoefficients[0] = key[i];
                matrix.SetRow(i, currentCoefficients);
            }

            return matrix;
        }

        public static uint GetRandomX(uint lowerBond, uint
upperBond)
        {
            uint thirtyBits = (uint)rand.Next(1 << 30);
            uint twoBits = (uint)rand.Next(1 << 2);
            return (thirtyBits << 2) | twoBits;
```

```
        }
    }
}
 //клас, розподіляючий секрет на фрагменти
  public class SharesManager
    {
        public static string[] SplitKey(ushort[] key, int
players, int required)
        {
            int keyLengthInBits = key.Length * 16;
            int polynomsCount = key.Length;

            var polynoms =
PolynomGenerator.GeneratePolynomsMatrix(key, polynomsCount,
required - 1);

            /* Console.WriteLine("Generated matrix:");
            for (int i = 0; i < polynoms.GetLength(0); i++)
            {
                for (int j = 0; j < polynoms.GetLength(1); j++)
                {
                    Console.Write("{0} ", polynoms[i, j]);
                }
                Console.WriteLine();
            } */

            var playerPoints = new uint[players, polynomsCount,
2];
            var currentPolynomPoints = new uint[players, 2];
            var currentPolynom = new ushort[required];

            for (int i = 0; i < polynomsCount; i++)
            {
                currentPolynom = polynoms.GetRow(i);
                currentPolynomPoints =
PolynomSolver.GetRandomPoints(currentPolynom, players);
                for (int j = 0; j < players; j++)
                {
                    playerPoints[j, i, 0] =
currentPolynomPoints[j, 0];
                    playerPoints[j, i, 1] =
currentPolynomPoints[j, 1];
                }
            }

            var geneneratedHexes = new string[players];
```

```
            var currentPlayerPoints = new uint[polynomsCount *
2];
            var counter = 0;
            for (int i = 0; i < playerPoints.GetLength(0); i++)
            {
                for (int j = 0; j < playerPoints.GetLength(1);
j++)
                {
                    currentPlayerPoints[counter] =
playerPoints[i, j, 0];
                    counter++;
                    currentPlayerPoints[counter] =
playerPoints[i, j, 1];
                    counter++;
                }
                counter = 0;
                geneneratedHexes[i] =
HexConverter.NumbersArrToHexString(currentPlayerPoints, '-');
            }

            return geneneratedHexes;
        }

        public static ushort[] CombineKey(string[] playerPoints)
        {
            var polynomsCount = (playerPoints[0].Split('-
').Length - 1) / 2 + 1;
            var required = playerPoints.Length;

            var convertedPoints = new uint[polynomsCount,
required, 2];

            var testingPlayersPoints = new uint[polynomsCount];
            var counter = 0;

            var currentPlayerPoints = new uint[polynomsCount];

            for (int i = 0; i < required; i++)
            {
                currentPlayerPoints =
HexConverter.HexStringToNumbersArr(playerPoints[i], '-');

                for (int j = 0; j < polynomsCount; j++)
                {
```

```
                    convertedPoints[j, i, 0] =
currentPlayerPoints[counter];
                    counter++;
                    convertedPoints[j, i, 1] =
currentPlayerPoints[counter];
                    counter++;
                }
                counter = 0;
            }

           /* for (int i = 0; i < convertedPoints.GetLength(0);
i++)
            {
                for (int j = 0; j <
convertedPoints.GetLength(1); j++)
                {
                    Console.Write("({0}, {1})",
convertedPoints[i, j, 0], convertedPoints[i, j, 1]);
                }
                Console.WriteLine();
            } */

            var testingSolved =
CoefficientsSolver.GetCoefficients(convertedPoints);

            return testingSolved;
        }
    }
```