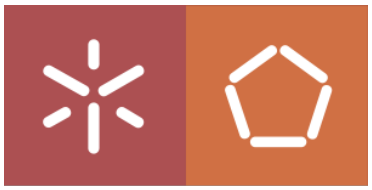


Universidade do Minho
Escola de Engenharia

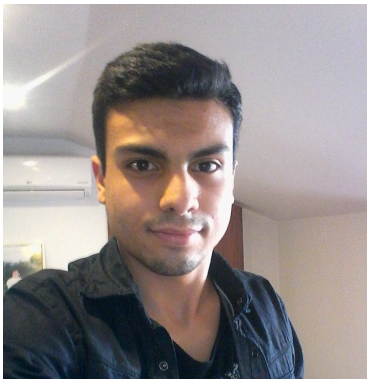
Universidade do Minho
Mestrado Integrado em Engenharia Informática

Tecnologia Criptográfica

Trabalho prático 1

29 Novembro 2020

Ana Margarida Campos
(A85166)



Nuno Pereira
(PG42846)

Contents

1 Introdução **4**
- 1.1 Contextualização . . . 4
- 1.2 Objetivos e Trabalho Proposto . . . 4
- 1.3 Estrutura do Relatório . . . 4

2 Cifra de Vigenère **5**
- 2.1 Cifragem . . . 5
- 2.2 Decifragem . . . 5
  - 2.2.1 Encontrar tamanho da chave: . . . 6
  - 2.2.2 Encontrar cada número na chave: . . . 6
- 2.3 Programa Desenvolvido e Resultados obtidos . . . 6

3 Cifra de *Affine* **8**
- 3.1 Cifragem . . . 8
- 3.2 Decifragem . . . 8
- 3.3 Programa Desenvolvido e Resultados obtidos . . . 8

4 Cifra de Substituição **10**
- 4.1 Cifragem . . . 10
- 4.2 Decifragem e Programa Desenvolvido . . . 10
- 4.3 Resultados obtidos . . . 11

5 Conclusão **12**

6 Bibliografia **13**

A Código do Programa **14**
- A.1 Vigenère . . . 14
- A.2 Affine . . . 19
- A.3 Substituição . . . 21

List of Figures

2.1 Tabela Vigenère 5

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito do primeiro Trabalho Prático da Unidade Curricular de Tecnologia Criptográfica, que se insere no 1º semestre do 4º ano do Mestrado Integrado em Engenharia Informática (1º ano MEI).

1.2 Objetivos e Trabalho Proposto

Para este trabalho foi proposta a criptoanálise de três criptogramas cifrados com cifras *affine*, de substituição e Vigenère. Para tal foi necessário o desenvolvimento de três programas em *python* que têm como objetivo decifrar e apresentar o texto-limpo de todos os criptogramas.

1.3 Estrutura do Relatório

O relatório encontra-se dividido em 5 secções. Na primeira secção é feita uma introdução e contextualização deste trabalho prático. A segunda secção aborda a cifra de Vigenère e a criptoanálise do texto cifrado com essa cifra. A secção 3 é relativa à cifra *affine* onde é apresentado o seu funcionamento junto com a explicação do que foi desenvolvido. Na secção 4 é abordada a cifra de susbtituição e relativa análise e decifragem do criptograma. Por último, na secção 5 é feita uma breve conclusão do trabalho realizado.

Cifra de Vigenère

A cifra de Vigenère é um método criptográdico que usa uma série de diferentes cifras de César. Na cifra de César, cada letra do alfabeto é deslocada da sua posição um número fixo de lugares, ou seja, se a deslocação for de 3, a letra A irá corresponder a D, B irá corresponder a E, e assim sucessivamente. A cifra de Vigenère consiste no uso de várias cifras de César em sequência mas com diferentes valores de deslocamento.

2.1 Cifragem

De modo a cifrar um texto-limpo usando esta cifra utiliza-se uma tabela de alfabetos que consiste no alfabeto escrito 26 vezes nas diferentes linhas mas sempre deslocando uma posição em relação à anterior. Consoante o texto-limpo e a chave escolhida, a cifragem ocorre procurando na tabela a letra que corresponde à interseção de uma letra do texto-limpo com uma letra da chave. Ou seja, caso tenhamos o texto-limpo "HELLO" e a chave "KEY", primeiramente a chave é transformada numa *keyString*, onde passa a ter o tamanho do texto-limpo: "KEYKE", posteriormente a primeira letra cifrada corresponde à interseção da primeira letra do texto-limpo "H" com a primeira letra da *keyString* "k", logo a primera letra cifrada será "R". Depois o processo é repetido para as restantes letras. A figura seguinte mostra a tabela utilizada para a cifragem:

	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

Figure 2.1: Tabela Vigenère

2.2 Decifragem

Existem vários métodos de decifrar uma cifra de Viginère. O método de decifragem que vai ser abordado é o método que foi depois utilizado no desenvolvimento do código em *python*. Caso se tenha conhecimento da chave, o método utilizado para decifrar consiste no método inverso ao da cifragem, ou seja, recorrendo à tabela da figura 2.1. Caso não se saiba a chave uma das téncias

centra-se em resolver estes 2 passos fundamentais: encontrar o tamanho da chave e encontrar cada número na chave, número este que corresponde ao deslocamento de cada letra.

2.2.1 Encontrar tamanho da chave:

Para determinar o tamanho da chave é necessário recorrer ao cálculo do índice de coincidência, criado em 1920 por William F. Friedman. Esta ideia baseia-se na análise de frequências de cada letra. Para tal, utiliza-se a seguinte fórmula:

$$\mathrm{IC} = \sum_{i=1}^{n} \frac{\mathrm{F_i}\,(\mathrm{F_i}-1)}{\mathrm{N}\,(\mathrm{N}-1)}$$

Em que F é a frequência do caractere, i é o número de ocorrências no criptograma e N é o número total de caracteres.

Sendo assim é necessário fazer a divisão do texto cifrado em vários grupos, em que para cada um desses grupos se calcula o índice de coincidência. O número do grupo que tiver o maior índice de coincidência será o tamanho da chave.

Para tal, é necessária a divisão de cada grupo em sub-grupos e o índice de coincidência do grupo final vai ser a média dos índices de coincidência dos sub-grupos. Para efetuar este cálculo calcula-se, para cada sub-grupo, o denominador seguido do numerador e posteriormente a divisão de ambos. De maneira a calcular o denominador, recorre-se ao somatório da multiplicação do número total de letras de cada sub-grupo (N) pelo *(N-1)*. O numerador de cada sub-grupo é calculado pelo somatório da multiplicação do número de vezes que uma determinada letra aparece (F) pelo (F-1). Após este cálculo e o armazenamento dos valores em por exemplo listas, é feita a média dos sub-grupos e obtido o índice de coincidência final de cada grupo. O maior destes respresenta o tamanho da chave.

2.2.2 Encontrar cada número na chave:

De maneira a descobrir cada número na chave, recorre-se à frequência das letras no alfabeto utilizado (neste caso alfabeto inglês). Como já se sabe o tamanho da chave, o texto cifrado é agrupado em várias *sub-strings* que são compostas pelas letras ou símbolos que aparecem na posição do tamanho da chave (notando que este tamanho vai decrementado para construir as outras substrings), ou seja, temos um texto cifrado "ABA;ABCC" e uma chave de comprimento 3, as *sub-strings* serão "A;C", "BAC" e "AB". Seguidamente é calculada a frequência de letras em cada substring. Isto faz-se somando o número de vezes que uma letra aparece a dividir pelo número total de letras nessa substring. Após isto, é necessário fazer uma série de cálculos onde, para cada *sub-string*, ocorre o somatório das frequências anteriores mutiplicadas pelas frequências do alfabeto. No fim deste somatório deparamos-nos com 26 frequências para cada *sub-string*. Se colocar-mos as frequências de uma *sub-string* numa lista com 26 elementos, o ínidice do maior elemento corresponde ao possível deslocamento da letra da chave. Imaginando como lista [0.2,0.4,0.6,0...], o ínidice do maior elemeto é 2, logo a letra corresponderia a um C. Assim, a partir do cálculo das frequências juntamente com as frequências do alfabeto inglês é possível chegar a uma chave. Após a descoberta da chave a decifragem ocorre utilizando o método inverso ao método da cifragem.

2.3 Programa Desenvolvido e Resultados obtidos

Como dito anteriormente, o programa desenvolvido baseia-se no método de decifragem explicado em cima. Foram portanto criadas 3 funções principais: uma para o cálculo do tamanho da chave,

outra para encontrar a chave e uma última para decifrar o texto com o tamanho e a chave. O código deste programa, que está devidamente comentado, encontra-se disponível nos anexos deste relatório.

Com a utilização das 3 funções chegamos à conclusão de que o tamanho da chave é de **5** elementos, a chave utilizada foi **DFLVY** e o texto-limpo é o seguinte:

```
THE DOCTRINE OF THE ORIGIN OF OUR SEVERAL DOMESTIC RACES FROM SEVERAL ABORIGINAL
STOCKS, HAS BEEN CARRIED TO AN ABSURD EXTREME BY SOME AUTHORS. THEY BELIEVE THAT
EVERY RACE WHICH BREEDS TRUE, LET THE DISTINCTIVE CHARACTERS BE EVER SO SLIGHT,
HAS HAD ITS WILD PROTOTYPE. AT THIS RATE THERE MUST HAVE EXISTED AT LEAST A SCORE
OF SPECIES OF WILD CATTLE, AS MANY SHEEP, AND SEVERAL GOATS, IN EUROPE ALONE, AND
SEVERAL EVEN WITHIN GREAT BRITAIN. ONE AUTHOR BELIEVES THAT THERE FORMERLY EXISTED
ELEVEN WILD SPECIES OF SHEEP PECULIAR TO GREAT BRITAIN! WHEN WE BEAR IN MIND THAT
BRITAIN HAS NOW NOT ONE PECULIAR MAMMAL, AND FRANCE BUT FEW DISTINCT FROM THOSE OF
GERMANY, AND SO WITH HUNGARY, SPAIN, ETC., BUT THAT EACH OF THESE KINGDOMS POSSESSES
SEVERAL PECULIAR BREEDS OF CATTLE, SHEEP, ETC., WE MUST ADMIT THAT MANY DOMESTIC
BREEDS MUST HAVE ORIGINATED IN EUROPE; FOR WHENCE OTHERWISE COULD THEY HAVE BEEN
DERIVED? SO IT IS IN INDIA. EVEN IN THE CASE OF THE BREEDS OF THE DOMESTIC DOG
THROUGHOUT THE WORLD, WHICH I ADMIT ARE DESCENDED FROM SEVERAL WILD SPECIES, IT
CANNOT BE DOUBTED THAT THERE HAS BEEN AN IMMENSE AMOUNT OF INHERITED VARIATION;
FOR WHO WILL BELIEVE THAT ANIMALS CLOSELY RESEMBLING THE ITALIAN GREYHOUND, THE
BLOODHOUND, THE BULL-DOG, PUG-DOG, ORBLENHEIM SPANIEL, ETC.| MEUNLIKE ALL WILD
CANIDAE|EVER EXISTED IN A STATE OF NATURE?
IT HAS OFTEN BEEN LOOSELY SAID THAT ALL OUR RACES OF DOGS HAVE BEEN PRODUCED BY
THE CROSSING OF A FEW ABORIGINAL SPECIES; BUT BY CROSSING WE CAN ONLY GET FORMS
IN SOME DEGREE INTERMEDIATE BETWEEN THEIR PARENTS; AND IF WE ACCOUNT FOR OUR SEVERAL
DOMESTIC RACES BY THIS PROCESS, WE MUST ADMIT THE FORMER EXISTENCE OF THE MOST EXTREME
FORMS, AS THE ITALIAN GREYHOUND, BLOODHOUND, BULL-DOG, ETC., IN THE WILD STATE.
MOREOVER, THE POSSIBILITY OF MAKING DISTINCT RACES BY CROSSING HAS BEEN GREATLY
EXAGGERATED. MANY CASES ARE ON RECORD SHOWING THAT A RACE MAY BE MODIFIED BY OCCASIONAL
CROSSES IF AIDED BY THE CAREFUL SELECTION OF THE INDIVIDUALS WHICH PRESENT THE DESIRED
CHARACTER; BUT TO OBTAIN A RACE INTERMEDIATE BETWEEN TWO QUITE DISTINCT RACES WOULD BE
VERY DIFFICULT. SIR J. SEBRIGHT EXPRESSLY EXPERIMENTED WITH THIS OBJECT AND FAILED.
THE OFFSPRING FROM THE FIRST CROSS BETWEEN TWO PURE BREEDS IS TOLERABLY AND SOMETIMES
(AS I HAVE FOUND WITH PIGEONS) QUITE UNIFORM IN CHARACTER, AND EVERY THING SEEMS SIMPLE
ENOUGH; BUT WHEN THESE MONGRELS ARE CROSSED ONE WITH ANOTHER FOR SEVERAL GENERATIONS,
HARDLY TWO OF THEM ARE ALIKE, AND THEN THE DIFFICULTY OF THE TASK BECOMES MANIFEST.
```

Cifra de *Affine*

A cifra de *Affine* é um tipo de cifra de substituição monoalfabética, ou seja, a cada letra do alfabeto corresponde um número (a=0,b=1,...z=25). Esta cifra tem a particularidade de recorrer a uma função matemática simples: $y = \alpha$ x + β , onde y é o número que representa a letra do texto-cifrado, x é o número que representa a letra do texto-limpo e (α, β) são números que representam a chave da cifra.

3.1 Cifragem

De modo a cifrar um texto-limpo é escolhida uma chave (α, β) e é utilizada a função anteriormente apresentada ($y = \alpha$ x + β) para cada letra do texto. No entanto, é necessário colocar o módulo do número de letras do alfabeto (neste caso mod 26) pois o cálculo de y não pode ser superior a este número.

3.2 Decifragem

Existem essencialmente duas maneiras de decifrar um cifra de *Affine*. A primeira consiste no chamado *Plain Text Attack*. Neste ataque um invasor, para além de ter conhecimento do criptograma, tem de conhecer pelo menos 2 letras do texto-limpo. Se ele obtiver esta informação consegue facilmente, através da fórmula, chegar ao texto limpo. Para tal basta aplicar a fórmula em ordem a x, ou seja, $x = \alpha^{-1}(y\text{-}\beta)$, onde α^{-1} é a aplicação do algoritmo de Euclides a α. Neste caso é também necessário se ter atenção ao valor de x. para este não passar o tamanho do alfabeto. Para tal é necessário recorrer ao cálculo do módulo novamente.
A segunda maneira, que foi a utilizada para o desenvolvimento do código em *python*, consiste em fazer *brute force*. Uma vez que qualquer palavra do texto cifrado é desconhecida, uma maneira de descobrir duas letras para depois se calcular a chave, consiste em calcular as frequências das letras no texto-cifrado e depois comparar com as frequências das letras no alfabeto inglês. Ou seja, descobrir e guardar quais as duas letras que aparecem mais no texto cifrado. Após isto, comparar com o alfabeto utilizado, isto é, a maior frequência do alfabeto inglês corresponde à letra *E*, a segunda à letra *T*, pelo que podemos utilizar estas duas letras para o cálculo da chave com as duas letras que aparecem mais no criptograma. Para o cálculo da chave é necessário resolver um sistema com duas equações. No fim ficamos a saber os valores de α e β. Seguidamente, utilizar a fórmula em ordem a x, $x = \alpha^{-1}(y\text{-}\beta)$, e verificar se o texto está completamente decifrado. Caso a chave encontrada não seja a adequada, é necessário repetir o processo mas com as outras letras mais frequentes no alfabeto.

3.3 Programa Desenvolvido e Resultados obtidos

Como dito anteriormente, o programa desenvolvido baseia-se no método de decifragem *brute force*. Foram portanto criadas 2 funções principais: uma para o cálculo da chave e outra para decifrar o texto a partir dessa chave. Como para o cálculo da chave é necessária a análise das

frequências do alfabeto inglês, a primeira combinação, ou seja, a letra que mais aparece no criptograma, juntou-se com a letra E e a segunda com letra T. Isto não funcionou uma vez que o texto-limpo gerado não era o correto. Ao trocar ambas as letras, letra T corresponde à letra com maior frequência no texto-cifrado e E à segunda maior frequência, a chave calculada foi a adequada uma vez que o texto-limpo ficou percetível. O código deste programa, que está devidamente comentado, encontra-se disponível nos anexos deste relatório.

Com a utilização destas funções chegamos à conclusão que a chave é **(11,8)** e o texto-limpo é:

```
IT IS NOT TOO MUCH TO REQUIRE THAT WHAT THE WISEST OF MANKIND, THOSEEWHO ARE BEST
ENTITLED TO TRUST THEIR OWN JUDGMENT, FIND NECESSARY TOEWARRANT THEIR RELYING ON IT,
SHOULD BE SUBMITTED TO BY THATEMISCELLANEOUS COLLECTION OF A FEW WISE AND MANY FOOLISH
INDIVIDUALS,ECALLED THE PUBLIC. THE MOST INTOLERANT OF CHURCHES, THE ROMAN
CATHOLICECHURCH, EVEN AT THE CANONISATION OF A SAINT, ADMITS, AND LISTENSEPATIENTLY TO,
A "DEVIL'S ADVOCATE." THE HOLIEST OF MEN, IT APPEARS,ECANNOT BE ADMITTED TO POSTHUMOUS
HONOURS, UNTIL ALL THAT THE DEVILECOULD SAY AGAINST HIM IS KNOWN AND WEIGHED. IF EVEN
THE NEWTONIANEPHILOSOPHY WERE NOT PERMITTED TO BE QUESTIONED, MANKIND COULD NOT FEELEAS
COMPLETE ASSURANCE OF ITS TRUTH AS THEY NOW DO. THE BELIEFS WHICHEWE HAVE MOST WARRANT
FOR, HAVE NO SAFEGUARD TO REST ON, BUT A STANDINGEINVITATION TO THE WHOLE WORLD TO PROVE
THEM UNFOUNDED. IF THEECHALLENGE IS NOT ACCEPTED, OR IS ACCEPTED AND THE ATTEMPT FAILS,
WEEARE FAR ENOUGH FROM CERTAINTY STILL; BUT WE HAVE DONE THE BEST THATETHE EXISTING STATE
OF HUMAN REASON ADMITS OF; WE HAVE NEGLECTEDENOTHING THAT COULD GIVE THE TRUTH A CHANCE
OF REACHING US: IF THEELISTS ARE KEPT OPEN, WE MAY HOPE THAT IF THERE BE A BETTER TRUTH,
ITEWILL BE FOUND WHEN THE HUMAN MIND IS CAPABLE OF RECEIVING IT; AND INETHE MEANTIME WE
MAY RELY ON HAVING ATTAINED SUCH APPROACH TO TRUTH, ASEIS POSSIBLE IN OUR OWN DAY. THIS
IS THE AMOUNT OF CERTAINTY ATTAINABLEEBY A FALLIBLE BEING, AND THIS THE SOLE WAY OF
ATTAINING IT.
```

Cifra de Substituição

4.1 Cifragem

Para a cifragem do texto limpo é necessário escolher uma chave com o tamanho fixo do alfabeto. Essa chave define um mapeamento para cada letra do alfabeto, para que depois se faça as permutações e se substitua no texto limpo.

4.2 Decifragem e Programa Desenvolvido

A cifra de substituição é um método de encriptação em que a chave define um mapa para cada letra do alfabeto, em consequência o tamanho da nossa chave será sempre 26.

Para decifrar o texto cifrado decidimos fazer uma análise de frequências do texto e comparar com uma tabela de frequências do alfabeto inglês. Para isso contamos o número de letras do texto cifrado e comparamos com a tabela de frequências. Os valores obtidos foram os seguintes: *E=0, W=0, P=3, Y=4, F=13, S=14, K=15, A=16, Z=22, J=23, R=25, B=27, N=27, O=33, L=35, M=38, Q=54, T=67, C=76, V=83, I=89, U=90, H=95, G=118, X=128.*

Como podemos reparar a letra X e G são os caracteres com o maior número de incidências no texto cifrado. Decidimos substituir o X por E e o G por T, uma vez que são as duas letras com maior frequência em inglês. As restantes letras também foram substituídas pelas letras da tabela de frequência mas como os valores tem uma discrepância muito pequena, os valores vão ser alterados pois estas podem induzir em erro.

Depois decidimos comparar os trigramas obtidos com a tabela de frequências de trigramas. Os nossos valores foram os seguintes: *NRG=1, OUS=1, UOO=1, UTX=1, JRG=1, XCL=1, ISC=1, QUL=1, GII=1, CIT=1, ZIT=1, UCA=2, NUL=2, SQI=2, OXG=2, KIG=2, MUC=2, HGV=2, CIG=3, QUV=3, UCL=5, GQX=11.*

Reparámos que o trigrama *GQX* é o que aparece mais vezes e o trigrama que tem a maior frequência na tabela é o *THE*, então procedemos à alteração. Depois fomos ao próximo trigrama o *UCL* e procedemos à alteração com a segunda maior frequência da tabela de trigramas o *AND*. Os restantes trigramas não foram substituídos pelos trigramas das tabelas de frequência porque os valores tem uma discrepância muito pequena, podendo induzir em erro.

A seguir substitui-mos o *TI* para *TO* para que faça sentido. Depois o *DANNOT ADMAT* para *CANNOT ADMIT*, portanto o D por C, o D estava na posição do M do alfabeto. E o A por I, o A estava na posição do Y do alfabeto.

A seguir mudamos *COMMWNITY* para *COMMUNITY*, a substituição foi feita na posição R do alfabeto. Depois o *OTHEH* para *OTHER* a substituição do H para o R foi feita na posição T do alfabeto. O *INVOJE* para *INVOKE*, substitui-mos o *ABARE* para *AWARE*, *AFFEAR* para *APPEAR*, *TO UE AGRAID* para *TO BE AFRAID*, *APPEARN* para *APPEARS*, *CIVICISED* para *CIVILISED*, a substituição foi feita na posição O do alfabeto. O *REKUIRE* para *REQUIRE*, a substituição foi feita na posição Y do alfabeto. E por fim *AMONP* por *AMONG*, a substituição foi feita na posição K do alfabeto.

4.3 Resultados obtidos

O programa desenvolvido serviu para contar frequências das letras e sequência de duas ou mais letras consecutivas como bigramas e trigramas, para que depois se possa comparar com uma tabela de frequências e proceder às permutações da chave com o alfabeto. Também se foi procedendo a permutações de modo a formar palavras que façam sentido como *COMMWNITY* para *COMMUNITY*. Com a utilização destas funções chegamos à conclusão que a chave é **YMNXJVTIOPGDCBLKHUWRASZEQF** e o texto-limpo é:

```
IT ALSO APPEARS SO TO ME, BUT I AM NOT AWARE THAT ANY COMMUNITY HAS A RIGHT TO FORCE
ANOTHER TO BE CIVILISED. SO LONG AS THE SUFFERERS BY THE BAD LAW DO NOT INVOKE
ASSISTANCE FROM OTHER COMMUNITIES, I CANNOT ADMIT THAT PERSONS ENTIRELY UNCONNECTED
WITH THEM OUGHT TO STEP IN AND REQUIRE THAT A CONDITION OF THINGS WITH WHICH ALL WHO
ARE DIRECTLY INTERESTED APPEAR TO BE SATISFIED, SHOULD BE PUT AN END TO BECAUSE IT IS
A SCANDAL TO PERSONS SOME THOUSANDS OF MILES DISTANT, WHO HAVE NO PART OR CONCERN IN
IT. LET THEM SEND MISSIONARIES, IF THEY PLEASE, TO PREACH AGAINST IT; AND LET THEM,
BY ANY FAIR MEANS (OF WHICH SILENCING THE TEACHERS IS NOT ONE), OPPOSE THE PROGRESS
OF SIMILAR DOCTRINES AMONG THEIR OWN PEOPLE. IF CIVILISATION HAS GOT THE BETTER OF
BARBARISM WHEN BARBARISM HAD THE WORLD TO ITSELF, IT IS TOO MUCH TO PROFESS TO BE
AFRAID LEST BARBARISM, AFTER HAVING BEEN FAIRLY GOT UNDER, SHOULD REVIVE AND CONQUER
CIVILISATION. ACIVILISATION THAT CAN THUS SUCCUMB TO ITS VANQUISHED ENEMY, MUST
FIRST HAVE BECOME SO DEGENERATE, THAT NEITHER ITS APPOINTED PRIESTS AND TEACHERS,
NOR ANYBODY ELSE, HAS THE CAPACITY, OR WILL TAKE THE TROUBLE, TO STAND UP FOR IT.
IF THIS BE SO, THE SOONER SUCH A CIVILISATION RECEIVES NOTICE TO QUIT, THE BETTER.
IT CAN ONLY GO ON FROM BAD TO WORSE, UNTIL DESTROYED AND REGENERATED (LIKE THE
WESTERN EMPIRE) BY ENERGETIC BARBARIANS.
```

Conclusão

Com este trabalho prático foi possível pôr em prática vários conhecimentos obtidos durante as aulas de Tecnologia Criptográfica, nomeadamente sobre os três tipos de decifragem e cifragem apresentados.

Podemos concluir que a utilização de qualquer um destes três métodos criptográficos (*Vigenère*, *Affine* e Substituição) não são a melhor escolha na implementação de um sistema informático uma vez que são facilemente decifrados, nomeadamente através de análise de frequências.

Bibliografia

https://www3.nd.edu/ busiforc/handouts/cryptography/Letter

Código do Programa

A.1 Vigenère

```
import collections
import re

class IC:
    def __init__(self, groupNº, IC, groupLetters):
        self.groupNº = groupNº+2
        self.IC = IC
        self.groupLetters = groupLetters

english_frequences = [0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015,
                      0.06094, 0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749,
                      0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758,
                      0.00978, 0.02360, 0.00150, 0.01974, 0.00074]

alphabet = ['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']

def getList(dict):
    return dict.keys()

def get_indexes_max_value(l):
    max_value = max(l)
    if l.count(max_value) > 1:
        return [i for i, x in enumerate(l) if x == max(l)]
    else:
        return l.index(max(l))

def get_length_key(cypherText):
    maxGroup = 1
    group = 2
    newString = ""
    my_list = []
    listAux = []
    indexAux = 0
    indexIteration = 0

    # separar por grupos
    # group começa por criar 2 grupos, max group o numero de iterações,
    # para poder adicionar ao group e assim fazer um grupo maior ex..(2,3,4)
```

```
# o index interaction percorre o numero de grupos maximo para o contexto
# por exemplo se tiver que fazer o grupo= 3 conta as 3 iterações
# indexAux escolhe em que letra começa a contar/adicionar
while maxGroup < 8:
    newString = ""
    while indexIteration < group:
        newString = ""
        for index, item in enumerate(cypherText):
            # make groups
            if 0 == (index + (indexAux - 1)) % (maxGroup + 1):
                newString += item

        listAux.insert(indexAux, newString)
        indexAux = indexAux + 1
        indexIteration = indexIteration + 1

    indexIteration = 0
    group = group + 1
    my_list.insert((maxGroup - 1), listAux)
    maxGroup += 1
    listAux = []

# index of coincidence get lnght keyword

##count letters
genericCountList = []
letterCountList = []
letterCountListDict = {}
countMaxLetter = 0
letterFrequency = None
index2Aux = 0
totalLetersGroups = []

for index, item in enumerate(my_list):
    for index2, item2 in enumerate(item):
        letterCountListDict = {}
        letterFrequency = collections.Counter(item2)
        for itemL, letterCount in letterFrequency.items():
            letterCountListDict[itemL] = letterCount
        letterCountList.insert(index2Aux, letterCountListDict)
        index2Aux = index2Aux + 1
    index2Aux = 0

    genericCountList.insert(index, letterCountList)
    letterCountList = []

ICList = []
listGroupIC = []
listGroupICTemp = []
countLetter = 0
letter = []
```

```
    icMultUp = 0
    icMultDown = 0
    icFinalG = 0
    indexGroup = 0
    groupAverage = 0

    # count letters in group
    for index, item in enumerate(genericCountList):
        for index2, item2 in enumerate(item):
            for key, val in item2.items():
                countLetter += val
                icMultUp += (val * (val - 1))
            # compute IC individualy
            icMultDown = countLetter * (countLetter - 1)
            icFinalG = icMultUp / icMultDown
            listGroupICTemp.insert(index2, icFinalG)
            icMultUp = 0
            icMultDown = 0
            countLetter = 0
            icFinalG = 0
        # average of the groups
        for item3 in listGroupICTemp:
            groupAverage += item3
        groupAverage = groupAverage / listGroupICTemp.__len__()
        listGroupIC.append(groupAverage)
        groupAverage = 0
        listGroupICTemp = []

    listGroupICAux = []
    # save results in class list
    for index, ic in enumerate(listGroupIC):
        listGroupICAux.append(IC(index, ic, my_list.__getitem__(index)))
    icSave = IC(0, 0, 0)
    for group in enumerate(listGroupICAux):
        if (icSave.IC < group.__getitem__(1).IC):
            icSave = IC(group.__getitem__(1).groupNº-2, group.__getitem__(1).IC,
            group.__getitem__(1).groupLetters)
    return icSave.groupNº


def get_key(ciphertext, keylength):
    list = []
    j = 0
    x = 0
    indice = 0
    for counter in range(keylength, 0, -1):
        letter_count_list_dict = {}
        string = ""
        # cipher é a substring do ciphertext
        cipher = ciphertext[x:len(ciphertext)]
        # ciclo que permite andar pelos índices corretos
```

```
    for index, item in enumerate(cipher):
        if 0 == index % keylength:
            string += item
    letter_frequency = collections.Counter(string)
    # conta e coloca no dicionário o número de letras da string
    for itemL, letterCount in letter_frequency.items():
        letter_count_list_dict[itemL] = letterCount
    lista = getList(letter_count_list_dict)
    # coloca na lista y quantas vezes uma certa letra aparece
    # (indice 0 corresponde à letra A)
    y = []
    for u in alphabet:
        if lista.__contains__(u):
            y.insert(index, letter_count_list_dict.get(u))
            indice += 1
        else:
            y.insert(index, 0)
            indice += 1
    list.insert(j, y)  #concatenação das listas y [[3,4,0,0...],[7,0,8,...]]
                       # na primeira lista 3A 4B, segunda lista 7A, 8C
    j += 1
    x += 1
lista_freqs_total = []
p = 0
# percorre a lista de listas e calcula a frequência de cada letra no seu grupo
for l in list:
    list_freqs = []
    k = 0
    while k < 26:
        if l[k] > 0:
            list_freqs.insert(k, l[k]/sum(l))
            k += 1
        else:
            list_freqs.insert(k, 0)
            k += 1

    lista_freqs_total.insert(p, list_freqs)
    p += 1
list_shifts_total = []

# multiplica a lista das frequências de grupo pela lista de frequências inglesas
# e realiza a soma de todas as multiplicações
j =0
for l in lista_freqs_total:
    shift=0
    list_shifts = []
    while shift < 26:
        soma = 0
        i = 0
        while i < 26:
            soma += l[getCircular(i+shift,26)]*english_frequences[i]
```

```
                i += 1
            list_shifts.insert(shift, soma)
            shift+=1
        list_shifts_total.insert(j,list_shifts)
        j+=1

    # calcula os índices que possuem os elementos maiores da lista
    lista_max_shifts = []
    i = 0
    for l in list_shifts_total:
        shift = get_indexes_max_value(l)
        lista_max_shifts.insert(i, shift)
        i += 1

    # através da lista dos índices calcula a letra da chave
    # (a lista dos índices tem o número de shifts)
    key = ""
    for i in lista_max_shifts:
        key += alphabet[i]
    return key

def getCircular(i,size):
    ret = (i)%size
    return ret

def vigenere_decrypt(encrypted_vigener, keyword):
    keyword_length = len(keyword)
    keyword_as_int = [ord(i) for i in keyword]
    encrypted_vigener_int = [ord(i) for i in encrypted_vigener]
    plaintext = ''
    for i in range(len(encrypted_vigener_int)):
        if encrypted_vigener[i].isalpha():
            value = (encrypted_vigener_int[i] - keyword_as_int[i % keyword_length]) % 26
            plaintext += chr(value + 65)
        else:
            plaintext += encrypted_vigener[i]
    return plaintext

if __name__ == '__main__':
  f = open("cript1", "r")
  length = get_length_key(f.read())
  f = open("cript1", "r")
  key = get_key(f.read(),length)
  f = open("cript1", "r")
  string = vigenere_decrypt(f.read(),key)
  file = open("PlaintexVigenere.txt", "w")
  file.write(string)
  file.close()
```

A.2 Affine

```
import collections

def getList(dict):
    return dict.keys()

# Aplica o Algoritmo de Euclides
def modInverse(a, m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return 1

# devolve o segundo maior elemento da lista
def get_second_max(list1):
    mx = max(list1[0], list1[1])
    secondmax = min(list1[0], list1[1])
    n = len(list1)
    u = 0
    for i in range(2, n):
        if list1[i] > mx:
            secondmax = mx
            mx = list1[i]
        elif secondmax < list1[i] != mx:
            secondmax = list1[i]
    return secondmax

def getCircular(i, size):
    ret = i % size
    return ret

alphabet = ['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']

lista_pontuacao = [" ", "!", ",", ".", ";", "?", "\"", "\'", ":"]

english_frequences = [8.12, 1.49, 2.71, 4.32, 12.0, 2.30, 2.03, 5.92, 7.31, 0.10, 0.69,
                      3.98, 2.61, 6.95, 7.68, 1.82, 0.11, 6.02, 6.28, 9.10, 2.88, 1.11,
                      2.09, 0.11, 2.11, 0.07]

def analisar_frequencias(ciphertext):
    list = []
    letter_count_list_dict = {}
    string = ""
    # ciclo que permite andar pelos índices corretos
    for index, item in enumerate(ciphertext):
        if 0 == index % 1:
            string += item
```

```
    letter_frequency = collections.Counter(string)
    # conta e coloca no dicionário o número de letras da string
    for itemL, letterCount in letter_frequency.items():
        letter_count_list_dict[itemL] = letterCount

    lista = getList(letter_count_list_dict)
    list = []
    index = 0
    # insere o número de vezes que uma letra aparece numa lista de 26 elementos
    # em que o índice 0 corresponde ao A, 1 ao B...
    for u in alphabet:
        if lista.__contains__(u):
            list.insert(index, letter_count_list_dict.get(u))
            index += 1
        else:
            list.insert(index, 0)
            index += 1

    list_freqs = []
    k = 0
    # calcula a frequência com que cada letra aparece no criptograma
    while k < 26:
        if list[k] > 0:
            list_freqs.insert(k, list[k] / sum(list))
            k += 1
        else:
            list_freqs.insert(k, 0)
            k += 1
    first_letter = list_freqs.index(max(list_freqs)) # letra que aparece + no texto cifrado
    second_letter = list_freqs.index(get_second_max(list_freqs)) # 2ª letra que aparece +
    # letra com maior frequência alfabeto inglês:
    letra = english_frequences.index(max(english_frequences))
    english_frequences.insert(english_frequences.index(max(english_frequences)), 0)
    english_frequences.remove(max(english_frequences))
    # segunda letra com maior frequência alfabeto inglês:
    segunda_letra = english_frequences.index(max(english_frequences))
    #resolução do sistema de equações:
    x = (segunda_letra - letra) % 26
    y = (first_letter - second_letter) % 26
    alpha = (y * modInverse(x, 26)) % 26
    beta = (first_letter - segunda_letra * alpha) % 26
    key = (alpha, beta)
    return key

def number_letter(letter):
    i = 0
    if alphabet.__contains__(letter):
        i = alphabet.index(letter)
    return i
```

```
# percorre o texto cifrado e atavés da chave e da fórmula, para cada letra calcula
# o número da letra do texto-limpo
def find_plaintext(alpha, beta, ciphertext):
    string = ""
    for c in ciphertext:
     if lista_pontuacao.__contains__(c): #coloca a pontuação e os espaços nos sítios certos
            string += c
        else:
            x = modInverse(alpha, 26) * (number_letter(c) - beta)
            modulo = x % 26
            string += alphabet[modulo] #vai ao alfabeto buscar a letra pelo índice
    return  string

if __name__ == '__main__':
    f = open("cript2", "r")
    alpha, beta = analisar_frequencias(f.read())
    f = open("cript2", "r")
    string = find_plaintext(alpha, beta, f.read())
    file = open("PlaintextAffine.txt", "w")
    file.write(string)
    file.close()
```

A.3 Substituição

```
import string
def SubstitutionCipher():
    alphabet = dict.fromkeys(string.ascii_uppercase, 0)
    nWords = 0
    f = open("cript3", "r")
    cipherText = f.read();
    #contar o numero de letras do cipher text
    for i in cipherText:
        if i in alphabet:
            alphabet[i] = alphabet[i] + 1
            nWords += 1

    bigrams=[]
    #for i, letter in cipherText
    #dividir as palavras numa lista
    for i in cipherText.split():
        if i.isalpha():
            bigrams.append(i)


    trigrams=bigrams.copy()
    i=0
    #criar remover todas as plavars com mais e menos de duas letras para criar bigramas
    while(len(bigrams)>i):
        if(len(bigrams[i]) >2  or len(bigrams[i]) <=1):
           bigrams.remove(bigrams[i])
```

```
        i=0
    i = i+1


auxBi = ""
# to remove duplicated of bigrams
# from list
res = []
for i in bigrams:
    if i not in res: #adiciona se não estiver na lista res
        res.append(i)

#trasnformar os bigramas num dicionario
cpyBi2 = dict.fromkeys(res,0)

#count number of bigrams
nw=0
for i in bigrams:
    if i in cpyBi2:
       cpyBi2[i] = cpyBi2[i] + 1
       nw += 1


sortedCipherBig = sorted(cpyBi2.items(), key=lambda kv: kv[1])

#trigrams
#  remover todas as plavars com mais e menos de 3 letras para criar trigramas
i=0
while (len(trigrams) > i):
    if (len(trigrams[i]) > 3 or len(trigrams[i]) < 3):
        trigrams.remove(trigrams[i])
        i = 0
    else:
        i = i + 1

#remove duplicades
removeDupli = []
for i in trigrams:
    if i not in removeDupli:
        removeDupli.append(i)

#convert to dict
cpytr = dict.fromkeys(removeDupli,0)
# count number of trigrams
for i in trigrams:
    if i in cpytr:
        cpytr[i] = cpytr[i] + 1

sortedCiphertri = sorted(cpytr.items(), key=lambda kv: kv[1])
```

```
    key="YMNXJVTIOPGDCBLKHUWRASZEQF"
    index=0
    tenta=cipherText
    for indw, al in enumerate(cipherText):
        if(alphabet.__contains__(al)):
            tenta =  tenta[:indw] + key[(ord(al)-65)] + tenta[indw + 1:]

    file = open("PlaintexSubstituicao.txt", "w")
    file.write(tenta)
    file.close()



if __name__ == '__main__':
    SubstitutionCipher()
```