**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

кафедра програмних засобів

# ЗВІТ

з лабораторної роботи № 1

з дисципліни «Безпека та захист програм і даних» на тему:

# «БАЗОВІ ШИФРИ. ЧАСТОТНИЙ КРИПТОАНАЛІЗ»

Виконав:

ст. гр. КНТ-113сп Іван ЩЕДРОВСЬКИЙ

Прийняв:

доцент Тетяна ЗАЙКО

2025

## 1 Мета роботи

Ознайомитись з базовими шифрами. Розглянути методику частотного аналізу

## 2 Завдання до лабораторної роботи

2.1 Розробити програми шифрування та дешифрування квадратом Полібія

2.2 Виконати частотний криптоаналіз отриманих зашифрованих текстів

## 3 Обране повідомлення для шифрування

В якості повідомлення для шифрування було обрано абзац з Wikipedia про криптографію:

"Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in actual practice by any adversary. While it is theoretically possible to break into a well-designed system, it is infeasible in actual practice to do so. Such schemes, if well designed, are therefore termed \"computationally secure\". Theoretical advances (e.g., improvements in integer factorization algorithms) and faster computing technology require these designs to be continually reevaluated and, if necessary, adapted. Information-theoretically secure schemes that provably cannot be broken even with unlimited computing power, such as the one-time pad, are much more difficult to use in practice than the best theoretically breakable but computationally secure schemes."

## 4 Обраний ключ

В якості ключа було обрано слово «Modern». Воно не має повторів літер в ньому, а це важливо при використанні квадрату Полібія

## 5 Зашифроване повідомлення

Для шифрування та дешифрування було використано квадрат Полібія розміром 5 на 5. Це пов’язано з тим, що при шифруванні використовується тільки латинський алфавіт який складається з 26 літер. Під час шифрування та дешифрування латинські літери “J” та “I” вважаються як однакові.

Програма має два основних режими роботи, які по різному змінюють початкові дані. Перший режим видаляє всі числа, пробіли та знаки пунктуації

В текстовому вигляді зашифроване повідомлення виглядає ось так:

«NABCFGKFEVYAPFHVQESXQCHMSUEIHXCBAGNHYQCNHYSKHUYQCAFEHGBKANVZYCFXKSCG
KCVFHKYSKCKFEVYAPFHVQSKHUPAFSYQNXHFCBCXSPGCBHFAZGBKANVZYHYSAGHUQHFBGCXXHXXZ
NVYSAGXNHTSGPXZKQHUPAFSYQNXQHFBYAIFCHTSGHKYZHUVFHKYSKCIEHGEHBMCFXHFEOQSUCSYS
XYQCAFCYSKHUUEVAXXSIUCYAIFCHTSGYAHOCUUBCXSPGCBXEXYCNSYSXSGLCHXSIUCSGHKYZHUVF
HKYSKCYABAXAXZKQXKQCNCXSLOCUUBCXSPGCBHFCYQCFCLAFCYCFNCBKANVZYHYSAGHUUEXCKZF
CYQCAFCYSKHUHBMHGKCXCPSNVFAMCNCGYXSGSGYCPCFLHKYAFSRHYSAGHUPAFSYQNXHGBLHXYCF
KANVZYSGPYCKQGAUAPEFCWZSFCYQCXCBCXSPGXYAICKAGYSGZHUUEFCCMHUZHYCBHGBSLGCKCXX
HFEHBHVYCBSGLAFNHYSAGYQCAFCYSKHUUEXCKZFCXKQCNCXYQHYVFAMHIUEKHGGAYICIFATCGCM
CGOSYQZGUSNSYCBKANVZYSGPVAOCFXZKQHXYQCAGCYSNCVHBHFCNZKQNAFCBSLLSKZUYYAZXCSG
VFHKYSKCYQHGYQCICXYYQCAFCYSKHUUEIFCHTHIUCIZYKANVZYHYSAGHUUEXCKZFCXKQCNCX»

А також є другий режим в якому залишаються всі числа, всі пробіли та знаки пунктуації. Цей режим потрібен для простішої демонстрації роботи програми та простішої демонстрації частотного криптоаналізу.

В текстового вигляді зашифроване повідомлення виглядає ось так:

«NABCFG KFEVYAPFHVQE SX QCHMSUE IHXCB AG NHYQCNHYSKHU YQCAFE HGB KANVZYCF
XKSCGKC VFHKYSKC; KFEVYAPFHVQSK HUPAFSYQNX HFC BCXSPGCB HFAZGB KANVZYHYSAGHU
QHFBGCXX HXXZNVYSAGX, NHTSGP XZKQ HUPAFSYQNX QHFB YA IFCHT SG HKYZHU VFHKYSKC IE HGE
HBMCFXHFE. OQSUC SY SX YQCAFCYSKHUUE VAXXSIUC YA IFCHT SGYA H OCUU-BCXSPGCB XEXYCN, SY
SX SGLCHXSIUC SG HKYZHU VFHKYSKC YA BA XA. XZKQ XKQCNCX, SL OCUU BCXSPGCB, HFC YQCFCLAFC
YCFNCB "KANVZYHYSAGHUUE XCKZFC". YQCAFCYSKHU HBMHGKCX (C.P., SNVFAMCNCGYX SG SGYCPCF

LHKYAFSRHYSAG HUPAFSYQNX) HGB LHXYCF KANVZYSGP YCKQGAUAPE FCWZSFC YQCXC BCXSPGX YA IC KAGYSGZHUUE FCCMHUZHYCB HGB, SL GCKCXXHFE, HBHVYCB. SGLAFNHYSAG-YQCAFCYSKHUUE XCKZFC XKQCNCX YQHY VFAMHIUE KHGGAY IC IFATCG CMCG OSYQ ZGUSNSYCB KANVZYSGP VAOCF, XZKQ HX YQC AGC-YSNC VHB, HFC NZKQ NAFC BSLLSKZUY YA ZXC SG VFHKYSKC YQHG YQC ICXY YQCAFCYSKHUUE IFCHTHIUC IZY KANVZYHYSAGHUUE XCKZFC XKQCNCX.»

Демонстрація роботи програми показана на рисунку 1.

```
C:\home\university-4\security>go run .
LB 1
Source text: Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms h
ard to break in actual practice by any adversary. While it is theoretically possible to break into a well-designed system, it is infeasible in actual practice to do so. Such schemes, if well designed, are ther
efore termed "computationally secure". Theoretical advances (e.g., improvements in integer factorization algorithms) and faster computing technology require these designs to be continually reevaluated and, if
necessary, adapted. Information-theoretically secure schemes that provably cannot be broken even with unlimited computing power, such as the one-time pad, are much more difficult to use in practice than the be
st theoretically breakable but computationally secure schemes.
Source code: MODERN

[EN] Prepared text: MODERN CRYPTOGRAPHY IS HEAVILY BASED ON MATHEMATICAL THEORY AND COMPUTER SCIENCE PRACTICE; CRYPTOGRAPHIC ALGORITHMS ARE DESIGNED AROUND COMPUTATIONAL HARDNESS ASSUMPTIONS, MAKING SUCH ALGOR
ITHMS HARD TO BREAK IN ACTUAL PRACTICE BY ANY ADVERSARY. WHILE IT IS THEORETICALLY POSSIBLE TO BREAK INTO A WELL-DESIGNED SYSTEM, IT IS INFEASIBLE IN ACTUAL PRACTICE TO DO SO. SUCH SCHEMES, IF WELL DESIGNED, A
RE THEREFORE TERMED "COMPUTATIONALLY SECURE". THEORETICAL ADVANCES (E.G., IMPROVEMENTS IN INTEGER FACTORIZATION ALGORITHMS) AND FASTER COMPUTING TECHNOLOGY REQUIRE THESE DESIGNS TO BE CONTINUALLY REEVALUATED A
ND, IF NECESSARY, ADAPTED. INFORMATION-THEORETICALLY SECURE SCHEMES THAT PROVABLY CANNOT BE BROKEN EVEN WITH UNLIMITED COMPUTING POWER, SUCH AS THE ONE-TIME PAD, ARE MUCH MORE DIFFICULT TO USE IN PRACTICE THAN
 THE BEST THEORETICALLY BREAKABLE BUT COMPUTATIONALLY SECURE SCHEMES.
[EN] Prepared code: MODERN
Polybius square:
M O D E R
N A B C F
G H I K L
P Q S T U
V W X Y Z
Encoded text: NABCFG KFEVYAPFHVQE SX QCHMSUE IHXCB AG NHYQCNHYSKHU YQCAFE HGB KANVZYCF XKSCGKC VFHKYSKC; KFEVYAPFHVQSK HUPAFSYQNX HFC BCXSPGCB HFAZGB KANVZYHYSAGHU QHFBGCXX HXXZNVYSAGX, NHTSGP XZKQ HUPAFSYQNX
QHFB YA IFCHT SG HKYZHU VFHKYSKC IE HGE HBMCFXHFE. OQSUC SY SX YQCAFCYSKHUUE VAXXSIUC YA IFCHT SGYA H OCUU-BCXSPGCB XEXYCN, SY SX SGLCHXSIUC SG HKYZHU VFHKYSKC YA BA XA. XZKQ XKQCNCX, SL OCUU BCXSPGCB, HFC YQC
FCLAFC YCFNCB "KANVZYHYSAGHUUE XCKZFC". YQCAFCYSKHU HBMHGKCX (C.P., SNVFAMCNCGYX SG SGYCPCF LHKYAFSRHYSAG HUPAFSYQNX) HGB LHXYCF KANVZYSGP YCKQGAUAPE FCWZSFC YQCXC BCXSPGX YA IC KAGYSGZHUUE FCCMHUZHYCB HGB, SL
 GCKCXXHFE, HBHVYCB. SGLAFNHYSAG-YQCAFCYSKHUUE XCKZFC XKQCNCX YQHY VFAMHIUE KHGGAY IC IFATCG CMCG OSYQ ZGUSNSYCB KANVZYSGP VAOCF, XZKQ HX YQC AGC-YSNC VHB, HFC NZKQ NAFC BSLLSKZUY YA ZXC SG VFHKYSKC YQHG YQC I
CXY YQCAFCYSKHUUE IFCHTHIUC IZY KANVZYHYSAGHUUE XCKZFC XKQCNCX.

[DE] Data to decode: NABCFG KFEVYAPFHVQE SX QCHMSUE IHXCB AG NHYQCNHYSKHU YQCAFE HGB KANVZYCF XKSCGKC VFHKYSKC; KFEVYAPFHVQSK HUPAFSYQNX HFC BCXSPGCB HFAZGB KANVZYHYSAGHU QHFBGCXX HXXZNVYSAGX, NHTSGP XZKQ HUPA
FSYQNX QHFB YA IFCHT SG HKYZHU VFHKYSKC IE HGE HBMCFXHFE. OQSUC SY SX YQCAFCYSKHUUE VAXXSIUC YA IFCHT SGYA H OCUU-BCXSPGCB XEXYCN, SY SX SGLCHXSIUC SG HKYZHU VFHKYSKC YA BA XA. XZKQ XKQCNCX, SL OCUU BCXSPGCB,
HFC YQCFCLAFC YCFNCB "KANVZYHYSAGHUUE XCKZFC". YQCAFCYSKHU HBMHGKCX (C.P., SNVFAMCNCGYX SG SGYCPCF LHKYAFSRHYSAG HUPAFSYQNX) HGB LHXYCF KANVZYSGP YCKQGAUAPE FCWZSFC YQCXC BCXSPGX YA IC KAGYSGZHUUE FCCMHUZHYCB
HGB, SL GCKCXXHFE, HBHVYCB. SGLAFNHYSAG-YQCAFCYSKHUUE XCKZFC XKQCNCX YQHY VFAMHIUE KHGGAY IC IFATCG CMCG OSYQ ZGUSNSYCB KANVZYSGP VAOCF, XZKQ HX YQC AGC-YSNC VHB, HFC NZKQ NAFC BSLLSKZUY YA ZXC SG VFHKYSKC YQH
G YQC ICXY YQCAFCYSKHUUE IFCHTHIUC IZY KANVZYHYSAGHUUE XCKZFC XKQCNCX.
[DE] Prepared code: MODERN
Decoded text: MODERN CRYPTOGRAPHY IS HEAVILY BASED ON MATHEMATICAL THEORY AND COMPUTER SCIENCE PRACTICE; CRYPTOGRAPHIC ALGORITHMS ARE DESIGNED AROUND COMPUTATIONAL HARDNESS ASSUMPTIONS, MAKING SUCH ALGORITHMS
HARD TO BREAK IN ACTUAL PRACTICE BY ANY ADVERSARY. WHILE IT IS THEORETICALLY POSSIBLE TO BREAK INTO A WELL-DESIGNED SYSTEM, IT IS INFEASIBLE IN ACTUAL PRACTICE TO DO SO. SUCH SCHEMES, IF WELL DESIGNED, ARE THE
REFORE TERMED "COMPUTATIONALLY SECURE". THEORETICAL ADVANCES (E.G., IMPROVEMENTS IN INTEGER FACTORIZATION ALGORITHMS) AND FASTER COMPUTING TECHNOLOGY REQUIRE THESE DESIGNS TO BE CONTINUALLY REEVALUATED AND, IF
 NECESSARY, ADAPTED. INFORMATION-THEORETICALLY SECURE SCHEMES THAT PROVABLY CANNOT BE BROKEN EVEN WITH UNLIMITED COMPUTING POWER, SUCH AS THE ONE-TIME PAD, ARE MUCH MORE DIFFICULT TO USE IN PRACTICE THAN THE B
EST THEORETICALLY BREAKABLE BUT COMPUTATIONALLY SECURE SCHEMES.
```

Рисунок 1 – Демонстація роботи програми

## 6 Розшифроване повідомлення

Розшифроване повідомлення виглядає ось так без пунктації:

«MODERNCRYPTOGRAPHYISHEAVILYBASEDONMATHEMATICALTHEORYANDCOMPUTERSCIENCEPRACTICECRYPTOGRAPHICALGORITHMSAREDESIGNEDAROUNDCOMPUTATIONALHARDNESSASSUMPTIONSMAKINGSUCHALGORITHMSHARDTOBREAKINACTUALPRACTICEBYANYADVERSARYWHILEITISTHEORETICALLYPOSSIBLETOBREAKINTOAWELLDESIGNEDSYSTEMITISINFEASIBLEINACTUALPRACTICETODOSOSUCHSCHEMESIFWELLDESIGNEDARETHEREFORETERMEDCOMPUTATIONALLYSECURETHEORETICALADVANCESEGIMPROVEMENTSININTEGERFACTORIZATIONALGORITHMSANDFASTERCOMPUTINGTECHNOLOGYREQUIRETHESEDESIGNSTOBECONTINUALLYREEVALUATEDANDIFNECESSARYADAPTEDINFORMATIONTHEORETICALLYSECURESCHEMESTHATPROVABLYCANNOTBEBROKENEVENWITHUNLIMITEDCOMPUTINGPOWERSUCHASTHEONETIMEPADAREMUCHMOREDIFFICULTTOUSEINPRACTICETHANTHEBESTTHEORETICALLYBREAKABLEBUTCOMPUTATIONALLYSECURESCHEMES»

Та ось так з пунктуацією:

«MODERN CRYPTOGRAPHY IS HEAVILY BASED ON MATHEMATICAL THEORY AND COMPUTER SCIENCE PRACTICE; CRYPTOGRAPHIC ALGORITHMS ARE DESIGNED AROUND COMPUTATIONAL HARDNESS ASSUMPTIONS, MAKING SUCH ALGORITHMS HARD TO BREAK IN ACTUAL PRACTICE BY ANY ADVERSARY.

WHILE IT IS THEORETICALLY POSSIBLE TO BREAK INTO A WELL-DESIGNED SYSTEM, IT IS INFEASIBLE IN ACTUAL PRACTICE TO DO SO. SUCH SCHEMES, IF WELL DESIGNED, ARE THEREFORE TERMED "COMPUTATIONALLY SECURE". THEORETICAL ADVANCES (E.G., IMPROVEMENTS IN INTEGER FACTORIZATION ALGORITHMS) AND FASTER COMPUTING TECHNOLOGY REQUIRE THESE DESIGNS TO BE CONTINUALLY REEVALUATED AND, IF NECESSARY, ADAPTED. INFORMATION-THEORETICALLY SECURE SCHEMES THAT PROVABLY CANNOT BE BROKEN EVEN WITH UNLIMITED COMPUTING POWER, SUCH AS THE ONE-TIME PAD, ARE MUCH MORE DIFFICULT TO USE IN PRACTICE THAN THE BEST THEORETICALLY BREAKABLE BUT COMPUTATIONALLY SECURE SCHEMES.»

# 7 Результати проведення частотного криптоаналізу

Для виконання частотного аналізу було використано онлайн ресурс «https://alexandernwilson.com/gallery/frequency_analysis.html», який дозволяє зручно підбирати букви шифру до реальних.

Також було використано таблицю з загальною частотою літер в англійській мові: «Relative Frequencies of Letters in General English Plain text From *Cryptographical Mathematics*, by Robert Edward Lewand»

І для проведення більш швидкого та реального аналізу було взято текст з пробілами та пунктуацією. Приклад як виглядає сайт без заповнених літер показаний на рисунку 2.

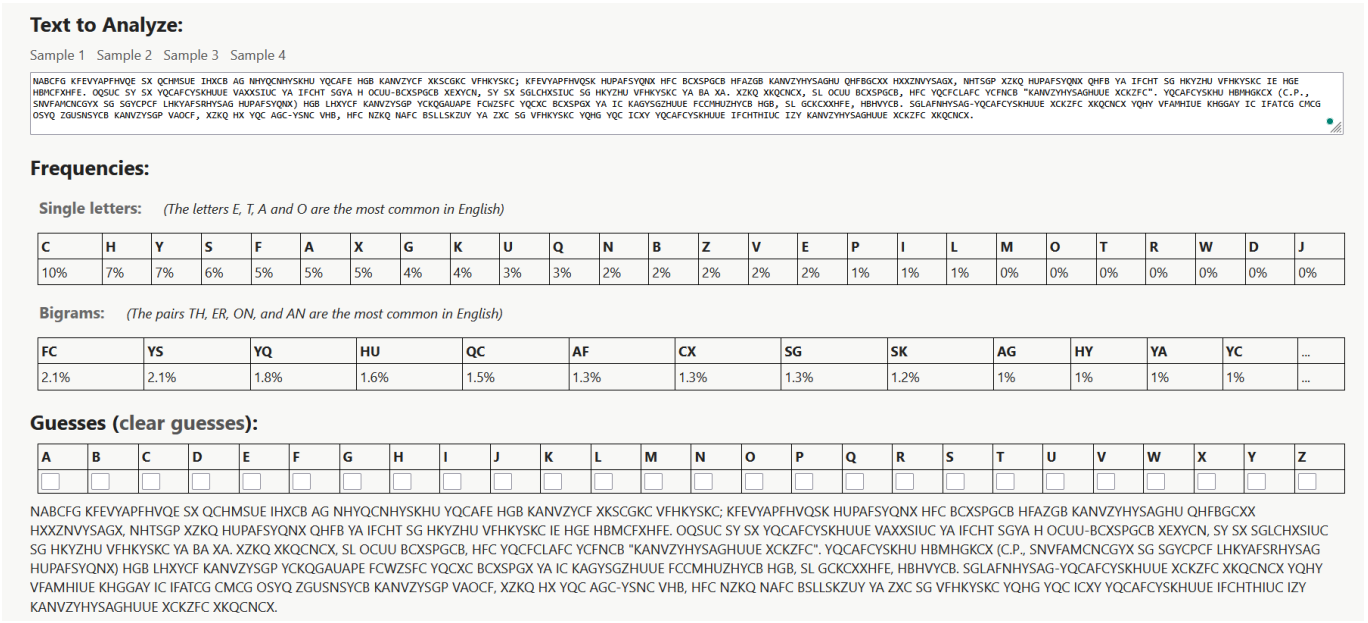


Рисунок 2 – Початок частотного аналізу

Для початку, оскільки це англійський текст, то тут є артиклі a, an та the. Відразу можна побачити одну букву H, яка стоїть перед словом. Скоріше за все це як раз і є “a”, а також буква “H” зустрічається дуже багато разів в тексті

Найбільш популярна буква в тексті це “C”, і згідно частоти це скоріше за все “e”. Після встановлення цих двох літер вигляд тексту показаний на рисунку 3

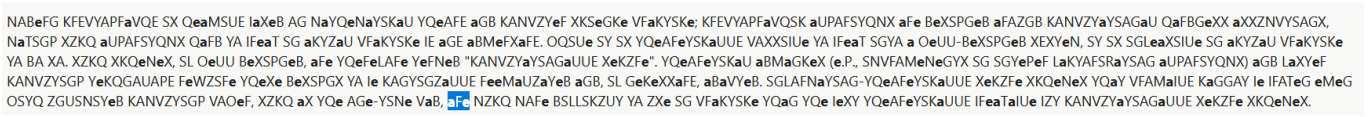
NABeFG KFEVYAPFaVQE SX QeaMSUE laXeB AG NaYQeNaYSKaU YQeAFE aGB KANVZYeF XKSeGKe VFaKYSKe; KFEVYAPFaVQSK aUPAFSYQNX aFe BeXSPGeB aFAZGB KANVZYaYSAGaU QaFBGeXX aXXZNVYSAGX, NaTSGP XZKQ aUPAFSYQNX QaFB YA IFeaT SG aKYZaU VFaKYSKe IE aGE aBMeFXaFE. OQSUe SY SX YQeAFeYSKaUUE VAXXSIUe YA IFeaT SGYA a OeUU-BeXSPGeB XEXYeN, SY SX SGLeaXSIUe SG aKYZaU VFaKYSKe YA BA XA. XZKQ XKQeNeX, SL OeUU BeXSPGeB, aFe YQeFeLAFe YeFNeB "KANVZYaYSAGaUUE XeKZFe". YQeAFeYSKaU aBMaGKeX (e.P., SNVFAMeNeGYX SG SGYePeF LaKYAFSRaYSAG aUPAFSYQNX) aGB LaXYeF KANVZYSGP YeKQGAUAPE FeWZSFe YQeXe BeXSPGX YA le KAGYSGZaUUE FeeMaUZaYeB aGB, SL GeKeXXaFE, aBaVYeB. SGLAFNaYSAG-YQeAFeYSKaUUE XeKZFe XKQeNeX YQaY VFAMalUE KaGGAY le IFATeG eMeG OSYQ ZGUSNSYeB KANVZYSGP VAOeF, XZKQ aX YQe AGe-YSNe VaB, aFe NZKQ NAFe BSLLSKZUY YA ZXe SG VFaKYSKe YQaG YQe leXY YQeAFeYSKaUUE IFeaTalUe IZY KANVZYaYSAGaUUE XeKZFe XKQeNeX.

Рисунок 3 – Заміна букв e та a

Як ми бачимо, є конструкція “aFe”, яка скоріше за все є “are”. Тому літера “F” це “r”

Із маленьких слів залишається “SL”, “SY”, “YA”, “SX”, “SG”, “AG”, “aGB”. Трошки погравшись з текстом мені здається, що “AG” це “on”. І тепер “aGB” це “anB” і скоріше за все “B” це як раз “d”

Проміжний єтап показаний на рисунку 4


Nodern KrEVYoPraVQE SX QeaMSUE laXed on NaYQeNaYSKaU YQeorE and KoNVZYer XKSenKe VraKYSKe; KrEVYoPraVQSK aUPorSYQNX are deXSPned aroZnd KoNVZYaYSonaU QardneXX aXXZNVYSonX, NaTSnP XZKQ aUPorSYQNX Qard Yo IreaT Sn aKYZaU VraKYSKe IE anE adMerXarE. OQSUe SY SX YQeoreYSKaUUE VoXXSIUe Yo IreaT SnYo a OeUU-deXSPned XEXYeN, SY SX SnLeaXSIUe Sn aKYZaU VraKYSKe Yo do Xo. XZKQ XKQeNeX, SL OeUU deXSPned, are YQereLore YerNed "KoNVZYaYSonaUUE XeKZre". YQeoreYSKaU adManKeX (e.P., SNVroMeNenYX Sn SnYePer LaKYorSRaYSon aUPorSYQNX) and LaXYer KoNVZYSnP YeKQnoUoPE reWZSre YQeXe deXSPnX Yo le KonYSnZaUUE reeMaUZaYed and, SL neKeXXarE, adaVYed. SnLorNaYSon-YQeoreYSKaUUE XeKZre XKQeNeX YQaY VroMalUE KannoY le IroTen eMen OSYQ ZnUSNSYed KoNVZYSnP VoOer, XZKQ aX YQe one-YSNe Vad, are NZKQ Nore dSLLSKZUY Yo ZXe Sn VraKYSKe YQan YQe leXY YQeoreYSKaUUE IreaTalUe IZY KoNVZYaYSonaUUE XeKZre XKQeNeX.

Рисунок 4 – Заміна літер r, o, n та d

Слово «Nodern» майже гарантовано є “modern”. Далі дивлячись на частоту літер можна припустити, що “Y” це “T”, а також через це складається багато слів, наприклад “to”. Далі дивлячись на частоту літер можна сказати, що “S” це “I”.
На цьому етапі вже є багато різних слів, це показано на рисунку 5

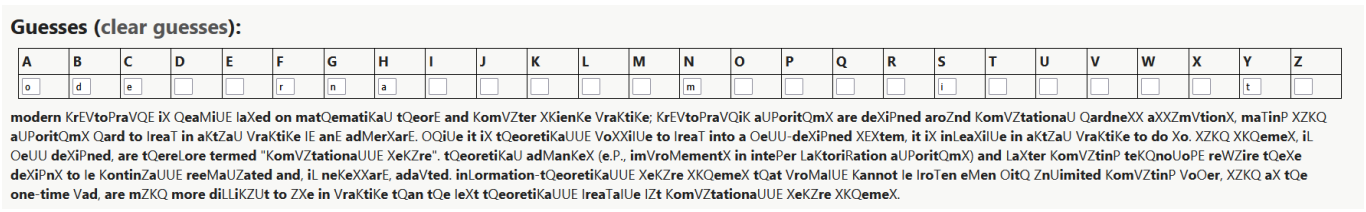


Рисунок 5 – Заміна літер t та i

Дивлячись на «matQematiKaU» можна сказати, що “Q” це “H”. Дивлячись на «iX» та «deXiPned» можна сказати, що “X” це “S”, а також що “P” це “G”. Слово «imVroMements» скоріше за все означає «improvements», а тому “V” це “P”, а “M” це “V”. З “Kannot” буква “K” це “C”. З “crEptographE” буква “E” це “Y”. З “heaviUy” буква “U” це “L”. І на цьому моменті вже можна прочитати повідомлення, це показано на рисунку 6

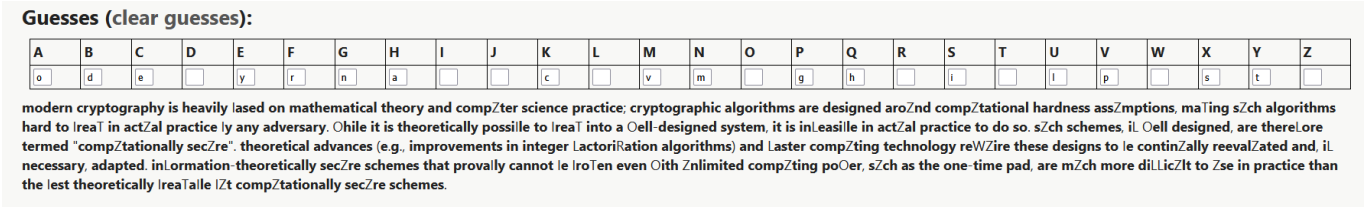


Рисунок 6 – Передостання заміна букв

Слово “aroZnd” це “around”, а тому “Z” це “U“. З слів «heavily Iased on» буква “I” це “b”. З “Ohile” буква “O” це “W”. Зі слів “cannot be broTen even” буква “T” це “k”.

З “inLormation” буква “L” це “F”. В “factoriRation” буква “R” це “Z”. Та в “reWuire” буква “W” це “q”. Повідомлення розкодовано. Це показано на рисунку 7. Також знайдені букви можна порівняти з квадратом, який показаний на рисунку 8

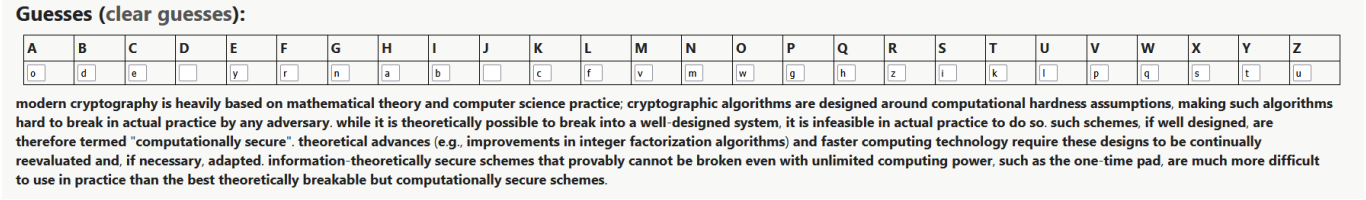


Рисунок 7 – Розшифрований текст

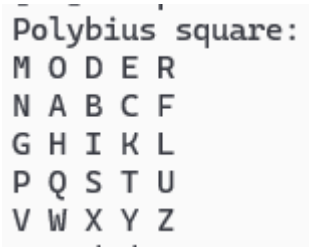


Рисунок 8 – Оригінальний квадрат Полібія

# 8 Текст розробленої програми

Програма була написана на мові програмування Golang

```
package lb1

import (
	"errors"
	"fmt"
	"regexp"
	"strings"
)

const WITH_PUNCTIATION_AND_NUMBERS = false

const WIDTH = 5
const HEIGHT = 5

type polybiusSquare struct {
	square  [][]rune
	charMap map[rune][2]int
}

// Base preparation, like remove punctuation and change all letters to uppercase
func prepareData(text string) string {
	if WITH_PUNCTIATION_AND_NUMBERS {
		return strings.ToUpper(text)
	}

	removePunctiation := regexp.MustCompile(`([^\w])|([\s\d])`)
	withoutPunctiation := removePunctiation.ReplaceAllLiteralString(text, "")

	return strings.ToUpper(withoutPunctiation)
}

func (square polybiusSquare) print() {
	fmt.Println("Polybius square:")

	for i, _ := range square.square {
		for j, _ := range square.square[i] {
```

```
                        value := square.square[i][j]

                        if value == 0 {
                                value = '-'
                        }

                        fmt.Print(string(value) + " ")
                }

                fmt.Print("\n")
        }
}

func isCodeValid(code string) bool {
        cache := make(map[rune]bool)

        for _, r := range code {
                _, ok := cache[r]

                if ok {
                        return false
                }

                cache[r] = true
        }

        return true
}

// Generate square for latin alphabet
// Default size if 5x5, where I should be equal to J
func (polybiusSquare *polybiusSquare) init(code string) {
        square := make([][]rune, HEIGHT)

        for i := range square {
                square[i] = make([]rune, WIDTH)
        }

        charMap := make(map[rune][2]int)
        polybiusSquare.square = square
        polybiusSquare.charMap = charMap

        cache := make(map[rune]bool)
        cache['J'] = true

        for row := range HEIGHT {
                for col := range WIDTH {
                        index := row*WIDTH + col

                        if index >= len(code) {
                                break
                        }

                        char := rune(code[index])

                        if char == 'J' {
                                char = 'I'
                        }
```

```
                        square[row][col] = char
                        charMap[char] = [2]int{row, col}
                        cache[char] = true
                }
        }

        queue := make([]rune, 0)

        for char := 'A'; char <= 'Z'; char++ {
                if _, ok := cache[char]; ok {
                        continue
                }

                queue = append(queue, char)
        }

        for row := range HEIGHT {
                for col := range WIDTH {
                        if square[row][col] != 0 {
                                continue
                        }

                        char := queue[0]
                        queue = queue[1:]
                        square[row][col] = char
                        charMap[char] = [2]int{row, col}
                }
        }
}

func (square polybiusSquare) encode(char rune) (string, error) {
        pos, ok := square.charMap[char]

        if !ok {
                if !WITH_PUNCTIATION_AND_NUMBERS {
                        return "", errors.New("Char not found! Char: " + string(char))
                }

                return string(char), nil
        }

        newRow := pos[0] + 1

        if newRow > HEIGHT-1 {
                newRow = 0
        }

        return string(square.square[newRow][pos[1]]), nil
}

func (square polybiusSquare) decode(char rune) (string, error) {
        pos, ok := square.charMap[char]

        if !ok {
                if !WITH_PUNCTIATION_AND_NUMBERS {
                        return "", errors.New("Char not found! Char: " + string(char))
                }

                return string(char), nil
```

```
	}

	newRow := pos[0] - 1

	if newRow < 0 {
		newRow = HEIGHT - 1
	}

	return string(square.square[newRow][pos[1]]), nil
}

func encodePolybiusSquare(sourceData string, sourceCode string) (string, error) {
	if !isCodeValid(sourceCode) {
		return "", errors.New("Code is not valid")
	}

	data := prepareData(sourceData)
	code := prepareData(sourceCode)
	fmt.Println("[EN] Prepared text:", data)
	fmt.Println("[EN] Prepared code:", code)

	square := polybiusSquare{}
	square.init(code)
	square.print()

	encodedData := ""

	for _, char := range data {
		encodedChar, err := square.encode(char)

		if err != nil {
			return "", err
		}

		encodedData += encodedChar
	}

	return encodedData, nil
}

func decodePolybiusSquare(encodedData string, sourceCode string) (string, error) {
	if !isCodeValid(sourceCode) {
		return "", errors.New("Code is not valid")
	}

	code := prepareData(sourceCode)

	fmt.Println("[DE] Data to decode:", encodedData)
	fmt.Println("[DE] Prepared code:", code)
	square := polybiusSquare{}
	square.init(code)
	// square.print()

	decodedData := ""

	for _, char := range encodedData {
		encodedChar, err := square.decode(char)

		if err != nil {
```

```
                return "", err
            }

            decodedData += encodedChar
        }

        return decodedData, nil
}

func Run() {
        println("LB 1")

        sourceText := "Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in actual practice by any adversary. While it is theoretically possible to break into a well-designed system, it is infeasible in actual practice to do so. Such schemes, if well designed, are therefore termed \"computationally secure\". Theoretical advances (e.g., improvements in integer factorization algorithms) and faster computing technology require these designs to be continually reevaluated and, if necessary, adapted. Information-theoretically secure schemes that provably cannot be broken even with unlimited computing power, such as the one-time pad, are much more difficult to use in practice than the best theoretically breakable but computationally secure schemes."
        code := "MODERN"

        fmt.Println("Source text:", sourceText)
        fmt.Println("Source code:", code)
        fmt.Println()

        encodedText, err := encodePolybiusSquare(sourceText, code)

        if err != nil {
            fmt.Println("Error in encodePolybiusSquare:", err)
            panic(err)
        }

        fmt.Println("Encoded text:", encodedText)

        fmt.Println()
        decodedText, err := decodePolybiusSquare(encodedText, code)

        if err != nil {
            fmt.Println("Error in decodePolybiusSquare:", err)
            panic(err)
        }

        fmt.Println("Decoded text:", decodedText)
        fmt.Println()
}
```

# 9 Відповіді на контрольні питання

## Опишіть шифр Полібія

В шифрі Полібія ми спочатку будуємо квадрат, або ж прямокутник, але зазвичай квадрат. Далі ми обираємо кодове слово і записуємо його з верхньої лівої клітинки на право. Якщо воно більше, а ніж один рядок, то переносимо на другий. Всі інші клітинки ми заповнюємо літерами алфавіту, яких не було в кодовому слові

Важливо зауважити, що кодове слово повинно мати тільки унікальні літери

Для кодування ми знаходимо позицію оригінальної букви в матриці та беремо елемент, який знаходить на строкі нижче. Якщо це остання строка, то беремо зі строки 0

**Опишіть шифр простої заміни**

Шифр простої заміни це розвиток шифру Цезаря

В шифрі Цезаря спочатку ми виписуємо по порядку всі літери алфавіту в дві строки, а далі здвигаємо другу строку на деяку величину. Таким чином ми отримуємо таблицю відповідності по якій ми будемо кодувати наше повідомлення. Верхній рядок – це оригінальні букви, а нижній – закодовані.

Шифр простої заміни фактично використовує такий самий підхід, але другий рядок є не просто зсувом, а генерується випадковим чином. Тобто його ключем є положення літер в другому рядку. Такий шифр має набагато більше комбінацій. Шифр Цезаря має X комбінацій, де X – кількість літер в алфавіті. А шифр простої заміни має X!, тобто X факторіал варіантів. Для української мови це 2.6313084e+35

**Опишіть шифр Тритемія**

Ширф Тритемія представляє собою велику матрицію, яка має розмір X на X елементів, це X – це кількість елементів в алфавіті

Перший рядок це звичайні літери алфавіту записані по порядку

В другому копіюється перший, але циклічно здвигається на один елемент. В третьому копіюється другий та також циклічно здвигається на один елемент. Так продовжується до самого кінця матриці

Перший рядок є також положенням оригінальних букв.

Перша буква тексту кодується по першому рядку, тобто не змінюється. Друга кодується по другому рядку, тобто береться стовбець з першого рядка та знаходиться елемент який лежить на другому рядку. Всі інші літери кодуються таким самим чином

**Опишіть шифр перестановки**

В шифрі перестановки спочатку обирається ключ після цього генерується випадковим чином матриця перестановки елементів

Наприклад, якщо ключ 5, то оригінальний текст має порядок літер 1 2 3 4 5. Матриця ж елементів має, наприклад, порядок 3 4 2 1 5. Ці числа означають позиції елементів які будуть переставлені всередині тексту, оскільки шифр перестановки не додає нових літер, він тільки буквально переставляє оригінальні

Наприклад, слово «своїх» буде закодовано як «оївсх»

Щоб закодувати текст потрібно прибрати знаки пунктуації, пробіли та зробити всі літери великими, тобто зробити стандартну підготовку даних

Далі весь текст розбивається на блоки, де кожен блок має довжину ключа. Останній блок може доповнитись випадковими літерами, щоб відповідати довжині. В кожному блоці відбувається перестановка літер

**Чи є шифр Полібія шифром простої заміни?**

В шифрі простої заміни використовується випадково згенерований набір букв, а в шифрі Полібія використовується слово та алфавіт по порядку, тому ні, шифр Полібія не є шифром простої заміни.

Але якщо мається на увазі чи шифр Полібія просто заміняє букви на інші відповідні букви, то так, шифр Полібія дійсно це робить.

**Як залежить стійкість шифру від довжини ключа?**

Все залежить від алгоритму який ми використовуємо. Якщо це, наприклад, шифр Полібія, то наш ключ не сильно впливає на стійкість шифру.

При шифрі перестановки довжина ключа має вплив. Наприклад, якщо ми візьмемо ключ з довжиною 2, то це буде дуже просто розкодувати в майбутньому. Але все ж таки буде більш ефективно виконати цей алгоритм в декілька ітерацій, наприклад, спочатку з ключем 5, потім 10, потім ще якимось щоб отримати більш стійкий

Якщо брати більш сучасні алгоритми, то так, довжина ключа робить шифр більш стійким

**Опишіть метод частотного криптоаналізу?**

В алфавіті кількість букв є обмеженою, а також деякі букви зустрічаються набагато частіше, а ніж інші. Наприклад, в латинських словах буква «е» зустрічається найбільше

Таким чином ми можемо спробувати в закодованому тексті знайти букви, які зустрічаються найчастіше та замінити їх на ті, які загалом зустрічаються найбільше.

Також в багатьох мовах є пари букв, або ж навіть трійки, які зустрічаються дуже часто. Наприклад, “the” в англійській мові є досить поширеним. Таким чином аналізуючи текст ми можемо визначити деякі літери, а далі шляхом проб та помилок ми можемо прийти до рішення задачі.

**В яких випадках можна застосовувати метод частотного криптоаналізу?**

Метод частотного криптоаналізу можна застосовувати тільки якщо завжди кожна буква вхідного тексту відповідає іншій букві вихідного. Тобто, якщо буква “A” в одному випадку відповідає букві “B”, а в іншому “C” то використовуючи метод частотного криптоаналізу ми не зможемо зрозуміти початкове повідомлення

## 10 Висновки

Я ознайомився з базовими шифрами. Розглянув методику частотного аналізу