Solutions to *Introduction to Modern Cryptography* (Third Edition) by Katz & Lindell

Edgard Lima <edgard.lima@gmail.com>
https://edgardlima.com

2025

Contents

Chapter 1

Introduction and Classical Cryptography

1.1 Decrypt the ciphertext provided at the end of the section on monoalphabetic substitution ciphers.

To decrypt it, let's play with C++ and create some classes and helper methods. The source code is available at KatzCryptoBookSolutions repository on Zer0Leak's GitHub

Step 01: Finding the word *the*

The word *the* is the most common word in english (see https://norvig.com/ngrams/). That's why we go for it.

Lauch the program with *auto step = kStep01_1;* This prints:

VFP	-	4
FPF	-	4
QVF	-	4

Table 1.1: Possible words with three letters in cipherText that appears at least 4 times

Notice that only VFP or QVF may be decrypted to *the*.

Lauch the program with *auto step = kStep01_2;* This executes:

```
        AlphabetTools::attackMonoAlphaCipher(_englishAlphabet
            , cipherText, knownDict, true);
```

And it will print a side-by-side comparison between cipher text and general english letter probabilities.

F (0.152)	<−>	e (0.127)
Q (0.107)	<−>	t (0.091)
W (0.086)	<−>	a (0.082)
G (0.078)	<−>	o (0.075)
L (0.070)	<−>	i (0.070)
O (0.066)	<−>	n (0.067)
V (0.061)	<−>	s (0.063)
H (0.057)	<−>	h (0.061)
B (0.049)	<−>	r (0.060)
P (0.041)	<−>	vd (0.043)
J (0.037)	<−>	l (0.040)
I (0.037)	<−>	u (0.028)
Z (0.029)	<−>	c (0.028)
R (0.029)	<−>	w (0.024)
M (0.016)	<−>	m (0.024)
E (0.016)	<−>	f (0.022)
Y (0.012)	<−>	y (0.020)
K (0.012)	<−>	g (0.020)
C (0.012)	<−>	p (0.019)
A (0.012)	<−>	b (0.015)
S (0.008)	<−>	v (0.010)
D (0.008)	<−>	k (0.008)
X (0.004)	<−>	x (0.002)
U (0.000)	<−>	j (0.002)
T (0.000)	<−>	z (0.001)
N (0.000)	<−>	q (0.001)

Table 1.2: Side-by-side cipheText and English letters probabilities

In additon, it prints the sum square disfference of probabilites of from VFP and QVF to *the*.

QVF	−>	0.000849
VFP	−>	0.016486

Table 1.3: Sum square distance probabilities mapping to 'the'

Visually looking the table of sorted probabilities, QVF seems to be better translated into *the*, in addition, the sum square distance above shows it is much better, *i.e.* the sum square distance is much smaller.

Lauch the program with *auto step = kStep01_3;* This will set the translation table and print the cipherText decryption:

Q	t
V	h
F	e

Table 1.4: Tranlation table known until this step.

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWEO
GWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFPFW
QVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILEOGQ
ILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX

____t_____h_____te____ee_t_e_e__________tt_______e_e_the_e____
____e_e___________e__e_t______t___e________e_e_____t_____he_e_
th_t_ee_t_the_t__e___e_e___eth______the___he_e__e__ththe_____t
___tet__thh__e_e___th_t___h__he_e___e_______t______t___e__
```

Step 02: Finding the word *that*.

The word *that* is one of the most common words in english ,and we will benefit from *t* and *h* we have already found.

Lauch the program with *auto step = kStep02_1;*.

For this we will replace all characters in cipher text that correspond to letters *t*, *h*, and *e*.

This will print all the words that with 4 letters that repeats at least 2 times.

WJhe	-	3
JheP	-	3
hePe	-	3
WthH	-	2
JGRM	-	2
hHtW	-	2
ePeW	-	2
WHGe	-	2
GRMt	-	2
Othe	-	2
eAeG	-	2
thHt	-	2
ePtO	-	2

Table 1.5: Words with four letters that repeats at least 2 times.

Notice that *thHt* is the only one that can match *that*. Then we assume that *H* tranlates into *a*.

Looking Table 1.2 show that *H* probability is 0.057 and *a* probability is 0.082, these are not to close, but we decided to take the risk.

One more mathematical approach could be to see check square distance of *QVFH* to *THEA* we just found, against a different choice in first step, leading to *VFP?* to *THEA* in this step. (? is the character we would find here if we toke a VFP in the first step).

Also, instead of print Table 1.2 in step *kStep02_1*, we could narrow down to print only words that matches *tha.t*, but we wanted to print the table to see if something interesting could grab our attention.

Lauch the program with *auto step = kStep02_2;* This will increment the translation table and print the cipherText decryption:

Q	t
V	h
F	e
H	a

Table 1.6: Tranlation table known until this step.

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWEO
GWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFPFW
QVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILEOGQ
```

```
ILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX

____t___a_h_____te__a_ee_t_e_e__________tt_______e_e_the_e____
____e_ea____a_____e__e_t______t___e________e_e_____t_____he_e_
that_ee_t_the_t__e___e_e___etha_a___the___he_e__ea_ththe_____t
__atet__thh__e_e___that___h__he_e_a_e___a___t____a_t___ea_
```

Now, what bring me attention was the sequence *e__ea_ththe*, actually *a_th* or *ea_th*, because there is no word with *thth*, let's find words that matches *^a.th$* or *.*ea.th$*. Also notice that in case of *.*ea.th$*, the two chars before *ea*, can't be any in *thea*.

index:	count	-	word
1332:	61059905	-	earth
4722:	14325239	-	wealth
13096:	3287162	-	auth
16550:	2207395	-	breadth
34718:	623206	-	anth
43011:	430700	-	unearth
47176:	367476	-	dearth
58228:	256042	-	acth
59047:	249807	-	ealth
65661:	208067	-	arth
84092:	134308	-	ayth
185308:	34516	-	xearth
193319:	32100	-	aith
219593:	26605	-	aeth
249908:	21239	-	alth

Table 1.7: Words matching regex *(^a.th$ |.*ea.th$)* with additional constraints

.

We could map *eaGth* into *earth*, *i.e.* map G with 0.078 into r with 0.060. And *earth* is the most common between them.

Or, we could map *LeaGth* into *wealth*, *i.e.* map G with 0.078 into l with 0.040, and map L with 0.070 into w with 0.024. *wealth* is still very common. But this leads to a bad probability match. One could try to map with it and see if translation shows something interesting.

Also, we could map *aGth* into *auth*, *i.e.* map *G* with 0.078 into *u* with 0.028. The word *auth* not that common but is related to the subject of the book. But the probabilites are not good either.

I will try mapping *eaGth* into *earth*. And see if something interesting follows.

Lauch the program with *auto step = kStep03_ 2;* This will increment the translation table and print the cipherText decryption:

Q	t
V	h
F	e
H	a
G	r

Table 1.8: Tranlation table known until this step.

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWEO
GWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFPFW
QVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILEOGQ
ILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX

_r__t__ra_h_____te__aree_tre_e__________tt_______e_erthe_e____
r___erea____a_____e__ert______t___e________e_e__r__t_____he_e_
that_ee_t_the_t__e__re_e__retha_a___ther__he_e__earththe____rt
__atetr_thh__e_er__that___h__he_e_are___a___tr___a_t__rea_
```

From now on I will follow similar approach, trying to find words, looking letter's probabilitis, checking if translation seems weird or good. Lauch the program with the next steps to see the output of each step.

If something letter really pulls our attention and we know for sure its translation, then fine, we add it to translation table, but if we are not sure, we try to match a letters that appears more and helps more on next step to find words or weird combinations.

The final result is:

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWEO
GWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFPFW
QVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILEOGQ
ILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX
```

```
cryptographicsystemsareextremelydifficulttobuildneverthelessfo
rsomereasonmanynonexpertsinsistondesigningnewencryptionschemes
thatseemtothemtobemoresecurethananyotherschemeonearththeunfort
unatetruthhoweveristhatsuchschemesareusuallytrivialtobreak
```

1.2 Provide a formal definition of the **Gen**, **Enc**, and **Dec** algorithms for the mono-alphabetic substitution cipher.

Equating the English alphabet with the set $\{0, \ldots, 25\}$ (so $a = 0$, $b = 1$, etc.), the message space $\mathcal{M}$ is then any finite sequence of integers from this set, *i.e.* message $m = m_1 \cdots m_\ell$ (where $m_i \in \{0, \ldots, 25\}$).

Let the encryptation key $K = (k_0, k_1, \ldots, k_{25})$ be an ordered sequence of 26 elements such that:

1. $k_i \in \{0, 1, \ldots, 25\}$ for all i.
2. $k_i \neq k_j$ for all $i \neq j$ (i.e., all elements are unique).

Gen is funciton that creates a key k, which is any permutation of $\{0, \ldots, 25\}$ with 26! possibilities.

Let **Enc** be a function that encrypts a message $\mathcal{M} = (m_1, m_2, \ldots, m_\ell)$ with a given key k. The encryption is defined as:

$$\mathbf{Enc}_k(m_1, m_2, \ldots, m_\ell) = (c_1, c_2, \ldots, c_\ell) \quad where \quad c_i = k(m_i) = k_{m_i}$$

Let **Dec** be a function that decrypts a chiphered message $\mathcal{C} = (c_1, c_2, \ldots, c_\ell)$ with a given key k.

Let g be a inverse of k function, i.e. $g(k(m_i)) = m_i \ and \ k(g(c_i)) = c_i \ \forall \ i$

I.e., given that j is the index in k where c_i is found, therefore $j \in \{0, \ldots, 25\}$, we have $g(c_i) = j$.

The decryptation is defined as:

$$\mathbf{Dec}_k(c_1, c_2, \ldots, c_\ell) = (m_1, m_2, \ldots, m_\ell) \quad where \quad m_i = g(c_i)$$

1.3 Provide a formal definition of the **Gen**, **Enc**, and **Dec** algorithms for the Vigenère cipher. (Note: there are several plausible choices for **Gen**; choose one.)

Equating the English alphabet with the set $A = \{0, \ldots, 25\}$ (so $a = 0$, $b = 1$, etc.), the message space $\mathcal{M}$ is then any finite sequence of integers from this set, *i.e.* message $m = m_1 \cdots m_\ell$ (where $m_i \in \{0, \ldots, 25\}$).

Let the encryptation key $k = (k_0, k_1, \ldots, k_n)$ be an ordered sequence of n finite natural integer elements such that $k_i \in A \ \forall \ i$ and n is a finite integer number. So, for each n there are 26^n different keys.

Gen is a function that generates a key k, which is an ordered selection with repetition (or a permutation with repetition) of elements from the set $\{0, \ldots, 25\}$ in n positions. And is defined as:

$$\mathbf{Gen}_n = (k_0, k_1, \ldots, k_{n-1}) \quad where \quad k_i \in \{0, \ldots, 25\}$$

Let **Enc** be a function that encrypts a message $\mathcal{M} = (m_0, m_1, \ldots, m_{\ell-1})$ with a given key k of length n. The encryption is defined as:

$$\mathbf{Enc}_k(m_0, m_1, \ldots, m_{\ell-1}) = (c_0, c_1, \ldots, c_{\ell-1}) \ where \ c_i = k(m,i), \ and \ k(m,i) \ is \ defined \ as:$$

$$\mathbf{h}(m,i) = c_i = [(m_i + k_j) \ mod \ 26] \quad where \ j \ = \ i \ mod \ n$$

Let **Dec** be a function that decrypts a ciphered message $\mathcal{C} = (c_0, c_1, \ldots, c_{\ell-1})$ with a given key k of length n. The decryptations is defined as:

$$\mathbf{Dec}_k(c_0, c_1, \ldots, c_{\ell-1}) = (m_0, m_1, \ldots, m_{\ell-1}) \ where \ m_i = g(c,i), \ and \ g(c,i) \ is \ defined \ as:$$

$$\mathbf{g}(c,i) = m_i = [(c_i - k_j) \ mod \ 26] \quad where \ j \ = \ i \ mod \ n$$

1.4 Say you are given a ciphertext that corresponds to English-language text that was encrypted using either the shift cipher or the Vigenère cipher with period greater than 1. How could you tell which was the case?.

Shift cipher is a special case of Vigenère where the key length is 1.

If the key length is 1, and q_i is the probability of occurrent of i-th letter in cipher text. This means that, if the key is j, $q_{(i+j) \ mod \ 26} \approx p_i$, where p_i is the probability of occurrent of i-th letter in english texts. I.e. probalities are close to the same, with shifted j index in q and p. Therefore, it is ShiftCipher, check the following:

$$\sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} p_i^2 \approx 0.065$$

If you want to double try to find key length using *index of coincidence method* as described in text book and implemented at:

vigenereattack.cpp: _findKeyLength on **chap_01** solutions at Zer0Leak's solutions of KatzCryptoBookSolutions on GitHub.

1.5 Implement the attacks described in this chapter for the shift cipher and the Vigenère cipher.

See **chap_01** solutions at Zer0Leak's solutions of KatzCryptoBookSolutions on GitHub.

1.6 The shift and Vigenère ciphers can also be defined on the 256-character alphabet consisting of all possible bytes (8-bit strings), and using XOR instead of modular addition.

(a) Provide a formal definition of both schemes in this case.

Let's provide the formal defintion of Vigenère cipher. The formal definition of Shift chipher is a special case of it where the encryptation key is $K = (k_0)$ i.e. the encryptation key has length 1.

Let the *Sign* alphabet to be the set $\{0x00, \ldots, 0xFF\}$ (where each symbol is an unsigned 8-bits integer), the message space $\mathcal{M}$ is then any finite sequence from this set, *i.e.* message $m = m_1 \cdots m_\ell$ (where $m_i \in Sign$).

Let the encryptation key $K = (k_0, k_1, \ldots, k_w)$ be an finite ordered sequence of w elements such that $k_i \in Sign$ for all i.

Gen is funciton that creates a key k with a finite length w, which is any permutation of $\{0x00, \ldots, 0xFF\}$ with 255^w possibilities.

Let **Enc** be a function that encrypts a message $\mathcal{M} = (m_1, m_2, \ldots, m_\ell)$ with a given key k. The encryption is defined as:

$$\mathbf{Enc}_k(m_1, m_2, \ldots, m_\ell) = (c_1, c_2, \ldots, c_\ell) \quad where \quad c_i = m_i \; XOR \; k_{(i \; mod \; w)}$$

Let **Dec** be a function that decrypts a chiphered message $\mathcal{C} = (c_1, c_2, \ldots, c_\ell)$ with a given key k. The decryptation is defined as:

$$\mathbf{Dec}_k(c_1, c_2, \ldots, c_\ell) = (m_1, m_2, \ldots, m_\ell) \quad where \quad m_i = c_i \; XOR \; k_{(i \; mod \; w)}$$

(b) Discuss how the attacks we have shown in this chapter can be modified to break these schemes.

The attacks discussed in this chapter, and implemented in exercise **1.5** can be modified as follow:

The **Gen** function now uses alphabet $\{0x00, \dots, 0xFF\}$ instead of $\{0, \dots, 25\}$.

The **Enc** and **Dec** functions now, for each symbol uses $f_i = s_i\ XOR\ k_{(i\ mod\ w)}$ instead of $f_i = (s_i\ \pm\ k_{(i\ mod\ w)})\ mod\ 26$

1.7 The index of coincidence method relies on a known value for the sum of the squares of plaintext-letter frequencies (cf. Equation (1.1)). Why would it not work using the $\sum_i p_i$ itself?

First of all, $\sum_i p_i$ is always 1 (*i.e.* 100%). So, it is useless.

But also, using square sum, uneven distribution will lead to higher values of sumation, while uniform will result in smaller values. So, using squared sum will help in find the statistic distribution that is closer to english distribuition.

1.8 Show that the shift, substitution, and Vigenère ciphers are all trivial to break using a chosen-plaintext attack. How much chosen plaintext is needed to recover the key for each of the ciphers?

In **chosen-plaintext attack** there is a cipherd text of your knowledge you want to discover its plain-text pair. You don't know the key though. But you can choose any plain-text of your choice to cipher using the same unknown key. Thus you can have any pair of text/cipher text of you choice generated by the same key that was used to cipher the text under attack.

For the Shift cipher, just use the plain text "a" wich is integer 0, the key is $k = c_0$. Then use the just discovered key to break the cipher text under attack.

For the Vigenère cipher, just use the plain text "aaa..." wich are integers 000..., for each symbol i, the key symbol at index i is $k_i = c_i$. The length of the plain-text you need it at most the lenght of the cipher text you want to break. Then, from k, if it has a repeating sequence, extract a simpler key from it. Then use the just discovered key to break the cipher text under attack.

1.9 Assume an attacker knows that a user's password is either **bcda** or

bedg. Say the user encrypts his password using the shift cipher, and the attacker sees the resulting ciphertext. Show how the attacker can determine the user's password, or explain why this is not possible.

It is trivial to discover.

bcda is 1230 and bedg is 1436. So, using Shift ciphder will just shift these by k mod 26. But the difference between them must remain the same.

Consider the cipher text to be $c_0c_1c_2c_3$.

If key is **bcda**:

$(c_1 - c_0) \bmod 26 = 1$
$(c_2 - c_0) \bmod 26 = 2$
$(c_3 - c_0) \bmod 26 = 25$

If key is **bedg**:

$(c_1 - c_0) \bmod 26 = 3$
$(c_2 - c_0) \bmod 26 = 2$
$(c_3 - c_0) \bmod 26 = 5$

It is still possible to break by comparing c_1 and c_0, or c_3 and c_0. You can compare indexes if their diffence is not the same, *i.e.* comparing c_2 and c_0 in this case is useless

1.10 Repeat the previous exercise for the Vigenère cipher using period 2, using period 3, and using period 4.

(a) Period 2

If key is **bcda**:

$(c_2 - c_0) \bmod 26 = 2$
$(c_3 - c_1) \bmod 26 = 24$

If key is **bedg**:

$(c_2 - c_0) \bmod 26 = 2$
$(c_3 - c_1) \bmod 26 = 2$

So, it is still possible to break by comparing c_3 and c_1

(a) Period 3

If key is **bcda**:

$(c_3 - c_0) \bmod 26 = 25$

If key is **bedg**:

$(c_3 - c_0) \bmod 26 = 5$

So, it is still possible to break by comparing c_3 and c_0
(a) Period 4

It is impossible to discover.

1.11 The attack on the Vigenère cipher has two steps: (a) find the key length by identifying with $S_\tau \approx 0.065$ (cf. Equation (1.3)) and (b) for each character of the key, find j maximizing I_j (cf. Equation (1.2)), using $\{p_i\}$ corresponding to English text. What happens in each case if the underlying plaintext is in a language other than English?

If the language has a different sum of square of probababilites, and we don't know that, we will notice that for multiple τ the value will be closely the same. And then we can figure out τ and the language's sum of square of probababilites. Considering it is a latin alphabet of 26 letter, from 'a' to 'z', for wrong values of τ, the sum will tend towards 0.038, i.e. $\sum_{i=0}^{25} \left(\frac{1}{26}\right)^2$

For the second part, equation 1.2, it will result in wrong decryption producing garbage.

Chapter 2

Perfectly Secret Encryption

TODO