

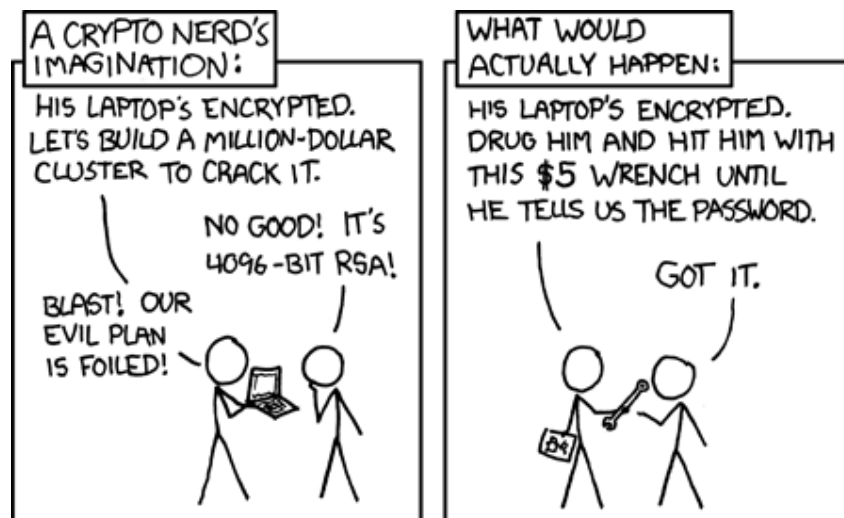


OPDRACHT 3

DANILO DE GOEDE[‡] - JULIAN VERWEIJ[†]

[‡] UNIVERSITY OF AMSTERDAM - [†] UNIVERSITY OF OXFORD

Vigenère



Deadline: zaterdag 25 september 2021, 23:59

1 Introductie

In de afgelopen weken heb je relatief korte programma's geschreven. In deze opdracht ga je een groter programma schrijven waarbij je een gegeven framework uitbreidt. Dit programma stelt de gebruiker in staat een bericht naar keuze te versleutelen en ontsleutelen met het *Vigenère cipher*¹. Dit klassieke handcijfer werd in de 16de eeuw bedacht door *Giovan Battista Bellaso*.

Langer dan 300 jaar dacht men dat dit cijfer onbreekbaar was, totdat *Friedrich Kasiski* in de 19de eeuw een methode publiceerde voor het breken van het Vigenère cipher¹. In het tweede deel van deze opdracht ga je een dergelijke methode implementeren om de gebruiker in staat te stellen een cryptogram te *kraken* door de onbekende sleutel van een cryptogram te achterhalen.

Het op te leveren programma bestaat uit (1) een implementatie van versleuteling en ontsleuteling volgens het Vigenère cipher, en (2) de hierboven genoemde aanval op het Vigenère cipher.

2 Deel 1: Implementatie van het Vigenère cipher

Voordat we het Vigenère cipher behandelen, kijken we eerst naar een eenvoudigere variant, het welbekende *Caesar cipher*. Bij het versleutelen van een bericht worden de karakters drie posities *verschoven*: een 'a' wordt een 'd', een 'b' wordt een 'e', een 'z' wordt een 'c', et cetera (zie Tabel 1).

a	b	c	d	e	f	g	h	i	j	k	l	m	...	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	...	z	a	b	c

Tabel 1: Versleuteling van individuele karakters met het Caesar cipher.

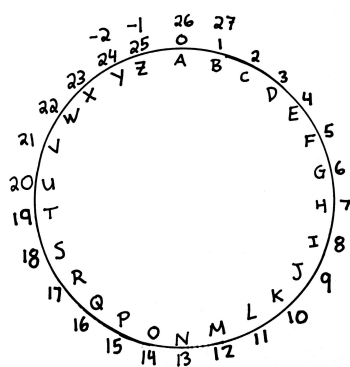
Deze operatie kan ook op een wiskundige manier worden beschreven. Hiertoe beschouwen we de karakters als getallen zoals geïllustreerd in Figuur 1. Wanneer we het te versleutelen woord m beschouwen en het i -de karakter met m_i aanduiden, is het i -de karakter in het cryptogram c gedefinieerd als

$$c_i = m_i + 3 \mod 26, \quad (1)$$

waarbij de modulo operatie (mod) de rest na een deling geeft (bijv. $31 \mod 26 = 5$). Dit is te interpretern als een *wrap-around* of klok, zie hiervoor Figuur 1. Om een cryptogram vervolgens te *ontsleutelen* passen we de inverse operatie

$$c_i = m_i - 3 \mod 26 \quad (2)$$

toe.



Figuur 1: Illustratie van de modulo-operatie met een klok.

Bij het Vigenère cipher worden de letters ook op deze manier verschoven, alleen wordt niet elke letter hetzelfde aantal posities doorgeschoven. In plaats daarvan wordt een *sleutel* gebruikt die bepaalt hoe ver elke letter moet worden doorgeschoven.

¹ https://en.wikipedia.org/wiki/Vigenere_cipher

De sleutel is een woord of korte zin. Net als bij het bericht zelf representeren de karakters van de sleutel een getal van 0 tot en met 25 zoals geïllustreerd in Figuur 1. De eerste letter van de sleutel bepaalt op die manier hoe ver de eerste letter van het bericht moet worden doorgeschoven, de tweede letter van de sleutel bepaalt hoe ver de tweede letter van het cryptogram moet worden doorgeschoven, et cetera. Deze sleutel herhaalt zich totdat de lengte van het bericht bereikt is.

Als voorbeeld nemen we de geheime sleutel ‘*collatz*’ en versleutelen we het bericht ‘*One who shaves all those, and those only, who do not shave themselves.*’. Gezien onze encoding alleen kleine letters uit het alfabet ondersteunt, zetten we alle hoofdletters om naar kleine letters en filteren we alle andere tekens weg. Het overgebleven bericht is dan ‘*onewhoshavesallthoseandthoseonlywhodonotshavethemselves*’. De versleuteling hiervan is dan als volgt:

bericht	:	onewhos	havesal	lthosea	ndthose	onlywho	donotsh	avethem	selves
sleutel	:	collatz	collatz	collatz	collatz	collatz	collatz	collatz	collat

cryptogram	:	qbphhhr	jogpstk	nhszszx	presold	qbwjwan	fcyztlg	cjpehxl	uswgel

Implementeer op basis van bovenstaande theorie de volgende functies:

- **strip**

Implementeer de functie *strip* die hoofdletters in de input-string naar kleine letters omzet en niet-alfabetische karakters uit de input-string verwijdert:

```
void strip(char src[], char dest[])
```

Het argument *src* bevat het oorspronkelijke bericht en het resulterende gefilterde bericht dient weg te worden geschreven in *dest*.

- **encrypt**

Implementeer de functie *encrypt* zoals in de tekst hierboven beschreven:

```
void encrypt(char plaintext[], char key[], char ciphertext[])
```

Het argument *plaintext* bevat het **ongefilterde** te versleutelen bericht, *key* bevat de geheime sleutel en het resulterende cryptogram dient weg te worden geschreven in *ciphertext*.

- **decrypt**

Implementeer de functie *decrypt*, die de omgekeerde operatie uitvoert:

```
void decrypt(char ciphertext[], char key[], char plaintext[])
```

Het argument *ciphertext* bevat het te ontsleutelen cryptogram, *key* bevat de geheime sleutel en het resulterende bericht dient weg te worden geschreven in *plaintext*.

Om makkelijk te kunnen werken met grotere berichten is het handig als berichten of cryptogrammen samen met hun sleutel in een bestand kunnen worden opgeslagen. Een bestand bestaat uit twee of drie regels. De eerste is altijd een *operation*: “ENCRYPT”, “DECRYPT” of “CRACK”. De tweede regel is het betreffende cryptogram of bericht. Als de opdracht “ENCRYPT” of “DECRYPT” is, is de derde regel de betreffende sleutel. We illustreren deze structuur voor elk van de operaties aan de hand van het eerder beschreven voorbeeld:

ENCRYPT
onewhoshavesallthoseandthoseonlywhodonotshavethemselves
collatz

DECRYPT
qbphhhrjogpstknhszszxpresoldqbwjwanfcyztlgcjpehxluswgel
collatz

CRACK
qbphhhrjogpstknhszszxpresoldqbwjwanfcyztlgcjpehxluswgel

- **read_file**

Implementeer de functie `read_file` die een bestand inleest, zodat later de opdracht beschreven in het bestand uitgevoerd kan worden (deze functie voert de operatie zelf niet uit).

```
int read_file(char pathname[], char input[], char key[])
```

Het argument *pathname* bevat het pad van het in te lezen bestand, het bericht of cryptogram dient weg te worden geschreven in *input*, en afhankelijk van de operatie dient de geheime sleutel weg te worden geschreven in *key*.

De functie geeft ook een waarde terug, die aangeeft welke operatie er in het bestand werd gespecificeerd. De mogelijke waarden kun je halen uit de enum *operation* in de meegeleverde code. In het geval van foutieve invoer moet de waarde is *ERROR* uit die enum worden teruggegeven.

Tip Alle arrays in de functieargumenten zijn groot genoeg gealloceerd (lengte *MAX_INPUT_SIZE*).

Tip Gebruik de meegeleverde functie *mod* voor de implementatie van *encrypt* en *decrypt*.

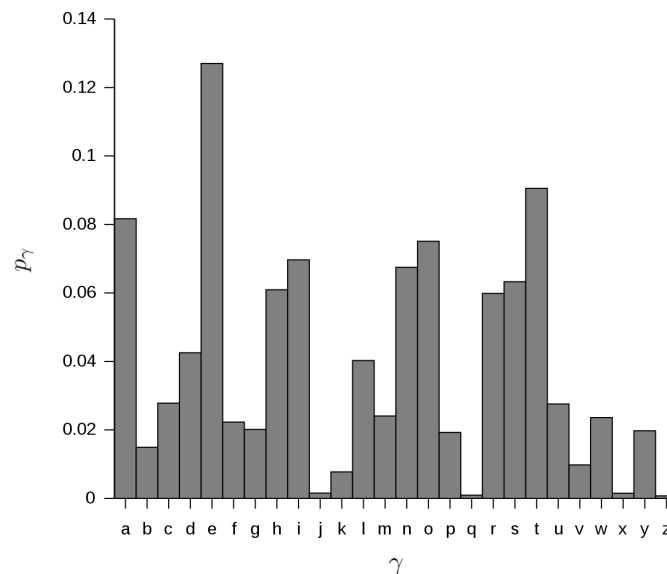
Tip Werk eerst een aantal voorbeelden uit, zodat je er zeker van bent dat je de theorie snapt.

Tip Gebruik eventueel de functie *strcspn* om newlines uit de ingelezen regels te verwijderen.

3 Deel 2: Aanval op het Vigenère cipher

3.1 Bepaling van de lengte van de geheime sleutel

De aanval van *Friedrich Kasiski* berust op frequentieanalyse op de letters uit het cryptogram. Zo komen de letter ‘e’, ‘t’ en ‘a’ vaak voor in Engelse tekst, terwijl de letters ‘j’, ‘q’, ‘x’ en ‘z’ bijna niet voorkomen (zie Figuur 2).



Figuur 2: De distributie van de letterfrequenties in de Engelse taal.

Deze *nonuniformiteit* kunnen we gebruiken om de lengte van de sleutel te bepalen. Laat p_γ de frequentie (‘probability’) van het karakter γ zijn in een gemiddelde Engelse tekst (bijv. op basis van Figuur 2 is p_a ongeveer 0.08). Merk op dat bij een sleutellengte van n elk n -de karakter in het cryptogram dezelfde verschuiving is ondergaan; ze zijn immers met hetzelfde sleutelkarakter versleuteld. Wanneer we de frequenties van de n -de karakters uit het cryptogram bepalen verwachten we dan de letterdistributie van een gemiddelde Engelse tekst (de p_γ ’s), maar dan verschoven (d.w.z. de staven in Figuur 2 zijn naar links of rechts verschoven). Wanneer we niet de n -de karakters (of een veelvoud hiervan) beschouwen, verwachten we dat deze frequenties daarentegen uniform verdeeld zijn.

Voorbeeld

Als de sleutellengte 7 is, betekent het dat de karakters op posities 0, 7, 14, ... in het cryptogram allemaal dezelfde verschuiving zijn ondergaan. Als we dus deze karakters samennemen en hun frequenties bepalen, verwachten we dezelfde letterfrequentie van een gemiddelde Engelse tekst, maar dan verschoven. We geven deze frequenties ook een naam: q_a is de frequentie van de letter a op posities 0, 7, 14, ... van het cryptogram.

Als we de sleutellengte verkeerd hebben gekocht, en we nemen nog steeds elk n -de karakter uit het cryptogram, dan zien we dat de frequenties meer zijn “uitgesmeerd”: de frequenties liggen dichter bij elkaar, en er zijn minder scherpe pieken en dalen in de distributie.

Op basis hiervan kunnen we een functie definiëren die meet hoe goed de letterfrequentie lijkt op wat we verwachten als we de sleutellengte goed hebben. Door alle letterfrequenties te kwadrateren krijgen we een soort “score”, die hoger ligt naarmate de frequenties ongelijkmatiger zijn verdeeld:

$$S_{\text{key}}(q) = q_a^2 + q_b^2 + \dots + q_z^2, \quad (3)$$

waar q_γ de frequentie is van het karakter γ in de reeks van n -de karakters uit het cryptogram. Let op: deze frequenties zijn *genormaliseerd*, de som van de frequenties is gelijk aan 1. Deze functie wordt gemaximaliseerd bij de juiste sleutellengte (of een veelvoud ervan).

Implementeer op basis hiervan de volgende functies:

- **sequence_distribution**

Implementeer de functie *sequence_distribution* die de letterfrequentiedistributie (d.w.z. de array $[q_a, q_b, \dots, q_z]$) maakt voor de reeks van n -de karakters uit het gegeven cryptogram:

```
void sequence_distribution(char ciphertext[], int n, double q[])
```

Het argument *ciphertext* bevat het cryptogram, n geeft de potentiële sleutellengte, en de resulterende distributie dient weg te worden geschreven in q .

- **key_score**

Implementeer de functie *key_score* die de key-score $S_{\text{key}}(q)$ berekent voor de gegeven letterfrequentiedistributie van een reeks van n -de karakters uit een cryptogram volgens Vergelijking (3):

```
double key_score(double q[])
```

Het argument q bevat de distributie $[q_a, q_b, \dots, q_z]$. Deze functie geeft de key-score $S_{\text{key}}(q)$ terug.

- **determine_key_length**

Implementeer de functie *determine_key_length* die de lengte van de geheime sleutel bepaalt aan de hand van het meegeleverde cryptogram:

```
int determine_key_length(char ciphertext[])
```

Het argument *ciphertext* bevat het cryptogram waarvoor de sleutellengte bepaald moet worden. Deze functie bepaalt de lengte van de geheime sleutel om het cryptogram te verkrijgen (of een veelvoud hiervan), op basis van de hierboven beschreven heuristiek.

3.2 Vaststelling van de geheime sleutel

Nadat we de lengte van de sleutel hebben bepaald, gaan we de individuele karakters van de sleutel achterhalen. Hierbij maken we opnieuw gebruik van frequentieanalyse op de letters uit het cryptogram.

We weten dat de reeks van n -de karakters, startende vanaf het i -de karakter, met hetzelfde sleutelkarakter versleuteld is. We noemen de reeks startende vanaf het i -de karakter (waar $0 \leq i < n$) de i -de *keystream*. Deze i -de keystream bestaat dus uit de karakters op posities i , $i + n$, $i + 2n$, et cetera. In de context van het eerder beschreven voorbeeld, waarbij $n = 7$, ziet dit er als volgt uit:

```

bericht      : onewhos havesal lthosea ndthose onlywho donotsh avethem selves
sleutel      : collatz collatz collatz collatz collatz collatz collatz collatz collatz
-----
cryptogram   : qbpqhhr jogpstk nhszszx presold qbwjwan fcyztlg cjpexhl uswgel

Keystream 0: qjnpqfcu
Keystream 1: bohrbcjs
Keystream 2: pgsewypw
Keystream 3: hpzsjzeg
Keystream 4: hssowthe
Keystream 5: htntlalxl
Keystream 6: rkzdnagl

```

Om het i -de karakter uit de sleutel te achterhalen, doen we frequentieanalyse op de karakters uit de i -de keystream. Hiertoe proberen we de keystream te ontsleutelen met ieder mogelijk sleutelkarakter (d.w.z. shift) om 26 verschillende ‘*candidate plaintext streams*’ te verkrijgen. We zijn op zoek naar de ‘candidate plaintext streams’ waarvan de distributie van de karakters zo goed mogelijk correspondeert met die van het Engels. Dit ‘*corresponderen*’ kunnen we wiskundig uitdrukken met behulp van de similarity-score:

$$S_{\text{sim}}(p, q, s) = p_a q_{\text{shift}_s(a)} + p_b q_{\text{shift}_s(b)} + \dots + p_z q_{\text{shift}_s(z)} \quad (4)$$

waar $\text{shift}_s(\gamma)$ gedefinieerd is als $\gamma + s \bmod 26$, p_γ de verwachte letterfrequentie is van het karakter γ volgens de Engelse letterdistributie, en q_γ de frequentie van karakter γ is in de i -de keystream. Merk op dat het verschuiven van de distributie met de functie $\text{shift}_s(\cdot)$ de transformatie is van de keystream distributie naar één van de 26 candidate plaintext stream distributies.

Het karakter die Vergelijking (4) maximaliseert is het meest waarschijnlijk als i -de sleutelkarakter. Wanneer je deze karakters achterhaalt voor elke mogelijke keystream (alle i) is de sleutel bepaald.

Implementeer op basis van bovenstaande theorie de volgende functies:

- **keystream_distribution**

Implementeer de functie *keystream_distribution* die de letterfrequenties bepaalt een keystream:

```
void keystream_distribution(char ciphertext[], int n, int i, double q[])
```

Het argument *ciphertext* bevat het cryptogram, n is de sleutellengte, i is de index van de keystream, en de resulterende distributie dient weg te worden geschreven in q .

- **similarity_score**

Implementeer de functie *similarity_score* die de score $S_{\text{sim}}(p, q)$ tussen twee letterfrequentiedistributies $[p_a, p_b, \dots, p_z]$ en $[q_{\text{shift}_s(a)}, q_{\text{shift}_s(b)}, \dots, q_{\text{shift}_s(z)}]$ berekent volgens Vergelijking (4):

```
double similarity_score(double p[], double q[], int s)
```

De argumenten p en q bevatten de te vergelijken distributies, s is een potentiële sleutelletter die bepaalt hoe de i -de keystream naar een candidate plaintext stream wordt omgevormd.

- **crack**

Implementeer de functie *crack* die de sleutel bepaalt aan de hand van het meegeleverde cryptogram:

```
void crack(char ciphertext[], int n, char key[])
```

Het argument *ciphertext* bevat het te kraken cryptogram, het argument n bevat de sleutellengte en de gevonden sleutel dient te worden weggeschreven in het argument *key*.

4 Voorbeelduitvoer

```
Welcome to Erenegiv, your Vigenere cipher cracking suite!
Please select your cracking option!
1 - Encrypt a message using the Vigenere Cipher.
2 - Decrypt a given ciphertext with a key.
3 - Crack a ciphertext with an unknown key.
4 - Read a .txt-file and perform the corresponding operation.
5 - Test the key score function.
6 - Exit the program.
$ Erenegiv:
> 1
Please enter the plaintext to be encrypted:
> One who shaves all those, and those only, who do not shave themselves.
Please enter the key to be used for encryption:
> collatz
CIPHERTEXT:
qbphhhrjogpstknhszszxpresoldqbwjwanfcyztlgcjpehkluswgel
$ Erenegiv:
> 2
Please enter the ciphertext to be decrypted:
> qbphhhrjogpstknhszszxpresoldqbwjwanfcyztlgcjpehkluswgel
Please enter the key to be used for decryption:
> collatz
PLAINTEXT:
onewhoshavesallthoseandthoseonlywhodonotshavethemselves
$ Erenegiv:
> 5
Please enter the ciphertext to calculate the key-scores for:
> qkvigjonxthnjkeucoilkjfziemjbbewdgjlbzrjidwfciojxbcfnbihpbkuumtryhuehelnw
kumuimumbbhdwftztjbfztioqmkitqmyiouxqhtbaqnupxhyjafelfbaqhpvxftabrffwnjwpux
ezbuxqmvzxcyobyelbvhrdfkmqnfdxhseiriwbtxixfavhccmwsrddtimjktbjiglywtbaqnnidu
mowluhtmbjctmbjbfzaulfwkjbfzxhyewkrfvmbvqfktndtirmbjkaynjamxuuackmujxsuvaxez
pckyaowkqhdmppubbdxdnmwhayejnjzpzjkuobncmvxxhjpabjcpvljbfzxicnxeoctvhtygggynf
igiqfzmxxyqzhfysbrezqwlynjwgymjtetygggux
KEY-SCORES:
Keylength 1: 0.043838
Keylength 2: 0.044939
Keylength 3: 0.048703
Keylength 4: 0.046168
Keylength 5: 0.077664
Keylength 6: 0.057143
Keylength 7: 0.058889
Keylength 8: 0.053044
Keylength 9: 0.054776
Keylength 10: 0.083900
Keylength 11: 0.063712
Keylength 12: 0.061224
Keylength 13: 0.068871
Keylength 14: 0.071111
Keylength 15: 0.119898
Keylength 16: 0.064472
Keylength 17: 0.072000
Keylength 18: 0.069444
Keylength 19: 0.074380
Keylength 20: 0.092971
```

```

$ Erenegiv:
> 4
Please enter the path to the file to be read.
> sample_input/crack3.txt
KEY:
qubitqubitqubit
PLAINTEXT:
aquantumparticleinasuperpositioncontrarytocommonbeliefisnotreallyintwoormore
statesatonce ratherasuperpositionmeansthatthereismorethanonepossibleoutcomeof
ameasurementforanobjectateverydayscalesdescribedbyclassicalphysics thatmakesn
osenseitiseitherhereorthereredorblueifwecantsaywhichitisthatsjustbecauseofou
rignorance wehaventlookedbutforquantumsuperpositionstheresimplyisnodefinitean
swerthepropertyofpositionisilldefined
$ Erenegiv:
> 4
Please enter the path to the file to be read.
> wrong.txt
The given file does not exist or is invalid.
$ Erenegiv:
> 2
Please enter the ciphertext to be decrypted:
> qbphhhr jogpstk nhszszx presold qbwjwan fcyztlg cjpehxl uswgel
Please enter the key to be used for decryption:
> !@jK(LD Ls&Ad*
The ciphertext and key may only contain lowercase characters.
> 6
See ya!

```

5 Inleveren

Lever **vigenere.c** in op Canvas en laat executables vooral achterwege.