НАЦИОНАЛНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО КОМПЮТЪРНИ ТЕХНОЛОГИИ И СИСТЕМИ
гр.ПРАВЕЦ при ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ДИПЛОМНА РАБОТА

ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ – ЧАСТ ПО ТЕОРИЯ НА ПРОФЕСИЯТА

на

Ученик: Валентин Цветомиров Цонков

ученик от 17107 (XII) клас

Тема: **Проектиране и разработка на приложение за декодиране на шифри**

ПРОФЕСИОНАЛНО НАПРАВЛЕНИЕ: 481 КОМПЮТЪРНИ НАУКИ
ПРОФЕСИЯ: 481020 СИСТЕМЕН ПРОГРАМИСТ
СПЕЦИАЛНОСТ: 4810201 СИСТЕМНО ПРОГРАМИРАНЕ

Ученик: Валентин Цонков

Ръководител: *Момчил Петков*

Правец, 2022

Съдържание

Увод

Днешно време изпращането на информация онлайн е нещо, което всеки един от нас извършва ежедневно. Когато става въпрос за мрежата ние трябва да подсигурим защита за данните които споделяме, защото предлагащата се от наличните приложения не винаги е достатъчна и доста често бива преодоляна неминуемо. Един от вариантите за тази цел е да се запознаем с криптирането и декриптирането на текст или по известно като криптография. Криптографията се прилага в редица области на съвременния живот. От нея зависят сигурността на дебитните карти, на компютърните пароли, на електронната търговия, както и на всякакъв вид информация, която не трябва да достига до външни лица. Приложението което, ще ви представя е шифратор/дешифратор, чрез които ще можете да комуникирате спокойно без да се притеснявате от това дали някой нежелан потребител ще получи вашата информация. Разработено е на “C++” и предлага благоприятни условия за работене с него.

История на шифрите.

Науката за **Тайнопис**, или **Криптография** датира от няколко хиляди години. От най-ранни времена хората се опитват да комуникират тайно било то за да прикият важна информация, да скрият своите военни планове от противниците си или просто тяхната връзка да остане в тайна. В криптографията се изучават принципите, средствата и методите за преобразуване на данни (*криптиране*) с оглед прикриване на тяхната семантика, предотвратяване на неоторизиран достъп или тяхната подмолна промяна и манипулация от трети лица (*Противникът*). В криптоанализа *Противникът* е математически алгоритъм за тестване и за изучаване на пробиви и уязвимости в криптографски алгоритми.

Криптографските принципи, средства и методи се ползват широко и обстойно от съвременната информатика за осигуряване на кибер сигурност, включват поверителност, интегритет на данни, неопровержимост и автентичност.

Криптографията е една от най-старите науки в света и нейната история датира от няколко хиляди години. Първоначално криптографията изучавала начините за шифриране и дешифриране на информация или на еднозначно обратимо преобразуване на прав нешифриран текст на базата на криптографски алгоритъм и ключ в шифрограма прав текст. Тази, така наречена традиционна криптография, обхваща само раздел симетрично криптиране, при което криптирането и декриптирането се извършва на база ползване на един и същ секретен ключ.

Освен този раздел съвременната криптография включва и раздел асиметрично криптиране, чиито криптографски алгоритми се прилагат широко в системите за електронен подпис, хеширане, получаването на скрита информация и квантова криптография – идеята да се създаде квантов суперкомпютър, който да разбива всички видове криптографски кодове в реално време, която идея е по-скоро в сферата на научната фантастика.

Основните класически видове шифриране са:

Преместване, при което буквите на съобщението се разместват.

Например “помогни ми” става “опомнги им” при най-простото преместване.

Замяна, когато буквите или групи от букви по определено правило се заменят с други букви или групи от букви.

Например “fly at once” става “gmz bu podf” при замяна на всяка буква със следващата в азбуката.

Шифър на нихилистите бил използван от революционерите в Русия по време на терора срещу царския режим.

Шифрираните текстове, получавани след прилагане на класическите шифри винаги съдържат статистическа информация за изходния текст, която може да бъде използвана за разбиването им. След разработването на честотния анализ в IX век, практически всички такива шифри стават уязвими за достатъчно квалифициран дешифратор, чак до измислянето на многоазбучния шифър, от Леоне Батиста Алберти, около 1465 година. Неговото нововъведение се състои в това, че използва различни шрифтове за различните части на съобщението. Също така, той изобретява вероятно първата автоматична шифровъчна машина – колело, което осъществявало частична реализация на неговото изобретение. В многоазбучния шифър на Виженер, за шифриране се използва *ключова дума*, която контролира замяната на буквите в зависимост от това, коя буква от ключовата дума е използвана. В средата на 1800-те години Тогава английският математик Чарлз Бабидж разработва тест, който може да открие дължината на ключа на шифъра на Виженер, и тогава разглежда шифъра като сбор на обикновени Цезарови шифри.

В XIX век става окончателно ясно, че секретността на алгоритъма за шифриране не се явява гаранция срещу разбиването му, още повече става ясно, че адекватната криптографска защита трябва да остава защитена дори ако противника познава напълно алгоритъма за шифриране. Секретността на ключа трябва да е достатъчна, за да се осигури криптоустойчивост. Този фундаментален принцип е формулиран за първи път от Август Кирхов, холандски лингвист и криптограф, през 1883 г. Алтернативна формулировка прави Клод Шеннон като *Максимата на Шенон* – „врагът знае нашата система“.

За облекчаване на шифрирането са разработвани различни спомагателни устройства. Едно от най-древните е скиталата, измислена в Древна Гърция, представляваща просто пръчка. С появата на многоазбучния шифър, устройствата започват да се усложняват,

като например диска с шифротекст на Алберти, квадратната дъска(tabula recta) на Йоханес Тритемиус и дисковият шифър на Томас Джеферсън. Най-известна през XX век е роторната машина Енигма, използвана от немците по време на Втората световна война.

Развитието на електрониката и компютърната техника дава силен тласък в развитието на криптографията и криптоанализа. Започва развиването на академични криптографски изследвания (приблизително от 1970-те год) с публикуването на стандарта DES от NBS, в статията на Дифи – Хелман и откриването на алгоритъма RSA. След това акцентът в криптографията се премества от чисто лингвистичните методи към математическите, включително теория на информацията, статистика, комбинаторика, абстрактна алгебра и теория на числата.

Терминология:

Прав некодиран текст – данни (не задължително текстови), предавани без използване на криптография.

Кодиран текст (шифрограма) – данни, получени след използване на криптосистема с указан ключ.

Шифър (Криптосистема) – семейство обратими преобразувания на открит текст в шифрован.

Шифриране (криптиране) – процес на нормално прилагане на криптографско преобразуване на открит текст на основата на алгоритъм и ключ, в резултат на което възниква шифрован текст.

Дешифриране (декриптиране) – процес на нормално прилагане на криптографско преобразуване на шифриран текст в открит.

Ключ – параметър на шифъра, определящ избора на конкретно преобразуване на дадения текст. В съвременните шифри *алгоритъма на шифриране* е известен и криптографичната устойчивост на шифъра изцяло се определя от секретността на ключа.

Криптоанализ – наука, изучаваща математическите методи за нарушаване на конфиденциалността и интегритета (цялостността) на информацията.

Криптоаналитик – човек, създаващ и прилагащ методите на криптоанализа.

Криптографията и криптоанализът съставят **Криптологията**, като единна наука за създаване и разбиване на шифри.

Криптографски инженеринг е дисциплина, която използва криптографията за разрешаване на човешки проблеми: когато трябва да се гарантира сигурността на данни, идентифициране на хора и устройства, или за да се потвърди целостта на данни в рискови ситуации.

Криптографска атака – опит на криптоаналитик да предизвика отклонения в атакуемата защитена система за обмен на информация. Успешната криптографска атака се нарича **разбиване** или **отваряне**.

Криптографска устойчивост – способността на криптографския алгоритъм да противостои на криптоанализ.

Шифрите са се използвали предимно за да се гарантира секретна комуникация в шпионажа, военното дело, и дипломацията. Но криптография се е използвала и за защита на личността, например в индийската Кама Сутра се препоръчва като средство за връзка между любовниците.

Видове шифри:

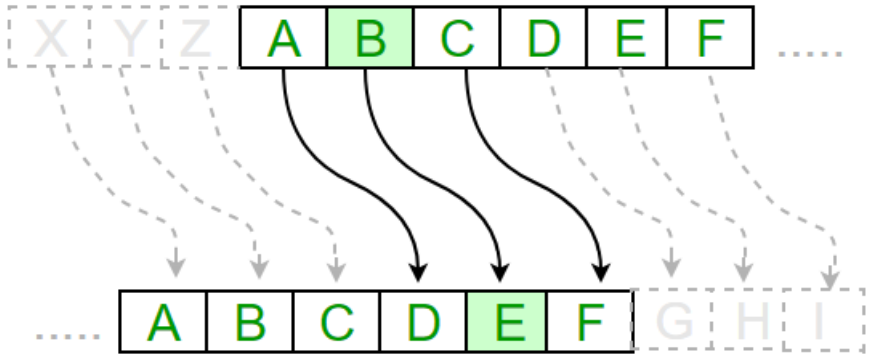
Шифърът на Цезар(Caesar Cipher)

В криптографията шифърът на Цезар е една от най-простите и широко известни техники за криптиране . Това е вид заместващ шифър , при който всяка буква в открития текст се заменя с буква, определен брой позиции надолу в азбуката . Например, при изместване наляво от 3, “D” ще бъде заменен с “A” , “E” ще стане “B” и така нататък. Методът е кръстен на Юлий Цезар, който го е използвал в личната си кореспонденция. Етапът на криптиране, извършен от шифър Цезар, често се включва като част от по-сложни схеми, като шифъра на Vigenère , и все още има модерно приложение в системата ROT13 . Както

при всички заместващи шифри с една азбука, шифърът на Цезар лесно се разбива и в съвременната практика не предлага по същество никаква комуникационна сигурност.

Шифърът на Цезар е кръстен на Юлий Цезар , който, според Светоний , го е използвал за защита на съобщения с военно значение. Докато Цезар е първото регистрирано използване на тази схема, известно е, че други заместващи шифри са били използвани по-рано. Съществуват доказателства, че Юлий Цезар е използвал и по-сложни системи, и един писател, Авл Гелий , се позовава на сега изгубен трактат за неговите шифри.

Трансформацията може да бъде представена чрез подравняване на две азбуки; шифрованата азбука е обикновена азбука, завъртяна наляво или надясно с определен брой позиции.



Например, ето един шифър на Цезар, използващ ляво завъртане на три места, еквивалентно на дясно изместване от 23.

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

Право съобщение: The frog says croak croak.

Криптирано съобщение: Qeb cold pxvp zolxh zolxh.

Когато криптира, човек търси всяка буква от съобщението в "обикновения" ред и записва съответната буква в реда "шифър".

Дешифрирането се извършва в обратна посока, с дясно изместване от 3 на кодираната азбука.

Шифроването може да бъде представено и с помощта на модулна аритметика , като първо се трансформират буквите в числа, съгласно схемата, A → 0, B → 1, ..., Z → 25. Шифриране на буква *x* чрез отместване *n* може да се опише математически.

За криптиране:

$$E_n(x) = (x + n) \mod 26.$$

За декриптиране:

$$D_n(x) = (x - n) \mod 26.$$

Има различни дефиниции за модулната операция . В горния резултат резултатът е в диапазона от 0 до 25; т.е., ако *x* + *n* или *x* − *n* не са в диапазона от 0 до 25, трябва да извадим или добавим 26. Замяната остава една и съща през цялото съобщение, така че шифърът се класифицира като тип *едноазбучно заместване*, за разлика от *полиазбучно заместване* .

Шифърът на Цезар може лесно да бъде разбит дори в сценарий само с шифротекст. Могат да се разглеждат две ситуации:

В първия случай нападателят знае или предполага, че е използван някакъв прост заместващ шифър, но не конкретно, че това е схема на Цезар. Шифърът може да бъде разбит, като се използват същите техники като за общ прост заместващ шифър, като честотен анализ или шаблонни думи. Докато решава, е вероятно нападателят бързо да

забележи редовността в решението и да заключи, че шифърът на Цезар е специфичният използван алгоритъм.

В втория случай нападателят знае, че се използва шифър на Цезар, но не знае стойността на смяната. Във втория случай нарушаването на схемата е още по-лесно. Тъй като има само ограничен брой възможни смени - 25 на английски. При втория начин нападателят просто трябва да приложи най-лесният метод за декодиране, а именно brute-force, просто трябва да премине през всичките възможни вариации на шифъра и да открие къде получава смислен текст.

Има още методи, които могат да бъдат приложени за дешифроване на шифър на Цезар, както и на други видове шифри. С шифъра на Цезар криптирането на текст многократно не осигурява допълнителна сигурност. Това е така, защото две криптирания, да речем, смяна A и shift B , ще бъдат еквивалентни на едно криптиране със смяна $A + B$. В математически план наборът от операции за криптиране под всеки възможен ключ образува група под състав.

Шифърът на Виженер(Vigenère Cipher)

Шифърът на **Виженер** е метод за криптиране на азбучен текст чрез използване на серия от преплетени шифри на Цезар, базирани на буквите на ключова дума. Той използва форма на полиазбучно заместване .

Описан за първи път от Джован Батиста Белазо през 1553 г., шифърът е лесен за разбиране и прилагане, но устоява на всички опити да бъде разбит до 1863 г., три века по-късно. Това му спечели описанието „неразбираемия шифър“. Много хора са се опитвали да приложат схеми за криптиране, които по същество са шифри на Виженер. През 1863 г. Фридрих Касиски е първият, който публикува общ метод за дешифриране на шифри на Виженер.

За криптиране може да се използва таблица с азбуки, наречена *квадрат* на Виженер. Азбуката е изписана 26 пъти в различни редове, като всяка азбука се измества циклично наляво в сравнение с предишната азбука, съответстваща на 26 възможни шифра на Цезар. В различни точки от процеса на криптиране, шифърът използва различна азбука

от един от редовете. Азбуката, използвана във всяка точка, зависи от повтаряща се ключова дума.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Например, да предположим, че откритият текст , който трябва да бъде криптиран, е

ATTACKATDAWN

Човекът, изпращащ съобщението, избира ключова дума и я повтаря, докато съвпадне с дължината на обикновения текст, например ключовата дума „LEMON“

LEMONLEMONLE

Всеки ред започва с ключова буква. Останалата част от реда съдържа буквите от A до Z в изместен ред. Въпреки че са показани 26 ключови реда, кодът ще използва само толкова клавиши различни азбуки, колкото има уникални букви в ключовия низ, тук само 5 клавиша: (L, E, M, O, N). За последователни букви от съобщението ще бъдат взети

последователни букви от ключовия низ и всяка буква на съобщението ще бъде шифрована с помощта на съответния ключов ред. Следващата буква на ключа е избрана и този ред се изминава, за да се намери заглавието на колоната, което съответства на символа на съобщението. Буквата в пресечната точка на е шифрованата буква. Например, първата буква на откритият текст, a, е сдвоена с L, първата буква на ключа. Следователно се използват ред “A” и колона “L” на квадрата на Виженер, откриваме пресечната или свързващата им точка, която дава буквата "L". По същия начин, за втората буква на откритият текст се използва втората буква на ключа. Буквата в реда “T” и колоната “E” дават буквата “X”. Останалата част от обикновения текст се шифрована по същия метод.

Обикновен текст: ATTACKATDAWN

Ключ: LEMONLEMONLE

Шифрован текст: LXFOPVEFRNHR

Декриптирането се извършва чрез преминаване към реда в таблицата, съответстващ на ключа, намиране на позицията на буквата на шифрования текст в този ред и след това използване на етикета на колоната като отворен текст. Например, в ред “L” от ключ (L, E, M, O, N), шифротекстът “L” се появява в колона “A”, като “A” е първата буква на отвореня текст. След това, в ред “E”, шифротекстът “X” се появявав колона “T”, като “T” е втората буква на отворения текст.

Шифърът може също така да се види алгебрично. Ако буквите A–Z се вземат числата 0–25, и събирането се извърши с модул 26 (за английски) или 30 за български.

За криптиране:

$$C_i \equiv P_i + K_i \pmod{26}$$ (за английски)

За декриптиране:

$$P_i \equiv C_i - K_i \pmod{26}$$ (за английски)

Идеята зад шифъра на Виженер, подобно на всички други многоазбучни шифри, е да се прикрие честотата на буквите в обикновения текст, за да се намеси в директното прилагане на честотен анализ . Например, ако “P” е най-често срещаната буква в шифров текст, чийто откровен текст е на английски , може да се подозира, че “P”отговаря на “E” тъй като “E” е най-често използваната буква на английски. Въпреки това, с помощта на шифъра на Vigenère, може да се шифрова като различни букви от шифротен текст в различни точки на съобщението, което побеждава простия честотен анализ.

Основната слабост на шифъра на Виженер е повтарящият се характер на неговия ключ . Ако криптоаналитик отгатне правилно дължината на ключа *n* , шифровият текст може да бъде третиран като *n* преплетени шифри на Цезар, които лесно могат да бъдат разбити поотделно. Дължината на ключа може да бъде открита чрез тестване с груба сила на всяка възможна стойност на *n* или изследването на Касиски и тестът на Фридман може да помогне за определяне на дължината на ключа.

Теория на кодирането

Кодовата теория изучава свойствата на кодовете и как да се използват за различни цели – компресиране на данни, криптография, отстраняване на грешки и мрежово кодиране. Кодовете също така се изучват от редица научни дисциплини – теория на информацията, електроинженерство, лингвистика, математика и компютърна наука, като главната цел на всички тях е разработването на надеждни и ефективни начини за предаване на данни. Този процес обикновено се състои от премахването на ненужни или повтарящи се парчета информация, засичането и поправянето на грешки и оптимизирането на начините за предаване и приемане на данни.

Теорията на кодовете има 4 практически дисциплини:

1. Компресиране на данни

2. Отстраняване на грешки

3. Криптографично кодиране

4. Линейно кодиране

Компресията на данни има за цел да намали размера на данните, но и да запази техния интегритет за да могат да се трансферират по-лесно и бързо.

Корекцията на грешки използва редундантно кодиране за да осигури целостта на съобщението и да намали влиянието на външни фактори.

Криптографията или криптографичното кодиране използва кодовете като средство за гарантиране на сигурност и неприкосновеност на чувствителни данни, например лична или финансова информация.

Линейното кодиране се занимава с предаването на бинарни данни, както и най-ефективните и надеждни начини за осъществяването на тази цел.

Компресия на данни

Компресирането на данни или кодиране на източника представлява създаването на код, който представя същата информация с по-малък обем, с цел по-удобно съхраняване. Кодовете са функции, които преобразуват низове от информация от една азбука в низове ,обичайно по-кратки, от друга азбука. Източник, сорс (Source code), изходен или първичен код – е „суровата“ форма, в която се пазят данни, с които работят хора и компютри. Има два основни типа компресии: със загуба на информация и без загуба.

Компресия със загуба на информация

При някои типове данни загубата на конкретна информация е приемлива. Пример за едни от основните приложения на компресия със загуба са най-често използваните формати – mp3 и mp4 за звук, jpg за картина и xvid за видео. При тях се получава компромисен вариант за качество, за сметка на обем данни, от пет до петдесет пъти по-малък. Схемите за компресии със загуба основно се състоят в изследване на това кои данни са по-маловажни, спрямо човешкото възприятие.

Компресия без загуба на информация

Компресиране без загуба се постига, чрез алгоритми, които елиминират статистическото повторение в съобщението и следователно представят данните в по-сбит вид. Например ако имаме нужда да представим информация под формата на черен екран не запазваме няколко десетки хиляди повторения на „черен пиксел“, а ги представяме като „480 000 черни пиксела“ (при резолюция 800x600).

Теорема на Шанън

През 1948 Клод Шанън публикува „Математическа теория на комуникацията”, в която сред много други, излага теоремата за ентропия на източника, която гласи, че не е възможно да се постигне компресия с кодово съотношение (брой битове, които могат да се представят с един символ, наричано още съотношение на компресия), по-малко от ентропията на източника, без това да доведе до загуба на данни.
В теорията на информацията ентропия представлява „мерна единица“ за липса на информация. Например ако разгледаме стандартно зарче състоящо се от шест страни, всяка от които съдържа число от едно до шест, всяко число има еднаква вероятност да се окаже отгоре. В този случай ентропията има максимална стойност. Също така ако използваме зарче с тежест, която подсигурява винаги едно и също число да бъде отгоре ентропията има нулева стойност, защото знаем какво ще се случи всеки път, когато метнем зарчето.

След като знаем какво представлява ентропия, можем да обобщим, че теоремата на Шанън гласи, че кодовото съотношение е обратно пропорционално на сложността на данните.

Алгоритми за кодиране

Има различни видове алгоритми за кодиране, които целят оптимален резултат при работа с различни данни. Следващите са примери за различни принципи при кодирането:

Кодиране по дължина – Посоченият по-горе пример с черните пиксели представлява кодиране по дължина. То е най-ефективно при данни, с много повторения.

Речниково кодиране – Алгоритъм без загуба, най-често използван при кодиране на текст. Всяко съвпадение в текста се заменя с индекса на кореспондиращата дума, намираща се в предефинирана структура от данни (речник). Такива алгоритми са LZ77, LZ78.

Кодиране с частично съвпадение – Използва статистически техники за предсказване на следващия символ, спрямо предишните.

Ентропологично кодиране – Алгоритъм за компресия без загуба, в който всеки символ се замества с уникален код, като най-честите символи ползват най-късите кодове. Често използвани техники са Кодировка на Хъфман и аритметичното кодиране:

Кодировка на Хъфман – Алгоритъм за ентропологично кодиране при който се използват кодове от таблица, специфично генерирана според оценената вероятност за появяване на всеки от символите от сорса.

Аритметично кодиране – Разновидност на ентропологично кодиране, при която не се кодират символи по отделно, а се кодира цялата информация под формата на число между 0.0 до 1.0

Кодиране чрез сортиране по блокове – Алгоритъм за кодиране без загуба, при който символите не се променят, а се пренареждат на редици с много повторения на един и същи символ и след това се прилага алгоритъм за кодиране на принцип, подобен на кодирането по дължина.

Засичане и отстраняване на грешки

Засичането и коригирането на грешки се състои от техники, позволяващи надеждния трансфер на данни по ненадеждни канали. Тъй като почти всички физически канали за предаване на информация са уязвими към външни смущения, множество техники са разработени с цел откриване и дори отстраняване на грешките. Дисциплината е основана от Ричард Хеминг в средата на 20 век.

Основният способ, който се използва е добавянето на допълнителни данни, които получателят може да използва, за да провери дали съобщението е пристигнало в първоначалния си вид, като някои методи позволяват и отстраняването на възникналите грешки. Спецификите на кода, който се използва зависят от характеристиките на

комуникационния канал и дали грешките възникват случайно в течение на дълъг период от време или в сравнително кратки интервали.

Подходи към отстраняването на грешки

Отстраняването на грешки има 2 главни имплементации:

Automatic Repeat Request (ARQ) – при тази методология получателят може да изисква повторно изпращане на дадени данни ако се открият грешки в тях.

Forward Error Correction (FEC) – тук изпращача включва самия код, нужен за коригиране на грешки към съобщението, като най-често резултата е близка апроксимация до оригиналните данни.

Също така съществуват множество подходи за самия процес на откриване и отстраняване на грешки:

Repetition codes – съобщението, което бива изпращано се повтаря определен брой пъти, за да се гарантира обективна интерпретация от получателя.

Parity bits – след всяка група битове се поставя бит който пази информация за това дали броят на 1-ците е четен или нечетен.

Checksum – чексумата представлява резултата от аритметична или логическа (или комбинация от двете) операция извършена върху части съобщението и прибавена след края му.

Cyclic redundancy checks – често използван при дигитални системи поради лесна имплементация и висока ефективност, този подход използва резултата от полиномиално делене върху данните.

Cryptographic hash function – поради уникалността на генерираните хеш-функции, този метод може да засече дори и най-малката промяна в интегритета на дадено съобщение. Допълнителни мерки за сигурност също могат да бъдат въведени.

Засичането и отстраняването на грешки намира огромно приложение в модерния свят, като различни имплементации се използват както и за справяне с драскотини върху DVD-та, така и за поддържане на сателитите в орбита около Земята, които са постоянно изложени на влиянието на открития космос.

Криптографично кодиране

Криптографията или криптографичното кодиране се занимава с изучаването и разработването на техники за безопасна и сигурна комуникация. Много важна част има изграждането на мрежови протоколи, които имат ролята да ограничават достъпа на трети лица до определена информация. Освен сигурността на информацията, криптографията се опитва да гарантира както конфиденциалността, така и целостта на данните, които биват предавани. Криптографията също така може да се разглежда като начин за правене на дадено съобщение по-трудно за четене.

Криптографията в модерната си форма практически не е съществувала преди началото на дигиталната революция. Техники за шифроване са съществували хиляди години преди христа, но едва през 20 век и най-вече през Втората Световна Война технологиите започват да навлизат в създаването и разбиването на шифри.

В днешни дни математическата теория и компютърната наука са изключително неразделни части от криптографията и практическата ѝ апликация. Основен принцип на сигурността, който също важи и при криптографията е това, че не е необходимо създаването на идеалната защита. Достатъчно е просто усилията и ресурсите, които биха

били нужни за разбиване на подобна система да надвишават потенциалната печалба или полза, съдържаща се в системата. Развитието и разпространението на компютрите във всичките им форми налага постоянната еволюция на криптографията и разработването на все по-трудни за пробиване решения за сигурност, чиято имплементация да е сравнително лесна и бърза.

Линеевото кодиране (Line Coding)

Линеевото кодиране представлява начинът, по който се избира оптимален код за репрезентация на даден дигитален сигнал. Използва се модулация на електрически сигнал, като моделът на вълната (waveform pattern) кореспондира на бинарната репрезентация на дигиталните данни. Спецификите на използвания код зависят от свойствата на физическия канал, по който ще се предава сигналът. Най-често срещаните видове кодове са еднополюсен, полярен, биполярен и код Манчестър.

Други приложения на теория на кодирането:

Лингвистика (Linguistics)

Лингвистиката може да се счита като клон на теорията на кодирането, тъй като езикът на най-основно ниво е вид код. Концепции, обекти и феномени от реалния свят се компресират и представят като поредица от символи. Това позволява изключително ефективна комуникация и улеснява боравенето с абстракции.

Теорията на кодирането също така позволява и изчисляването на сложността на дадено езиково семейство.

Групово тестване (Group testing)

Груповото тестване работи с големи групи елементи. Целта е да се открият малкият брой елементи, които се различават от другите като се извършат най-малко тестове. За това

се използват редица алгоритми и способи от комбинаториката. Първи стъпки в полето прави Робърт Дорфман.

Неврално кодиране (Neural coding)

Невралното кодиране изучава взаимоотношенията между това как мрежите от неврони реагират на редица стимули и как електрическата активност в мозъка кореспондира на тези стимули. Начина, по който мозъкът кодира и интерпретира информация споделя много прилики с модерните подходи за кодиране на дигитални данни. Амплитудата, честотата, силата и времето между невронните импулси могат да носят информация за стимула, на който са продукт.

Друго приложение на кодовете, използвани в някои мобилни телефонни системи, е с кодово разделен множествен достъп (CDMA). На всеки телефон е присвоена последователност от код, която е приблизително свързана помежду си с кодовете на други телефони. При предаване, кодовата дума се използва за модулиране на бита от данни, представляващи гласово съобщение.

В приемника процес демодулация се извършва за възстановяване на данни. Свойствата на този клас кодове позволяват на много потребители с различни кодове, да използват един и същ радио канал по едно и също време.

Използвани технологии

Програмният език “C”

“C” е императивен, процедурен език в традицията на ALGOL. Има система от статичен тип. В C целият изпълним код се съдържа в подпрограми, наричани още "функции", макар и не в смисъла на функционално програмиране. Параметрите на функцията се предават по стойност, въпреки че масивите се предават като указатели, тоест адреса на първия елемент в масива. *Предаване по препратка* се симулира в C чрез изрично предаване на указатели към нещото, което се препраща.

Изходният текст на програмата C е в свободен формат , използвайки точка и запетая като разделител на изрази и фигурни скоби за групиране на блокове от изрази.

Езикът C също проявява следните характеристики:

Езикът има малък, фиксиран брой ключови думи, включително пълен набор от примитиви на контролния поток if/else : , for, do/while, while, и switch. Дефинираните от потребителя имена не се различават от ключовите думи с какъвто и да е вид сигил.

Той има голям брой аритметични, побитови и логически оператори: +, +=, ++, &, ||, и т.н.

В едно изявление може да се извърши повече от едно задание.

Функции:

Върнатите стойности на функцията могат да бъдат игнорирани, когато не са необходими.

Указателите на функции и данни позволяват *ad hoc* полиморфизъм по време на изпълнение.

Функциите не могат да бъдат дефинирани в рамките на лексикалния обхват на други функции.

Променливите могат да бъдат дефинирани в рамките на функция с обхват.

Функцията може да се извика сама, така че се поддържа рекурсия.

Въвеждането на данни е статично, но слабо наложено; всички данни имат тип, но са възможни неявни преобразувания.

Възможни са дефинирани от потребителя (typedef) и съставни типове.

Хетерогенните обобщени типове данни (struct) позволяват достъп до свързани елементи от данни и присвояване като единица.

Съюзът е структура с припокриващи се членове; валиден е само последният съхранен член.

Индексирането на масиви е вторична нотация, дефинирана от гледна точка на аритметиката на показалеца. За разлика от структурите, масивите не са първокласни обекти: те не могат да бъдат присвоени или сравнени с помощта на единични вградени оператори. Няма използвана или дефиниция ключова дума "масив"; вместо това квадратните скоби показват синтактично масивите, например month.

Изброените типове са възможни с enumключовата дума. Те са свободно конвертируеми с цели числа.

Низовете не са отделен тип данни, но конвенционално се изпълняват като символни масиви със завършване с нула.

Достъпът на ниско ниво до компютърната памет е възможен чрез преобразуване на адресите на машината в указатели.

Процедурите (подпрограми, които не връщат стойности) са специален случай на функция с нетипизиран тип връщане void.

Паметта може да бъде разпределена на програма с извиквания към библиотечни процедури.

Предпроцесорът извършва дефиниране на макроси, включване на файл с изходен код и условна компилация.

Съществува основна форма на модулност: файловете могат да бъдат компилирани отделно и свързани заедно, с контрол върху това кои функции и обекти с данни са видими за други файлове чрез static и extern атрибути.

Сложни функционалности като I/O, манипулиране на низове и математически функции последователно се делегират на библиотечни рутинни процедури.

Генерираният код след компилация има относително прости нужди на основната платформа, което го прави подходящ за създаване на операционни системи и за използване във вградени системи.

Въпреки че C не включва определени функции, намиращи се в други езици ,като ориентация на обекта и събиране на боклук, те могат да бъдат внедрени или емулирани, често чрез използване на външни библиотеки (напр. GLib Object System или Boehm gabage collector).

Програмния език “C++”

C++ е език за програмиране с общо предназначение, създаден от Bjarne Stroustrup като разширение на езика за програмиране C или “C с класове”. Езикът се разшири значително с течение на времето и съвременният C++ вече има обектно-ориентирани, общи и функционални функции в допълнение към средствата за манипулиране на паметта на ниско ниво. Почти винаги се изпълнява като акомпилиран език и много доставчици предоставят C++ компилатори, включително Free Software Foundation, LLVM, Microsoft, Intel , Oracle и IBM, така че е достъпен на много платформи.

C++ е проектиран с ориентация към системно програмиране и вграден софтуер с ограничени ресурси и големи системи, с производителност, ефикасност и гъвкавост на използване като акценти в дизайна. C++ също е намерен за полезен в много други контексти, като ключови силни страни са софтуерна инфраструктура и приложения с ограничени ресурси, включително настолни приложения, видео игри, сървъри, електронна търговия, търсене в мрежата или бази данни, и критични за производителността приложения, като телефонни превключватели или космически сонди.

C++ е стандартизиран от Международната организация по стандартизация (ISO), като последната версия на стандарта е ратифицирана и публикувана от ISO през декември 2020 г. като *ISO/IEC 14882:2020* (неофициално известен като C++20). Езикът за програмиране C++ първоначално е стандартизиран през 1998 г. като *ISO/IEC 14882:1998* , който след това е изменен от стандартите C++03, C++11, C++14 и C++17 . Настоящият стандарт C++20 ги замества с нови функции и разширена стандартна библиотека. Преди

първоначалната стандартизация през 1998 г., C++ е разработен от датския компютърен учен Bjarne Stroustrup в Bell Labs от 1979 г. като разширение на езика C ; той искаше ефективен и гъвкав език, подобен на C, който също предоставя функции на високо ниво за организация на програмата. От 2012 г. C++ е на тригодишен график за пускане с C++23 като следващия планиран стандарт.

Реализация на приложението

Приложението е разделено в два „.cpp“ и един „.h“ файла. В „main.c“ се съдържа базовата логика на приложението. В „functions.cpp“ се съдържат детайлната имплементация на функционалностите на приложението.

В „functions.h“ се съдържат декларациите на публичните функции от „functions.cpp“ и няколко константи, за да бъде по-добре организирана цялостната програма.

main.cpp

```
#include <iostream>
#include "functions.h"

using namespace std;

int main()
{
    int choice = 1; // Not 0

    //Set font color to green
    system("Color 0A");

    while (choice != 0)
    {
        system("CLS");
        printMenu();
        choice = getChoiceFromUser();

        switch (choice)
        {
        case 1:
            encryption();
            break;
        case 2:
            decryption();
            break;
        case 3:
            bruteForce();
            break;
        case 4:
            fileReader(true);
            break;
        case 5:
            fileReader(false);
            break;
        default:
            return 0;
            break;
        }
```

```
        printEnd();
        system("PAUSE");
    }
}
```

Инклудваме библиотеката „iostream“, за да използваме стандартните функции за вход и изход.

Инклудваме собствения хедър файл „functions“, за да имаме достъп до функциите в него.

Задаваме цвета на конзолата да бъде черен(0) и цвета на текста да бъде ярко-зелено(A).

В цикъл започваме да извършваме следните команди докато не получим избор „0“ от потребителя:

- Изтриваме съдържанието на конзолата
- Принтираме менюто
- Вземаме избор от потребителя
- На база този избор извикваме съответната функция или прекратяваме изпълнението на програмата
- Принтираме символи за край
- Паузираме изпълнението на програмата

functions.h

```
#ifndef FUNCTIONS_H_INCLUDED
#define FUNCTIONS_H_INCLUDED

//------------------------------------------------
// CONSTANTS
//------------------------------------------------
#define KEY_MIN 1
#define KEY_MAX 25

#define CHOICE_MIN 0
#define CHOICE_MAX 5

#define SLEEP_TIME 500

#define COUNT_LETTERS 26

//------------------------------------------------
// FUNCTION DECLARATIONS
//------------------------------------------------
void printMenu();
void printEnd();
int getChoiceFromUser();
void encryption();
void decryption();
void bruteForce();
void fileReader(bool mode);

#endif // FUNCTIONS_H_INCLUDED
```

Използваме защита против многократно инклудване на хедъра.

Дефинираме константи:

- KEY_MIN 1 – минимална валидна стойност за ключ
- KEY_MAX 25 – максимална валидна стойност за ключ
- CHOICE_MIN 0 – минимална валидна стойност за избор
- CHOICE_MAX 5 – максимална валидна стойност за избор
- SLEEP_TIME 500 – време за изчакване между принтирането на шифрираните/дешифрираните символи (в милисекунди)
- COUNT_LETTERS 26 – брой символи в Английската азбука

Декларираме функциите, които ще бъдат използвани в „main.cpp“

Погледнете Приложение 1

functions.cpp

Инклуди

Инклудваме библиотеката „iostream“, за да използваме стандартните функции за вход и изход.

Инклудваме библиотеката „Windows“, за да използваме функцията Sleep(), която спира работата на програмата за подаден брой милисекунди.

Инклудваме библиотеката „cctype“, за да използваме функцията isalpha (), която ни връща дали даден символ е буква.

Инклудваме библиотеката „fstream“, за да използваме стандартните функции за работа с потоци от данни.

Инклудваме собствения хедър файл „functions“, за да имаме достъп до функциите в него.

Частни функции

Име	checkMessageValidity
Параметри	string message – низ, който трябва да бъде валидиран
Връщана с-ст	bool – истина, ако низа е валиден, иначе лъжа
Описание	Проверява всеки символ от низа дали е буква или символ за празно пространство.

Име	getMessageFromUser
Параметри	няма
Връщана с-ст	string
Описание	Принтира информация на потребителя. Прочита един низ от конзолата. Ако низът е невалиден, се извежда съобщение за грешка и

	се изчаква за нов, това се повтаря, докато не се въведе валиден низ. Валидира се чрез checkMessageValidity().

Име	getKeyFromUser
Параметри	няма
Връщана с-ст	int – ключ от 1 до 25
Описание	Принтира информация на потребителя. Прочита едно цяло число от конзолата. Ако то е невалидно, се извежда съобщение за грешка и се изчаква за ново, това се повтаря, докато не се въведе валидно число.

Име	getFilePathFromUser
Параметри	string fileType – низ, използван за принтиране
Връщана с-ст	string – валиден път за файл
Описание	Принтира информация на потребителя. Прочита един низ от конзолата. Ако низът е по-къс от 5 символа или не завършва на „.txt“, се извежда съобщение за грешка и се изчаква за нов, това се повтаря, докато не се въведе валиден низ.

Име	print
Параметри	string originalmsg – въведен низ от потребителя string processedmsg – преработен низ от алгоритъма
Връщана с-ст	няма
Описание	Принтира оригиналния низ. Започва да редува изтриване на низ с принтиране на +1 символ от преработения низ и остатъка от стария с кратка пауза. Създава се илюзията, че съобщението се променя символ по символ.

Име	encryptWithKey
Параметри	string originalmsg – въведен низ от потребителя int key – отместващото число
Връщана с-ст	string – криптирания низ

Описание	Отмества напред всеки един символ от оригиналния низ с „key“ позиции, освен празните пространства. Ако след отместването символът излиза от пределите на азбуката, остатъкът от отместването се започва от началото на азбуката. Логиката работи както с малки, така и с главни букви.

Име	decryptWithKey
Параметри	string originalmsg - въведен низ от потребителя int key - отместващото число
Връщана с-ст	string – декриптиран низ
Описание	Отмества назад всеки един символ от оригиналния низ с „key“ позиции, освен празните пространства. Ако след отместването символът излиза от пределите на азбуката, остатъкът от отместването се започва от края на азбуката. Логиката работи както с малки, така и с главни букви.

Публични функции

Име	printMenu
Параметри	няма
Връщана с-ст	няма
Описание	Принтира меню с опции.

Име	printEnd
Параметри	няма
Връщана с-ст	няма
Описание	Принтира нов ред и черта.

Име	getChoiceFromUser
Параметри	няма
Връщана с-ст	int – избор от 0 до 6

Описание	Принтира информация на потребителя. Прочита едно цяло число от конзолата. Ако то е невалидно, се извежда съобщение за грешка и се изчаква за ново, това се повтаря, докато не се въведе валидно число.

Име	encryption
Параметри	няма
Връщана с-ст	няма
Описание	Взима съобщение от потребителя чрез getMessageFromUser(). Взима ключ от потребителя чрез getKeyFromUser(). Криптира даденото съобщение с дадения ключ чрез encryptWithKey(). Принтира криптирането чрез print().

Име	decryption
Параметри	няма
Връщана с-ст	няма
Описание	Взима съобщение от потребителя чрез getMessageFromUser(). Взима ключ от потребителя чрез getKeyFromUser(). Декриптира даденото съобщение с дадения ключ чрез decryptWithKey(). Принтира декриптирането чрез print().

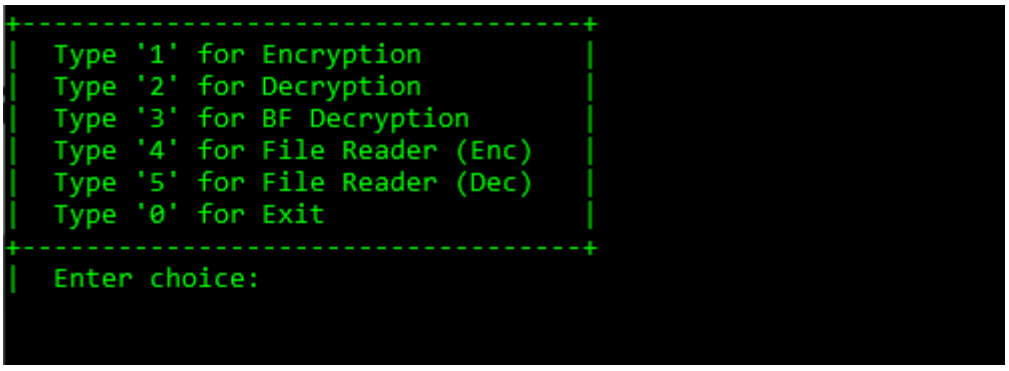
Име	bruteForce
Параметри	няма
Връщана с-ст	няма
Описание	Взима съобщение от потребителя чрез getMessageFromUser(). Декриптира даденото съобщение с всеки ключ от 1 до 25 чрез decryptWithKey(). Принтира всяко различно декриптирано съобщение едно след друго с кратка пауза.

Име	fileReader
Параметри	bool mode – истина за криптиране, лъжа за декриптиране
Връщана с-ст	няма

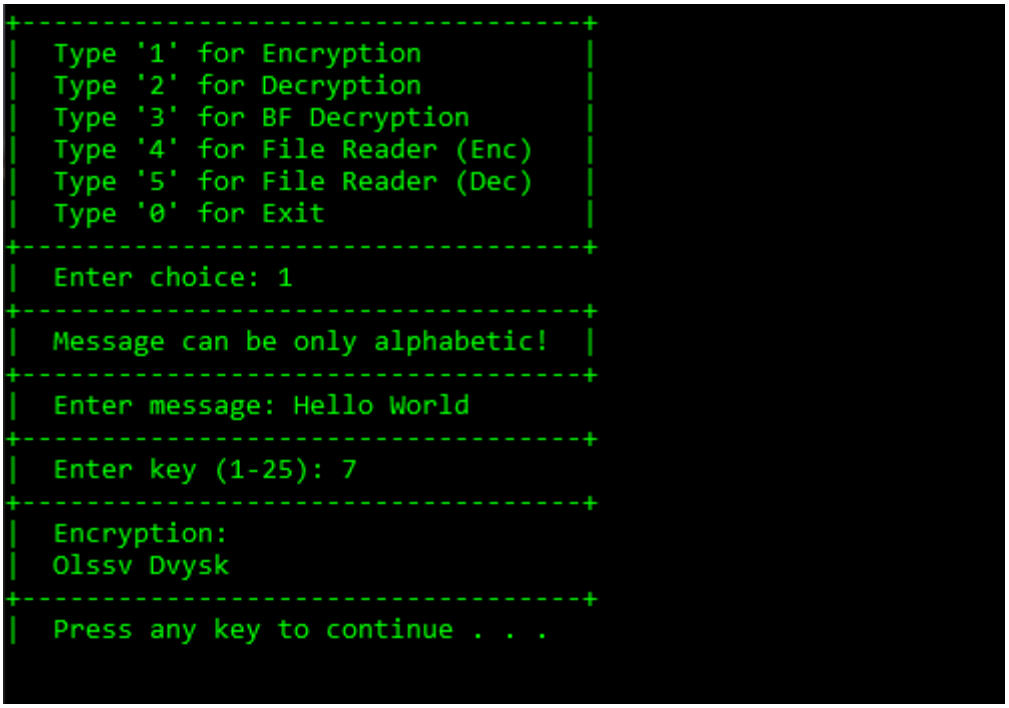
Описание	1. Взима файлов път от потребителя чрез getFilePathFromUser(). • Ако не може да се отвори такъв файл се извежда съобщение и се приключва изпълнението на функцията. 2. Проверява се съдържанието на файла чрез checkMessageValidity(). • Ако съдържанието не е валидно се извежда съобщение, затваря се входния файл и се приключва изпълнението на функцията. 3. Взима ключ от потребителя чрез getKeyFromUser(). 4. Взима файлов път от потребителя чрез getFilePathFromUser(). • Ако не може да се отвори такъв файл се извежда съобщение, затваря се входния файл и се приключва изпълнението на функцията. 5. Връща файловия указател в началото на входния файл, защото в момента се намира в края, заради проверката на съдържанието. 6. Проверява режима на работа. • При криптиране – прочита съдържанието на входния файл ред по ред. Криптира го чрез encryptWithKey(). Принтира криптирането чрез print(). Записва криптираното съдържание в изходния файл. • При декриптиране – прочита съдържанието на входния файл ред по ред. Декриптира го чрез decryptWithKey(). Принтира декриптирането чрез print(). Записва декриптираното съдържание в изходния файл. 7. Затваря входния и изходния файл.

Ръководство за потребителя

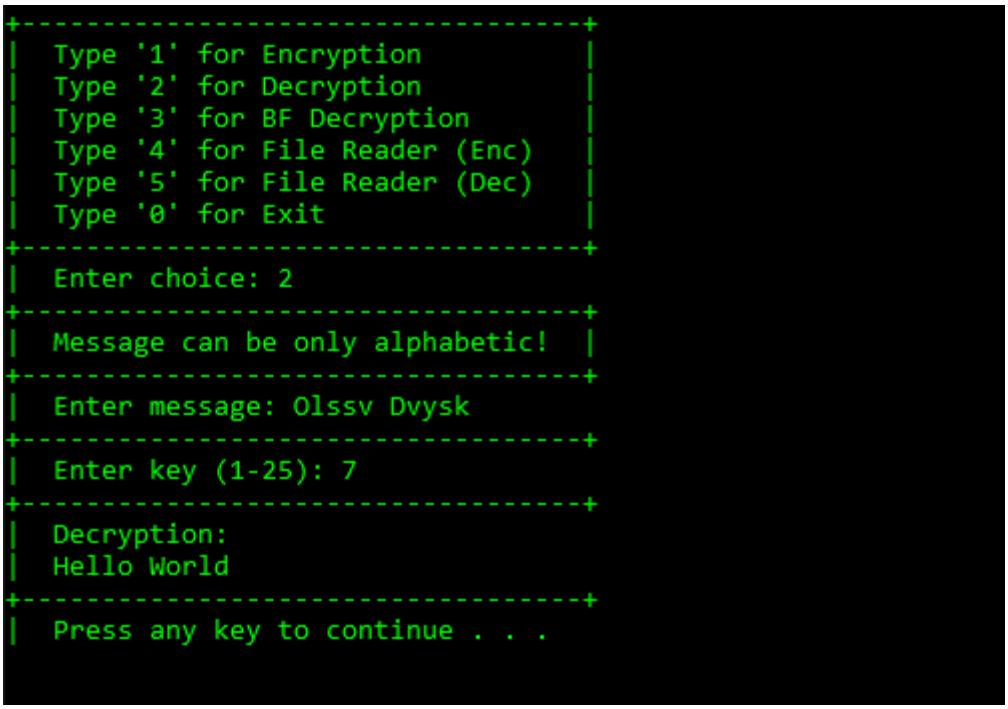
Начално меню



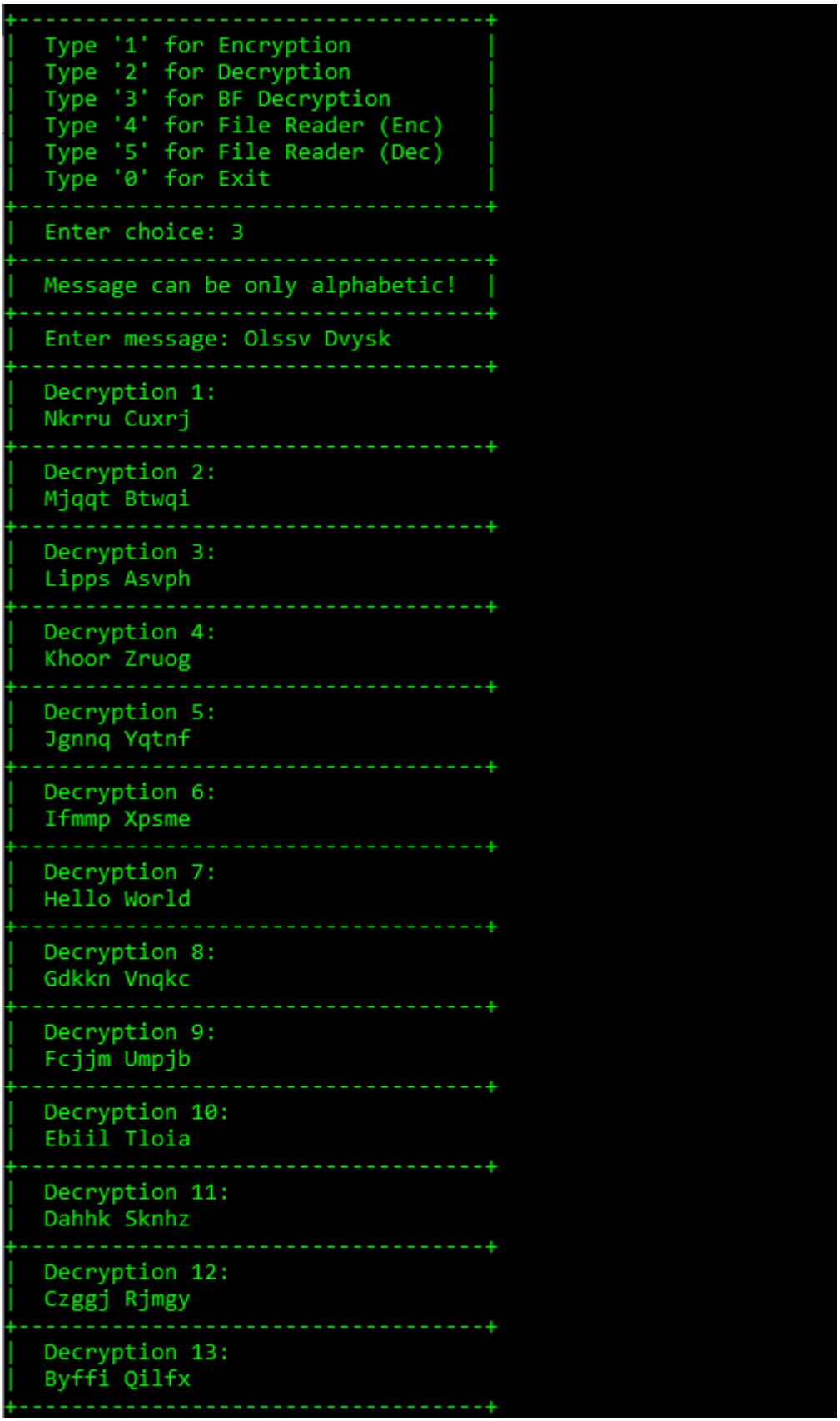
Опция 1 – Криптиране

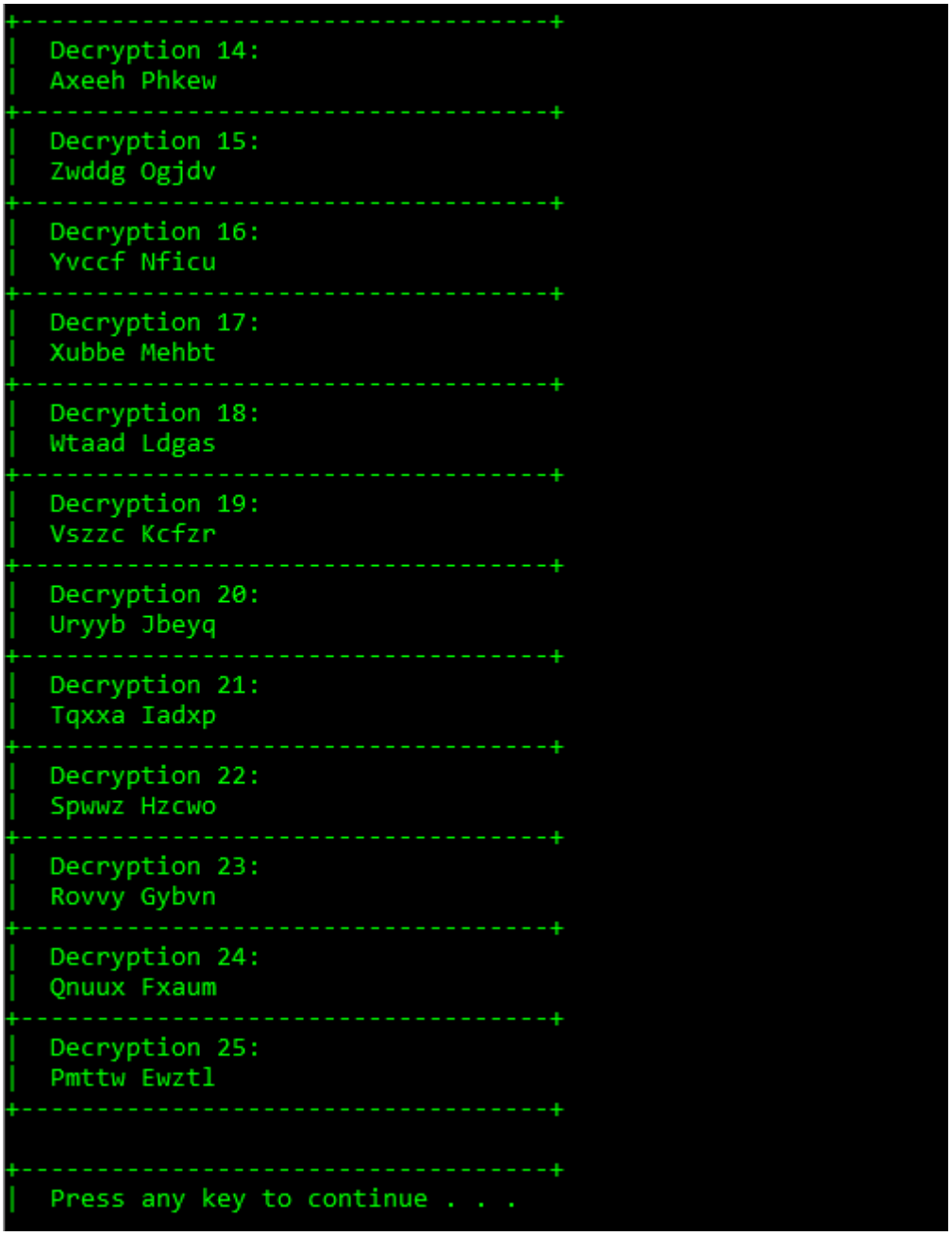


Опция 2 – Декриптиране

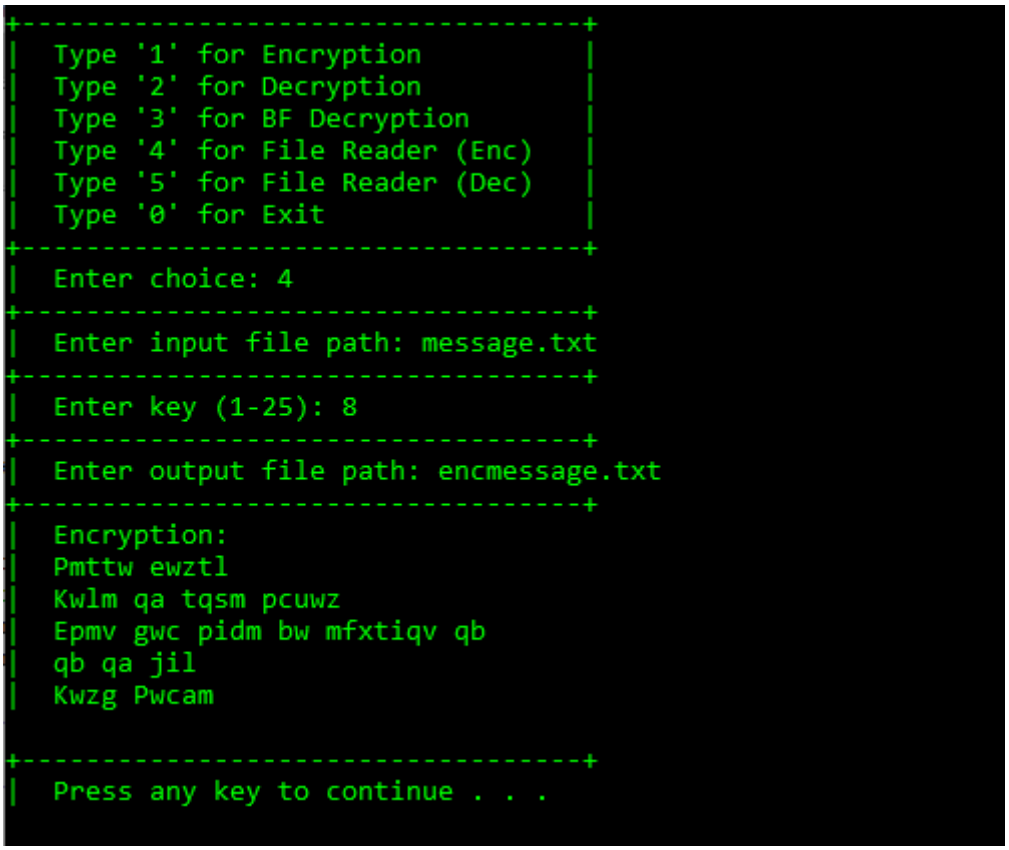


Опция 3 – Декриптиране чрез Brute Force

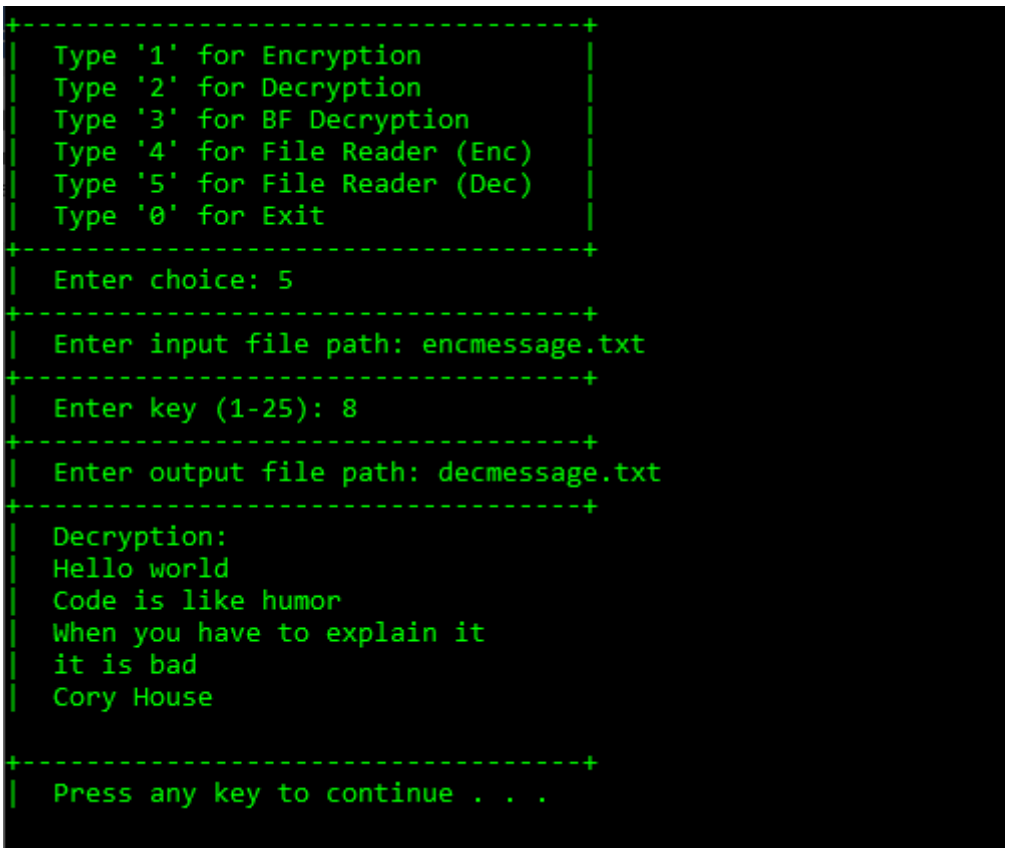



Decryption 14:
Axeeh Phkew
Decryption 15:
Zwddg Ogjdv
Decryption 16:
Yvccf Nficu
Decryption 17:
Xubbe Mehbt
Decryption 18:
Wtaad Ldgas
Decryption 19:
Vszzc Kcfzr
Decryption 20:
Uryyb Jbeyq
Decryption 21:
Tqxxa Iadxp
Decryption 22:
Spwwz Hzcwo
Decryption 23:
Rovvy Gybvn
Decryption 24:
Qnuux Fxaum
Decryption 25:
Pmttw Ewztl
Press any key to continue . . .

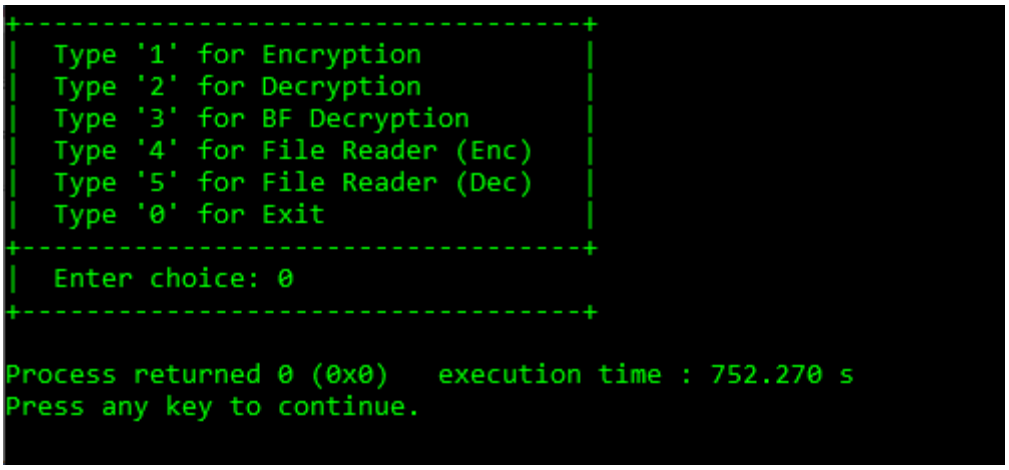
Опция 4 – Криптиране на информация от файл



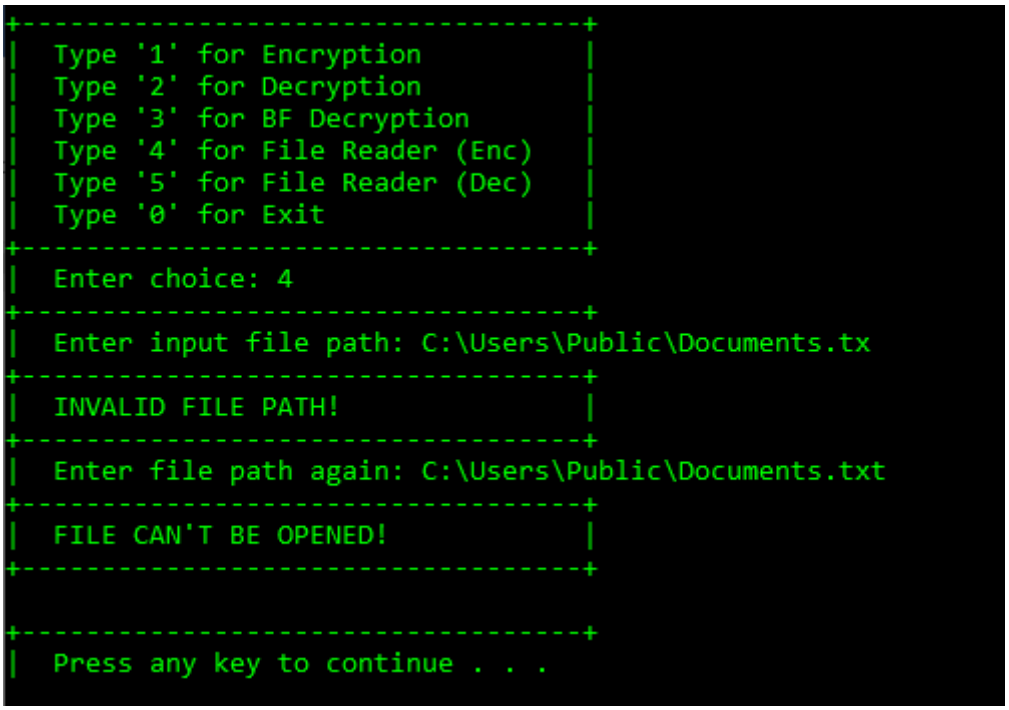
Опция 5 – Декриптиране на информация от файл

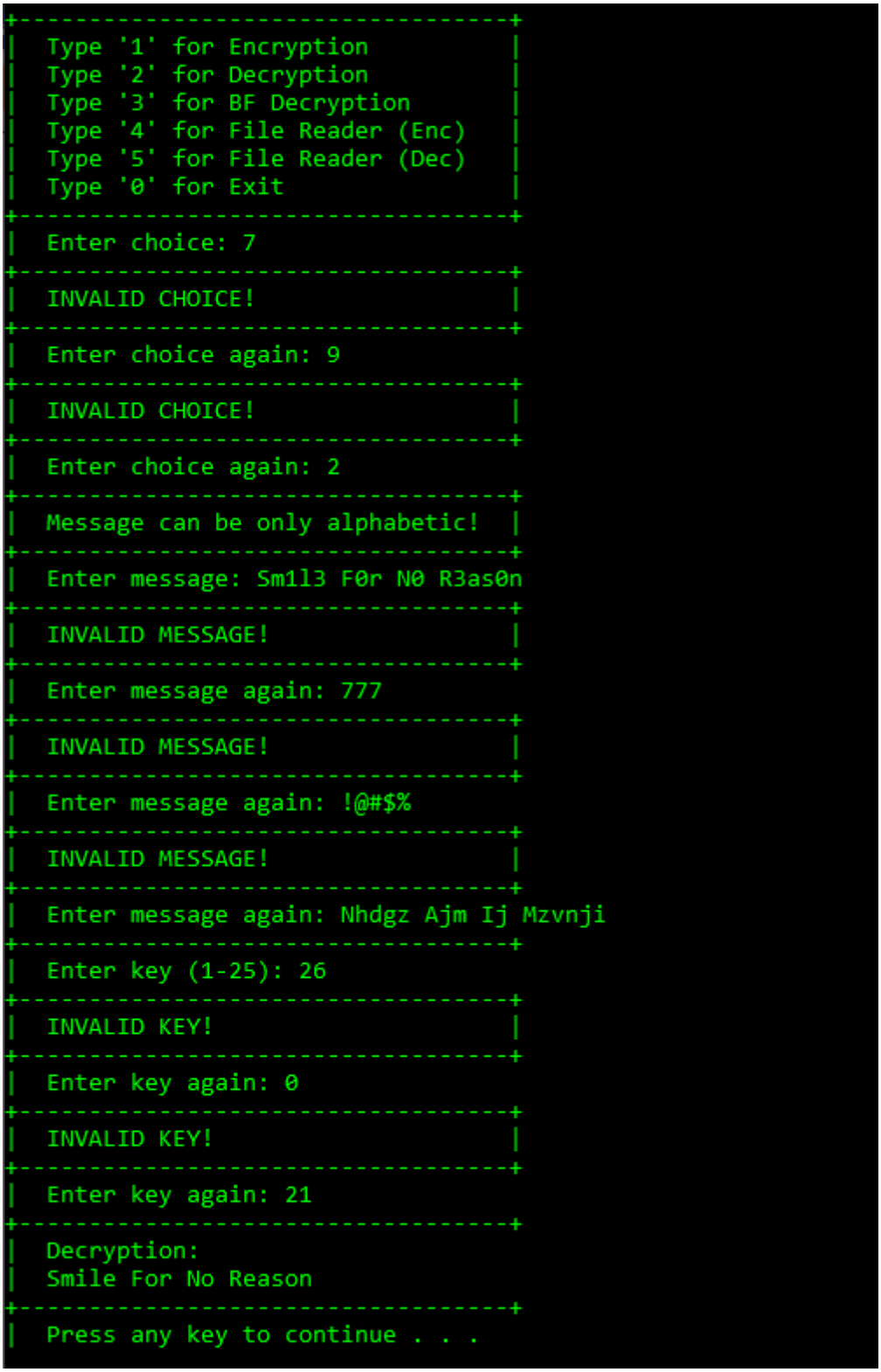


Опция 0 – Изход



Съобщения при невалидна информация




Type '1' for Encryption
Type '2' for Decryption
Type '3' for BF Decryption
Type '4' for File Reader (Enc)
Type '5' for File Reader (Dec)
Type '0' for Exit
Enter choice: 7
INVALID CHOICE!
Enter choice again: 9
INVALID CHOICE!
Enter choice again: 2
Message can be only alphabetic!
Enter message: Sm1l3 F0r N0 R3as0n
INVALID MESSAGE!
Enter message again: 777
INVALID MESSAGE!
Enter message again: !@#$%
INVALID MESSAGE!
Enter message again: Nhdgz Ajm Ij Mzvnji
Enter key (1-25): 26
INVALID KEY!
Enter key again: 0
INVALID KEY!
Enter key again: 21
Decryption:
Smile For No Reason
Press any key to continue . . .

Заключение:

Успях да реализирам цялостната система и идея на приложението.

Възможна актуализация за в бъдеще:

- Да се добави графичен интерфейс.
- Да се добавят n на брой функции за криптиране и декриптиране на различни видове шифри.
- Да се създаде тип игра за обучение.

Източници:

1. https://www.dcode.fr/en
2. https://www.geeksforgeeks.org/
3. https://en.wikipedia.org/wiki/Main_Page
4. https://softuni.bg/
5. https://www.w3schools.com/
6. https://www.programiz.com/cpp-programming
7. https://www.youtube.com/watch?v=-Gk9kaFoBxU&ab_channel=RyanKral
8. https://www.youtube.com/watch?v=SkJcmCaHqS0&ab_channel=Udacity
9. https://media.geeksforgeeks.org/wp-content/uploads/ceaserCipher.png
10. https://www.google.com/url?sa=i&url=https%3A%2F%2Fstackoverflow.com%2Fquestions%2F33807658%2Fvigenere-square-lookup-using-string-arrays&psig=AOvVaw0Mb7N_xDHgndDmWdBg4qgj&ust=1651657759019000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCPj80OKGw_cCFQAAAAAdAAAAABAJ

Приложение 1

functions.cpp

```
#include <iostream>
#include <Windows.h>
#include <cctype>
#include <fstream>
#include "functions.h"

using namespace std;

static bool checkMessageValidity(string message);
static string getMessageFromUser();
static int getKeyFromUser();
static string getFilePathFromUser(string fileType);
static void print(string originalmsg, string processedmsg);
static string decryptWithKey(string originalmsg, int key);
static string encryptWithKey(string originalmsg, int key);

//---------------------------------------------------
// PRIVATE FUNCTIONS
//---------------------------------------------------

//---------------------------------------------------
// Description : Checks if message contains only letters
// Parameters  : message - string to be verified
// Return value: true - if message is OK, false - otherwise
//---------------------------------------------------
static bool checkMessageValidity(string message)
{
    for (int i = 0; i < message.size(); i++)
    {
        if ( (isalpha(message[i]) == false) &&
             (message[i] != ' ' ) &&
             (message[i] != '\n') &&
             (message[i] != '\t') )
        {
            return false;
        }
    }
    return true;
}

//---------------------------------------------------
// Description : Gets valid input message from the User
// Parameters  : None
// Return value: String
//---------------------------------------------------
static string getMessageFromUser()
{
    string message;
    cout << "|  Message can be only alphabetic!  |" << endl;
    cout << "+-----------------------------------+" << endl;
    cout << "|  Enter message: ";
```

```
    getline(cin, message);
    cout << "+----------------------------------+" << endl;

    while (false == checkMessageValidity(message))
    {
        cout << "|  INVALID MESSAGE!                |" << endl;
        cout << "+----------------------------------+" << endl;
        cout << "|  Enter message again: ";
        getline(cin, message);
        cout << "+----------------------------------+" << endl;
    }

    return message;
}

//-----------------------------------------------------
// Description : Gets valid key from the User
// Parameters  : None
// Return value: Key from 1 to 25
//-----------------------------------------------------
static int getKeyFromUser()
{
    int key;
    cout << "|  Enter key (1-25): ";
    cin >> key;
    cin.ignore();
    cout << "+----------------------------------+" << endl;

    while( (key < KEY_MIN) || (key > KEY_MAX))
    {
        cout << "|  INVALID KEY!                    |" << endl;
        cout << "+----------------------------------+" << endl;
        cout << "|  Enter key again: ";
        cin >> key;
        cin.ignore();
        cout << "+----------------------------------+" << endl;
    }

    return key;
}

//-----------------------------------------------------
// Description : Gets valid file path from the User
// Parameters  : fileType - input or output
// Return value: String
//-----------------------------------------------------
static string getFilePathFromUser(string fileType)
{
    string filepath;
    cout << "|  Enter " << fileType << " file path: ";
    getline(cin, filepath);
    cout << "+----------------------------------+" << endl;
    int filePathSize = filepath.size();

    // Basic check for .txt
    while( (filePathSize < 5) ||
           ( (filepath[filePathSize-4] != '.') ||
             (filepath[filePathSize-3] != 't') ||
             (filepath[filePathSize-2] != 'x') ||
             (filepath[filePathSize-1] != 't') ) )
    {
```

```
        cout << "|  INVALID FILE PATH!                |" << endl;
        cout << "+------------------------------------+" << endl;
        cout << "|  Enter file path again: ";
        getline(cin, filepath);
        cout << "+------------------------------------+" << endl;
        filePathSize = filepath.size();
    }

    return filepath;
}

//-----------------------------------------------------
// Description : Prints the process of encryption/decryption
// Parameters  : originalmsg - message from user
//               processedmsg - message resulted from used algorithm
// Return value: None
//-----------------------------------------------------
static void print(string originalmsg, string processedmsg)
{
    string result = originalmsg;

    cout << "|  " << originalmsg;
    for (int i = 0; i < originalmsg.size(); i++)
    {
        Sleep(SLEEP_TIME); //time in ms
        // Delete old message
        for( int i = 0; i < originalmsg.size(); i++)
        {
            cout << "\b";
        }
        result[i] = processedmsg[i];
        cout << result;
    }
}

//-----------------------------------------------------
// Description : Encrypts a message by offsetting the original value with
the key
// Parameters  : originalmsg - message to be encrypted
//             : key - offest for encryption
// Return value: encryptedmsg - encrypted message
//-----------------------------------------------------
static string encryptWithKey(string originalmsg, int key)
{
    string encryptedmsg = originalmsg;

    for (int i = 0; i < originalmsg.size(); i++)
    {
        if ((originalmsg[i]== ' ') || (originalmsg[i] == '\n') ||
(originalmsg[i] == '\t'))
        {
            continue;
        }
        else
        {
            if ((originalmsg[i]+key) > 122)
            {
                //after lowercase z move back to a, z's ASCII is 122
                int temp = (originalmsg[i] + key) - 122;
                encryptedmsg[i] = 96 + temp;
            }
```

```
            else if (originalmsg[i] + key > 90 && originalmsg[i] <= 90)
            {
                //after uppercase Z move back to A, 90 is Z's ASCII
                int temp = (originalmsg[i] + key) - 90;
                encryptedmsg[i] = 64 + temp;
            }
            else
            {
                //in case of characters being in between A-Z & a-z
                encryptedmsg[i] += key;
            }
        }
    }

    return encryptedmsg;
}

//-------------------------------------------------
// Description : Decrypts a message by offsetting the original value with
the key
// Parameters  : originalmsg - message to be decrypted
//             : key - offest for decryption
// Return value: decryptedmsg - decrypted message
//-------------------------------------------------
static string decryptWithKey(string originalmsg, int key)
{
    string decryptedmsg = originalmsg;

    for (int i = 0; i < originalmsg.size(); i++)
    {
        if ((originalmsg[i]== ' ') || (originalmsg[i] == '\n') ||
(originalmsg[i] == '\t'))
        {
            continue;
        }
        else
        {
            if ((originalmsg[i] - key) < 97 && (originalmsg[i] >= 97)) //
97 -> a
            {
                int temp = (originalmsg[i] - key) + COUNT_LETTERS;
                decryptedmsg[i] = temp;
            }
            else if ((originalmsg[i] - key) < 65) // 65 -> A
            {
                int temp = (originalmsg[i] - key) + COUNT_LETTERS;
                decryptedmsg[i] = temp;
            }
            else
            {
                decryptedmsg[i] = originalmsg[i] - key;
            }
        }
    }

    return decryptedmsg;
}
//-------------------------------------------------
// PUBLIC FUNCTIONS
//-------------------------------------------------
```

```
//---------------------------------------------------
// Description : Prints Menu
// Parameters  : None
// Return value: None
//---------------------------------------------------
void printMenu()
{
    cout << "+-----------------------------------+" << endl;
    cout << "|  Type '1' for Encryption          |" << endl;
    cout << "|  Type '2' for Decryption          |" << endl;
    cout << "|  Type '3' for BF Decryption       |" << endl;
    cout << "|  Type '4' for File Reader (Enc)   |" << endl;
    cout << "|  Type '5' for File Reader (Dec)   |" << endl;
    cout << "|  Type '0' for Exit                |" << endl;
    cout << "+-----------------------------------+" << endl;
}

//---------------------------------------------------
// Description : Prints end characters
// Parameters  : None
// Return value: None
//---------------------------------------------------
void printEnd()
{
    cout << endl;
    cout << "+-----------------------------------+" << endl;
    cout << "|  ";
}

//---------------------------------------------------
// Description : Gets valid input choice from the User
// Parameters  : None
// Return value: Choice from MIN to MAX
//---------------------------------------------------
int getChoiceFromUser()
{
    int choice = 0;

    cout << "|  Enter choice: ";
    cin >> choice;
    cin.ignore();
    cout << "+-----------------------------------+" << endl;

    while( (choice < CHOICE_MIN) || (choice >CHOICE_MAX) )
    {
        cout << "|  INVALID CHOICE!                  |" << endl;
        cout << "+-----------------------------------+" << endl;
        cout << "|  Enter choice again: ";
        cin >> choice;
        cin.ignore();
        cout << "+-----------------------------------+" << endl;
    }

    return choice;
}

//---------------------------------------------------
// Description : Encrypts a message from User with a given key
// Parameters  : None
// Return value: None
//---------------------------------------------------
```

```
void encryption()
{
    string originalmsg = getMessageFromUser();

    int key = getKeyFromUser();

    string encryptedmsg = encryptWithKey(originalmsg, key);

    cout << "|  Encryption: " << endl;
    print(originalmsg,encryptedmsg);
}

//-----------------------------------------------------
// Description : Decrypts a message from User with a given key
// Parameters  : None
// Return value: None
//-----------------------------------------------------
void decryption()
{
    string originalmsg = getMessageFromUser();

    int key = getKeyFromUser();

    string decryptedmsg = decryptWithKey(originalmsg, key);

    cout << "|  Decryption: " << endl;
    print(originalmsg,decryptedmsg);
}

//-----------------------------------------------------
// Description : Decrypts a message from User by brute forcing with all
possible keys
// Parameters  : None
// Return value: None
//-----------------------------------------------------
void bruteForce()
{
    string originalmsg = getMessageFromUser();
    string decryptedmsg;

    for(int key = KEY_MIN; key <= KEY_MAX; key++)
    {
        string decryptedmsg = decryptWithKey(originalmsg, key);
        cout << "|  Decryption " << key << ":" << endl;
        cout << "|  " << decryptedmsg << endl;
        cout << "+---------------------------------+" << endl;
        Sleep(SLEEP_TIME);
    }
}

//-----------------------------------------------------
// Description : Opens a file given from the User, which contents will be
encrypted/decrypted
// Parameters  : mode - encryption or decryption
// Return value: None
//-----------------------------------------------------
void fileReader(bool mode)
{
    string filepath = getFilePathFromUser("input");
    string line;
    ifstream inputFile;
```

```
ofstream outputFile;
inputFile.open(filepath);

if (inputFile.is_open() == false)
{
    cout << "|  FILE CAN'T BE OPENED!            |" << endl;
    cout << "+-----------------------------------+" << endl;
}
else
{
    while (getline(inputFile,line))
    {
        if (checkMessageValidity(line) == false)
        {
            cout << "|  FILE HAS WRONG CONTENT!          |" << endl;
            cout << "+-----------------------------------+" << endl;
            inputFile.close();
            return;
        }
    }

    int key = getKeyFromUser();

    filepath = getFilePathFromUser("output");
    outputFile.open(filepath);
    if (outputFile.is_open() == false)
    {
        cout << "|  FILE CAN'T BE OPENED!            |" << endl;
        cout << "+-----------------------------------+" << endl;
        inputFile.close();
        return;
    }

    //Return file pointer to begging of the file
    inputFile.clear();
    inputFile.seekg(ios::beg);

    if(mode == true)
    {
        string encryptedmsg;
        cout << "|  Encryption: " << endl;
        while (getline(inputFile,line))
        {
            encryptedmsg = encryptWithKey(line, key);
            print(line,encryptedmsg);
            cout << endl;
            outputFile << encryptedmsg << endl;
        }
    }
    else
    {
        string decryptedmsg;
        cout << "|  Decryption: " << endl;
        while (getline(inputFile,line))
        {
            decryptedmsg = decryptWithKey(line, key);
            print(line,decryptedmsg);
            cout << endl;
            outputFile << decryptedmsg << endl;
        }
    }
```

```
        inputFile.close();
        outputFile.close();
    }
}
```