МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1—98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ОТЧЕТ ПО ПРАКТИЧЕСКИМ ЗАНЯТИЯМ**

по дисциплине «Основы информационной безопасности»

**Исполнитель**

студент(ка) 2 курса \_7\_ группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Банкузов Михаил

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ржеутская Н. В.

должность, ученая степень, ученое звание подпись, дата

**Практическое задание №4.1**

**Тема «Криптографическая защита информации»**

Цель: Овладение основными криптографическими алгоритмами симметричного шифрования .

**Теоретическое введение**

Криптография - наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним) и аутентичности (целостности и подлинности авторства) информации.

Изначально криптография изучала методы шифрования информации – обратимого преобразования открытого (исходного) текста на основе секретного алгоритма и/или ключа в шифрованный текст (шифротекст). Традиционная криптография образует раздел симметричных криптосистем, в которых зашифрование и расшифрование проводится с использованием одного и того же секретного ключа.

Помимо этого современная криптография включает в себя асимметричные криптосистемы, системы электронной цифровой подписи, хеш-функции, управление ключами, получение скрытой информации, квантовую криптографию.

Шифрованием (encryption) называют процесс преобразования открытых данных (plaintext) в зашифрованные (шифртекст, ciphertext) или зашифрованных данных в открытые по определенным правилам с применением ключей.

В англоязычной литературе зашифрование / расшифрование – enciphering / deciphering.

Классификация алгоритмов шифрования

1. Симметричные (с секретным, единым ключом, одноключевые, single-key).

1.1. Потоковые:

· с одноразовым или бесконечным ключом (infinite-key cipher);

· с конечным ключом;

· на основе генератора псевдослучайных чисел.

1.2. Блочные:

1.2.1. Шифры перестановки (permutation, P-блоки);

1.2.2. Шифры замены (substitution, S-блоки):

· моноалфавитные;

· полиалфавитные;

2. Асимметричные (с открытым ключом, public-key):

· Диффи-Хеллман DH (Diffie, Hellman);

· Райвест-Шамир-Адлeман RSA (Rivest, Shamir, Adleman);

· Эль-Гамаль (ElGamal).

Симметричные алгоритмы шифрования (или криптография с секретными ключами) основаны на том, что отправитель и получатель информации используют один и тот же ключ. Этот ключ должен храниться в тайне и передаваться способом, исключающим его перехват.

Обмен информацией осуществляется в 3 этапа:

* отправитель передает получателю ключ (в случае сети с несколькими абонентами у каждой пары абонентов должен быть свой ключ, отличный от ключей других пар);
* отправитель, используя ключ, зашифровывает сообщение, которое пересылается получателю;
* получатель получает сообщение и расшифровывает его.

Если для каждого дня и для каждого сеанса связи будет использоваться уникальный ключ, это повысит защищенность системы.

При блочном шифровании информация разбивается на блоки фиксированной длины и шифруется поблочно. Блочные шифры бывают двух основных видов:

· шифры перестановки (transposition, permutation, P-блоки);

· шифры замены (подстановки, substitution, S-блоки).

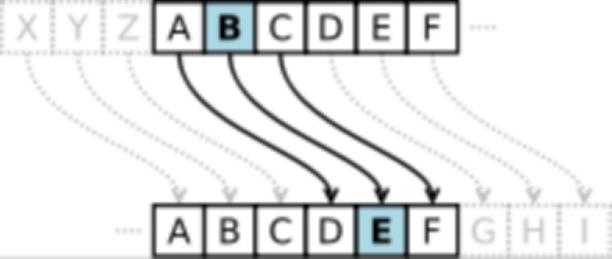
Шифры перестановок переставляют элементы открытых данных (биты, буквы, символы) в некотором новом порядке. Различают шифры горизонтальной, вертикальной, двойной перестановки, решетки, лабиринты, лозунговые и др.

Шифры замены заменяют элементы открытых данных на другие элементы по определенному правилу. Paзличают шифры простой, сложной, парной замены, буквенно-слоговое шифрование и шифры колонной замены. Шифры замены делятся на две группы:

· моноалфавитные (код Цезаря);

· полиалфавитные (шифр Видженера, цилиндр Джефферсона, диск Уэтстоуна, Enigma).

В моноалфавитных шифрах замены буква исходного текста заменяется на другую, заранее определенную букву. Например в коде Цезаря буква заменяется на букву, отстоящую от нее в латинском алфавите на некоторое число позиций.



Очевидно, что такой шифр взламывается совсем просто. Нужно подсчитать, как часто встречаются буквы в зашифрованном тексте, и сопоставить результат с известной для каждого языка частотой встречаемости букв.

В полиалфавитных подстановках для замены некоторого символа исходного сообщения в каждом случае его появления последовательно используются различные символы из некоторого набора. Понятно, что этот набор не бесконечен, через какое-то количество символов его нужно использовать снова. В этом слабость чисто полиалфавитных шифров.

В современных криптографических системах, как правило, используют оба способа шифрования (замены и перестановки). Такой шифратор называют составным (product cipher). Oн более стойкий, чем шифратор, использующий только замены или перестановки.

В асимметричных алгоритмах шифрования (или криптографии с открытым ключом) для зашифровывания информации используют один ключ (открытый), а для расшифровывания - другой (секретный). Эти ключи различны и не могут быть получены один из другого.

Схема обмена информацией такова:

· получатель вычисляет открытый и секретный ключи, секретный ключ хранит в тайне, открытый же делает доступным (сообщает отправителю, группе пользователей сети, публикует);

· отправитель, используя открытый ключ получателя, зашифровывает сообщение, которое пересылается получателю;

· получатель получает сообщение и расшифровывает его, используя свой секретный ключ.

**Задание к выполнению**

1. Изучить теоретические сведения по данной теме.
2. Зашифровать сообщение с использованием шифра Цезаря, Трисемуса, Плейфейра и Вижинера и полученного секретного ключа (по номеру варианта и ключевому слову «Защита»). В качестве сообщения использовать свою Фамилию Имя Отчество.

* Шифр Цезаря с ключом 5: Банкузов Михаил Олегович – Ёетпшмуж Снъенр Урйзужнь

public class CaesarCipher

{

const string alphabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

private string CodeEncode(string text, int k)

{

var fullAlphabet = alphabet + alphabet.ToLower();

var letterQty = fullAlphabet.Length;

var retVal = "";

for (int i = 0; i < text.Length; i++)

{

var c = text[i];

var index = fullAlphabet.IndexOf(c);

if (index < 0)

{

retVal += c.ToString();

}

else

{

var codeIndex = (letterQty + index + k) % letterQty;

retVal += fullAlphabet[codeIndex];

}

}

return retVal;

}

public string Encrypt(string plainMessage, int key)

=> CodeEncode(plainMessage, key);

public string Decrypt(string encryptedMessage, int key)

=> CodeEncode(encryptedMessage, -key);

}

* Шифр Трисемуса:

class TrisemusCipher

{

private char[,] trisemusSquare;

public TrisemusCipher(string key)

{

InitializeTrisemusSquare(key);

}

private void InitializeTrisemusSquare(string key)

{

string alphabet = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

trisemusSquare = new char[4, 8];

string uniqueKey = new string(key.ToUpper().Distinct().ToArray());

int index = 0;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 8; j++)

{

if (index < uniqueKey.Length)

{

trisemusSquare[i, j] = uniqueKey[index++];

}

}

}

foreach (char ch in alphabet)

{

if (index < 32)

{

if (!uniqueKey.Contains(ch))

{

trisemusSquare[index / 8, index % 8] = ch;

index++;

}

}

else

{

break;

}

}

}

private Tuple<int, int> GetCharPosition(char ch)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 8; j++)

{

if (trisemusSquare[i, j] == ch)

{

return Tuple.Create(i, j);

}

}

}

return null;

}

public string Encrypt(string plaintext)

{

plaintext = plaintext.ToUpper();

string ciphertext = "";

foreach (char ch in plaintext)

{

if (ch == ' ')

{

ciphertext += ' ';

continue;

}

var charPosition = GetCharPosition(ch);

if (charPosition != null)

{

int row = charPosition.Item1;

int col = charPosition.Item2;

ciphertext += trisemusSquare[(row + 1) % 4, col];

}

}

return ciphertext;

}

}

ЛЕЦУЬДЧМ ХЙЮЕЙФ ЧФПНЧМЙЗ

* Шифр Вижинера с ключом Защита:

public class VigenereCipher

{

const string defaultAlphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

readonly string letters;

public VigenereCipher(string alphabet = null)

{

letters = string.IsNullOrEmpty(alphabet) ? defaultAlphabet : alphabet;

}

private string GetRepeatKey(string s, int n)

{

var p = s;

while (p.Length < n)

{

p += p;

}

return p.Substring(0, n);

}

private string Vigenere(string text, string password, bool encrypting = true)

{

var gamma = GetRepeatKey(password, text.Length);

var retValue = "";

var q = letters.Length;

for (int i = 0; i < text.Length; i++)

{

var letterIndex = letters.IndexOf(text[i]);

var codeIndex = letters.IndexOf(gamma[i]);

if (letterIndex < 0)

{

retValue += text[i].ToString();

}

else

{

retValue += letters[(q + letterIndex + ((encrypting ? 1 : -1) \* codeIndex)) % q].ToString();

}

}

return retValue;

}

public string Encrypt(string plainMessage, string password)

=> Vigenere(plainMessage, password);

public string Decrypt(string encryptedMessage, string password)

=> Vigenere(encryptedMessage, password, false);

}

Банкузов Михаил Олегович – взлжотпб сдывщм рждбпбдю

* Шифр Плейфейра:

public abstract class PlayFairSettings

{

protected HashSet<char> HsAlphabet { get; private set; }

protected Dictionary<char, char> Replaces { get; private set; }

protected abstract char[] AlphabetChars { get; }

public abstract int Columns { get; }

public abstract int Rows { get; }

public abstract char Replacer { get; }

public IEnumerable<char> Alphabet

{

get

{

char[] chars = AlphabetChars;

foreach (var c in chars)

yield return c;

}

}

public PlayFairSettings()

{

HsAlphabet = new HashSet<char>();

Replaces = new Dictionary<char, char>();

foreach (var c in AlphabetChars)

HsAlphabet.Add(c);

}

public char GetChar(char Char)

{

Char = System.Char.ToLower(Char);

Char = Replaces.ContainsKey(Char) ? Replaces[Char] : Char;

return HsAlphabet.Contains(Char) ? Char : Replacer;

}

}

public class PlayFairRu56 : PlayFairSettings

{

public PlayFairRu56()

: base()

{

Replaces.Add('ё', 'е');

Replaces.Add('й', 'и');

Replaces.Add('ъ', 'ь');

}

protected override char[] AlphabetChars { get { return "абвгдежзиклмнопрстуфхцчшщьыэюя".ToCharArray(); } }

public override char Replacer { get { return 'х'; } }

public override int Columns { get { return 5; } }

public override int Rows { get { return 6; } }

}

public sealed class PlayFair

{

struct TablePosition

{

public int Row;

public int Column;

public TablePosition(int Row, int Column)

{

this.Row = Row;

this.Column = Column;

}

}

char[,] Matrix;

Dictionary<char, TablePosition> Positions;

public PlayFairSettings Settings { get; private set; }

public string Key { get; private set; }

public PlayFair(PlayFairSettings Settings, string Key)

{

this.Settings = Settings;

this.Key = Key;

Positions = new Dictionary<char, TablePosition>();

var items = MatrixItems().GetEnumerator();

Matrix = new char[Settings.Rows, Settings.Columns];

for (int r = 0; r < Settings.Rows; r++)

{

for (int c = 0; c < Settings.Columns; c++)

{

if (items.MoveNext())

{

Matrix[r, c] = items.Current;

Positions.Add(items.Current, new TablePosition(r, c));

}

else throw new ArgumentException("Алфавит слишком маленький");

}

}

}

IEnumerable<char> MatrixItems()

{

HashSet<char> used = new HashSet<char>();

foreach (char c in Key)

{

char rc = Settings.GetChar(c);

if (!used.Contains(rc))

{

used.Add(rc);

yield return rc;

}

}

foreach (char c in Settings.Alphabet)

{

if (!used.Contains(c))

{

used.Add(c);

yield return c;

}

}

}

public IEnumerable<char> Bigrams(string Text)

{

char prev = '\0';

bool even = false;

foreach (char c in Text)

{

char rc = Settings.GetChar(c);

if (!even)

{

prev = rc;

even = true;

}

else

{

if (prev == rc)

{

if (prev != Settings.Replacer)

{

yield return prev;

yield return Settings.Replacer;

prev = rc;

}

}

else

{

yield return prev;

yield return rc;

even = false;

}

}

}

if (even && prev != Settings.Replacer)

{

yield return prev;

yield return Settings.Replacer;

}

}

public string Crypt(string Text, bool ModeCrypt = true)

{

int shift = ModeCrypt ? 1 : -1;

StringBuilder sb = new StringBuilder();

var chars = Bigrams(Text).GetEnumerator();

while (chars.MoveNext())

{

var p1 = Positions[chars.Current];

chars.MoveNext();

var p2 = Positions[chars.Current];

int error = 0;

if (p1.Column == p2.Column)

{

p1.Row = Mod(p1.Row + shift, Settings.Rows);

p2.Row = Mod(p2.Row + shift, Settings.Rows);

error++;

}

else if (p1.Row == p2.Row)

{

p1.Column = Mod(p1.Column + shift, Settings.Columns);

p2.Column = Mod(p2.Column + shift, Settings.Columns);

error++;

}

if (error == 2)

throw new ArgumentException("Неверные биграммы");

sb.Append(Matrix[p1.Row, p2.Column]);

sb.Append(Matrix[p2.Row, p1.Column]);

}

return sb.ToString();

}

private int Mod(int x, int m)

{

return (x % m + m) % m;

}

public override string ToString()

{

StringBuilder sb = new StringBuilder();

sb.Append("Ключ: ");

sb.AppendLine(Key);

for (int r = 0; r < Settings.Rows; r++)

{

for (int c = 0; c < Settings.Columns; c++)

{

sb.Append(Matrix[r, c]);

sb.Append(' ');

}

sb.AppendLine();

}

return sb.ToString();

}

}

БА НК УЗ ОВ МИ ХА ИЛ ОЛ ЕГ ОВ ИЧ

ИА ЖУ ЁЗ ЦВ ХЫ ХЗ ИЕ БЛ МГ ЗК ЫЧ

1. Расшифровать следующие сообщения:

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
|  | **съчпщг окхчхиге ичлкпщг** | Расшифровать с помощью шифра Цезаря. Ключ 7 |

курить здоровью вредить

class Program

{

static void Main()

{

var cipher1 = new CaesarCipher();

Console.Write("Введите текст: ");

var message1 = Console.ReadLine();

Console.Write("Введите ключ: ");

var secretKey = Convert.ToInt32(Console.ReadLine());

var encryptedText = cipher1.Encrypt(message1, secretKey);

Console.WriteLine("Зашифрованное сообщение: {0}", encryptedText);

Console.WriteLine("Расшифрованное сообщение: {0}", cipher1.Decrypt(encryptedText, secretKey));

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введите ключ: ");

string keyword = Console.ReadLine();

string additionalChars = "БВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

string fullKeyword = keyword + additionalChars;

TrisemusCipher trisemus = new TrisemusCipher(fullKeyword);

Console.WriteLine("Введите текст для шифрования: ");

string plaintext = Console.ReadLine();

string ciphertext = trisemus.Encrypt(plaintext);

Console.WriteLine("Зашифрованный текст: " + ciphertext);

Console.WriteLine("-------------------------------------");

PlayFairSettings ps = new PlayFairRu56();

PlayFair pf = new PlayFair(ps, "Защита");

Console.WriteLine(pf);

string Text = "БанкузовМихаилОлегович";

Console.WriteLine("Исходный текст\r\n" + Text);

int i = 0;

foreach (char c in pf.Bigrams(Text))

{

Console.Write(c);

i++;

if (i % 2 == 0) Console.Write(' ');

if (i % 10 == 0) Console.WriteLine();

}

Text = pf.Crypt(Text, true);

Console.WriteLine("\r\nЗашифрованный текст\r\n" + Text);

Text = pf.Crypt(Text, false);

Console.WriteLine("\r\nРасшифрованный текст\r\n" + Text);

Console.WriteLine("-------------------------------------");

var cipher = new VigenereCipher("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ");

Console.Write("Введите текст: ");

var inputText = Console.ReadLine().ToUpper();

Console.Write("Введите ключ: ");

var password = Console.ReadLine().ToUpper();

var encryptedText4 = cipher.Encrypt(inputText, password);

Console.WriteLine("Зашифрованное сообщение: {0}", encryptedText4);

Console.WriteLine("Расшифрованное сообщение: {0}", cipher.Decrypt(encryptedText4, password));

Console.WriteLine("-------------------------------------");

string encryptedMessage = "съчпщг окхчхиге ичлкпщг";

int key1 = 7;

CaesarCipher cipher2 = new CaesarCipher();

string decryptedMessage = cipher2.Decrypt(encryptedMessage, key1);

Console.WriteLine("Расшифрованное сообщение: " + decryptedMessage);

}

}

