«به نام خدا»

تکلیف اول - مرضیه علیدادی – 9631983

1.

- **1-7)**

1. multiplication table for Z4:

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

2. multiplication table for Z5:

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

addition tables for Z5:

×	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

3. multiplication table for Z6:

×	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

addition tables for Z6:

×	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

4. Elements without a multiplicative inverse in Z4: 0, 2

Elements without a multiplicative inverse in Z6: 0, 2, 3, 4

5 is a prime number, so a multiplicative inverse for all nonzero elements of Z5 exists. And all nonzero elements smaller than 5 are relatively prime to 5.

- **1-8)**

Because 5 is relatively prime to 11, 12 and 13; there is a multiplicative inverse of it in each Z11, Z12 and Z13.

- trial-and-error search in Z11:
  - 5 . 0 = 0
  - 5 . 1 = 5
  - 5 . 2 = 10
  - 5 . 3 = 15 mod 11 = 4
  - 5 . 4 = 20 mod 11 = 9
  - 5 . 5 = 25 mod 11 = 3
  - 5 . 6 = 30 mod 11 = 8
  - 5 . 7 = 35 mod 11 = 2
  - 5 . 8 = 40 mod 11 = 7
  - 5 . 9 = 45 mod 11 = 1
  - 5 . 10 = 50 mod 11 = 6

So, the multiplicative inverse of 5 in Z11 is 9.

- trial-and-error search in Z12:
  - $5 . 0 = 0$
  - $5 . 1 = 5$
  - $5 . 2 = 10$
  - $5 . 3 = 15 \bmod 12 = 3$
  - $5 . 4 = 20 \bmod 12 = 8$
  - $5 . 5 = 25 \bmod 12 = 1$
  - $5 . 6 = 30 \bmod 12 = 6$
  - $5 . 7 = 35 \bmod 12 = 11$
  - $5 . 8 = 40 \bmod 12 = 4$
  - $5 . 9 = 45 \bmod 12 = 9$
  - $5 . 10 = 50 \bmod 12 = 2$
  - $5 . 11 = 55 \bmod 12 = 7$

  So, the multiplicative inverse of 5 in Z12 is 5.

- trial-and-error search in Z13:
  - $5 . 0 = 0$
  - $5 . 1 = 5$
  - $5 . 2 = 10$
  - $5 . 3 = 15 \bmod 13 = 2$
  - $5 . 4 = 20 \bmod 13 = 7$
  - $5 . 5 = 25 \bmod 13 = 12$
  - $5 . 6 = 30 \bmod 13 = 4$
  - $5 . 7 = 35 \bmod 13 = 9$
  - $5 . 8 = 40 \bmod 13 = 1$
  - $5 . 9 = 45 \bmod 13 = 6$
  - $5 . 10 = 50 \bmod 13 = 11$
  - $5 . 11 = 55 \bmod 13 = 3$
  - $5 . 12 = 60 \bmod 13 = 8$

  So, the multiplicative inverse of 5 in Z13 is 8.

- **1-9)**
  - $3^2 \bmod 13 = 9 \rightarrow$ **x = 9**
  - $7^2 \bmod 13 = 49 \bmod 13 = 10 \rightarrow$ **x = 10**
  - $3^{10} \bmod 13 = 3^4 . 3^3 . 3^3 \bmod 13 = 81 . 27 . 27$ mode $13 = 3 . 1 . 1 \bmod 13 = 3$
    $\rightarrow$ **x = 3**
  - $7^{100} \bmod 13 = (7^2)^{50} \bmod 13 = 49^{50} \bmod 13 = 10^{50} \bmod 13 = (-3)^{50} \bmod 13$
    $= (3^{10})^5 \bmod 13 = 3^5 \bmod 13 = 3^4 . 3 \bmod 13 = 81 . 3 \bmod 13$
    $= 3 . 3 \bmod 13 = 9 \rightarrow$ **x = 9**
  - $11 \bmod 13 = 11 + 13m = 7^x \rightarrow 7^x \bmod 13$
    trial-and-error: $x = 5 \rightarrow 7^5 = 7^2 . 7^2 . 7 \bmod 13$
    $= 10 . 10 . 7 \bmod 13$
    $= 70 . 10 \bmod 13$
    $= 50 \bmod 13 = 11$
    $(x = 12 \rightarrow 7^{12} = (7^3)^4 \bmod 13 = 5^4 \bmod 13 = 1)$
    $\rightarrow$ **x = 5 + 12k**

- **1-10)**
  - $m = 4 \rightarrow n = 1, 3 \rightarrow \varphi(4) = 4 . (1 - \frac{1}{2}) = 2$
  - $m = 5 \rightarrow n = 1, 2, 3, 4 \rightarrow \varphi(5) = 5 . (1 - \frac{1}{5}) = 4$
  - $m = 9 \rightarrow n = 1, 2, 4, 5, 7, 8 \rightarrow \varphi(9) = 9 . (1 - \frac{1}{3}) = 6$
  - $m = 26 \rightarrow n = 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25$
    $\rightarrow \varphi(26) = 26 . (1 - \frac{1}{2}) . (1 - \frac{1}{13}) = 12$

- **1-13)** In affine cipher: $y = a x + b$, And here we have (x_1 , y_1) and (x_2 , y_2)

$$\text{So: } \begin{cases} y_1 = a x_1 + b \\ y_2 = a x_2 + b \end{cases} \longrightarrow \begin{cases} a = (x_1 - x_2)^{-1} (y_1 - y_2) \bmod m \\ b = y_1 - a x_1 \bmod m \end{cases}$$

[note that: $(x_1 - x_2)^{-1}$ must exist mod m → i.e. $\gcd((x_1 - x_2) , m) = 1$]

.2

(2-1

- LFSR هایی که چندجمله ای فیدبک آن ها از نوع Irreducible polynomial است، یعنی نمی‌توان آن ها را به چندجمله ای با درجه کوچک تر تجزیه کرد؛ دوره تناوب خروجی شان دقیقا برابر exponent شان می شود ($P = N$). اگر tap آخر وجود داشته باشد ، تضمین می کند که خروجی حتماً پریودیک است، و چندجمله ای تبدیل به primitive می شود؛ اگر وجود نداشته باشد، نهایتاً پریودیک است.
- LFSR هایی که چندجمله ای فیدبک آن ها از نوع primitive polynomial است، خروجی شان دارای حداکثر پریود ممکن است. یعنی اگر طول LFSR را با m نمایش دهیم، پریود برابر است با:

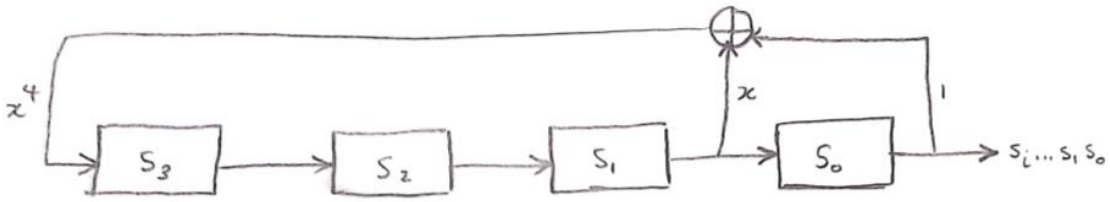
$P = 2^m - 1$

این ها حالت خاصی از Irreducible polynomial ها هستند. و دارای ویژگی های آن ها نیز هستند.
همچنین، خروجی ای که تولید می کنند، دارای خواص آماری خوبی است و دارای توزیع یکنواخت است. در این ها tap آخر باید حتما وجود داشته باشد. و خروجی حتما پریودیک است. ورودی را باید از حالت غیر صفر شروع کنیم.
چون tap آخر وجود دارد، اگر از حالت ورودی غیر صفر شروع کنیم، هیچ گاه به صفر نمی‌رسیم.

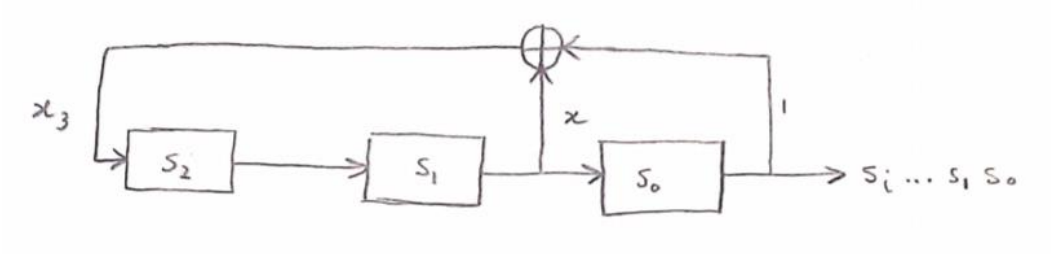
- برای LFSR هایی که چندجمله ای فیدبک آن ها از نوع Reducible polynomial است، یعنی می‌توان آن ها را به چندجمله ای با درجه کوچک تر تجزیه کرد؛ دیگر نمی توان گفت که خروجی شان دقیقا برابر exponent شان می شود. بلکه فقط می توان گفت که: $p \mid N \ : \ (N = p . q)$

(2-2

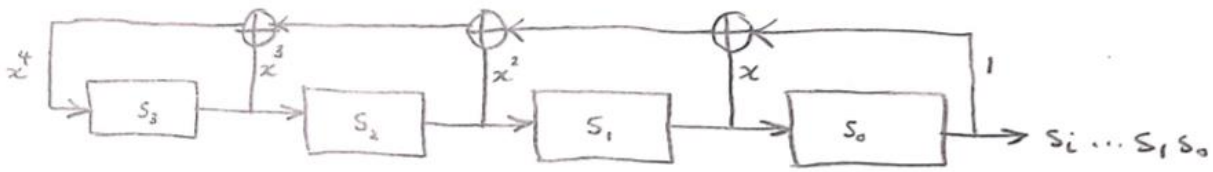
- $x^4 + x + 1$:



- $x^3 + x + 1$:

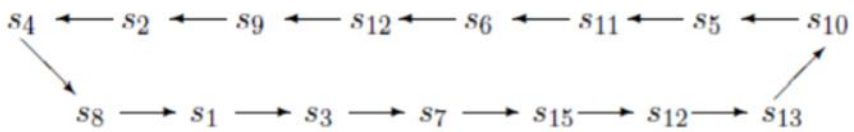


- $x^4 + x^3 + x^2 + x + 1$:

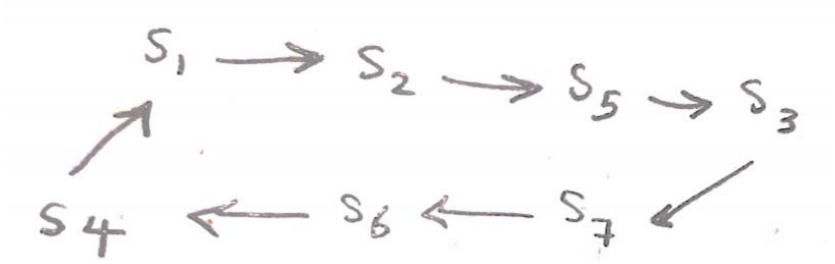


(2-3

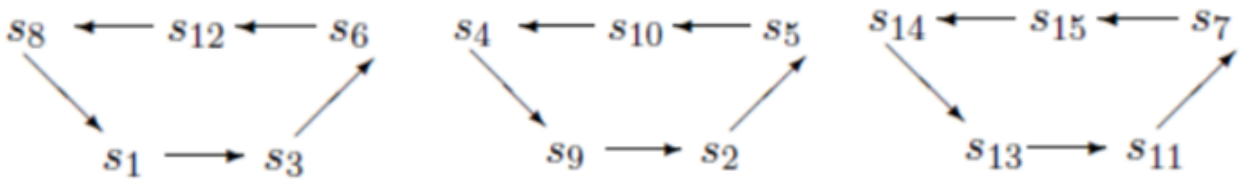
- چندجمله ای اول Irreducible و primitive است؛ زیرا اولاً به چندجمله ای های کوچک تر قابل تجزیه نیست، و دوماً دارای یک لوپ با طول حداکثر است. یعنی پریود آن به این شکل است:
  $P = 2^L - 1 = 2^4 - 1 = 15$



- چندجمله ای دوم Irreducible و primitive است؛ زیرا اولاً به چندجمله ای های کوچک تر قابل تجزیه نیست، و دوماً دارای یک لوپ با طول حداکثر است. یعنی پریود آن به این شکل است:
  $P = 2^L - 1 = 2^3 - 1 = 7$



- چندجمله ای سوم Irreducible است؛ زیرا به چندجمله ای های کوچک تر قابل تجزیه نیست، و دارای چند لوپ است:



(2-4

- چند جمله ای اول: $2^L - 1 = 2^4 - 1 = \mathbf{15}$
- چند جمله ای دوم: $2^L - 1 = 2^3 - 1 = \mathbf{7}$
- چند جمله ای دوم: $N = P = \mathbf{5}$

.3

(a Attacker به **512** جفت بیت متوالی x_i , y_i مربوط به plainText نیاز دارد، تا حمله ی موفقیت آمیزی داشته باشد.

(b

.1 Attacker اول باید آن 512 جفت بیتی که در قسمت قبل گفته شد، را تهیه کند.

.2 سپس باید $s_i = x_i + y_i \bmod 2$; $i = 0, 1, ..., 2m-1$ را محاسبه کند.

.3 Attacker باید به منظور به دست آوردن چندجمله ای فیدبک، 256 عبارت خطی با استفاده از رابطه ی بین بیت های آن و key stream خروجی آن که با رابطه ی زیر محاسبه می شود، تولید کند:

$$s_{i+m} \equiv \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \bmod 2 \quad ; \quad s_i , p_j \in \{0,1\} ; i = 0, 1, 2, ..., 255 \quad ; \quad m = 256$$

.4 پس از تولید این عبارت خطی، آن را حل می کنیم. و آن 256 ضریب فیدبک را فاش می کنیم.

(c کلید این سیستم با 256 ضریب فیدبک نشان داده می شود.
با توجه به اینکه محتوای اولیه ی این LFSR به صورت یک طرفه به بیرون آن شیفت داده می شود و با 256 بیت اول plaintext، XOR می شوند؛ به آسانی قابل محاسبه هستند.

---

4. به توجه به اینکه می دانیم سه حرف اول، چه حروفی هستند، بررسی می کنیم:

W: $22 = 10110_2$ → J: $9 = 01001_2$
P: $15 = 01111_2$ → 5: $31 = 11111_2$
I: $8 = 01000_2$ → A: $0 = 00000_2$

با توجه به اینکه داریم:

$\text{plaintext} \oplus \text{key} = \text{ciphertext} \rightarrow \text{plaintext} \oplus \text{ciphertext} = \text{key}$

با استفاده XOR برای این 3 جفت کاراکتر اصلی و کاراکتر رمز شده ی متناظر، کلید را پیدا می کنیم:

```
10110 01111 01000    : plaintext
01001 11111 00000    : ciphertext
-----------------
11111 10000 01000    : keystream
```

.1

```
def bit_encoding_map():
    return zip(string.ascii_lowercase + string.digits, range(0, 32))

def bitencode(text):
    def encode_char(c):
        map = dict(bit_encoding_map())
        return bin(map[c.lower()]).lstrip("-0b").zfill(5)
    return "".join([encode_char(c) for c in text])
def bitdecode(text):
    def decode_block(bits):
        map = dict([(b, a) for a, b in bit_encoding_map()])
        return map[int(bits, 2)].upper()
    return "".join([decode_block(bits) for bits in re.findall(".{1,5}", text)])

def lfsr(init_vector):
    registers = list(bin(init_vector).lstrip("-0b").zfill(6)[-6:])
    while True:
        registers.insert(0, "1" if registers[-1] != registers[-2] else "0")
        yield registers.pop()
def xor_bitstream(a, b):
    return "".join(["1" if a != b else "0" for a, b in zip(a, b)])
```

```
def apply_keystream(lfsr_init_vector, text):
    return bitdecode(xor_bitstream(lfsr(lfsr_init_vector), bitencode(text)))

def print_ciphertext_and_chosen_plaintext(ciphertext, chosen_plaintext):
    print "Ciphertext:"
    print "-------\n"
    print ciphertext
    print "-------\n"
    print "Chosen Plaintext:"
    print "-------\n"
    print chosen_plaintext

def print_keystream(keystream):
    print "-------\n"
    print "Revealed keystream:"
    print "-------\n"
    print keystream
def print_decrypted_message(plaintext):
    print "-------\n"
    print "Revealed plaintext:"
    print "-------\n"
    print plaintext, "\n"

if __name__ == "__main__":
    ciphertext = "j5a0edj2b"
    chosen_plaintext = "WPI"
    lfsr_init_vector = 63
    keystream = xor_bitstream(bitencode(ciphertext), bitencode(chosen_plaintext))
    plaintext = apply_keystream(lfsr_init_vector, ciphertext)
    print_ciphertext_and_chosen_plaintext(ciphertext, chosen_plaintext)
```

```
print_keystream(keystream)
print_decrypted_message(plaintext)
```

---

2. با توجه به اینکه طول LFSR برابر 6 است، 6 بیت ابتدایی کلید، همان initialization vector است:
111111

---

3. با توجه به این عبارات:

$$S_5p_5 + s_4p_4 + s_3p_3 + s_2p_2 + s_1p_1 + s_0p_0 = s_6$$
$$S_6p_5 + s_5p_4 + s_4p_3 + s_3p_2 + s_2p_1 + s_1p_0 = s_7$$
$$S_7p_5 + s_6p_4 + s_5p_3 + s_4p_2 + s_3p_1 + s_2p_0 = s_8$$
$$S_8p_5 + s_7p_4 + s_6p_3 + s_5p_2 + s_4p_1 + s_3p_0 = s_9$$
$$S_9p_5 + s_8p_4 + s_7p_3 + s_6p_2 + s_5p_1 + s_4p_0 = s_{10}$$
$$S_{10}p_5 + s_9p_4 + s_8p_3 + s_7p_2 + s_6p_1 + s_5p_0 = s_{11}$$

، ضرایب به این ترتیب محاسبه می شوند:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{matrix} = p_0 \\ = p_1 \\ = p_2 \\ = p_3 \\ = p_4 \\ = p_5 \end{matrix}$$

---

4. **‘wombat’** یک حیوان است که در چند ایالت **استرالیا** زندگی می کند.

---

5. Known-plaintext attack

---

5. با توجه به اینکه یک plaintext و ciphertext معادل آن را داریم، می توانیم keystream نظیر را محاسبه کنیم. به این منظور، باید اعداد اسکی نظیر حروف شرکت کننده در plaintext را به جای آن ها قرار دهیم. طبق رابطه‌ی $\text{ciphertext} = \text{plaintext} \oplus \text{key}$ که بر الگوریتم حاکم است، و طبق خواص XOR داریم:

$$\text{plaintext} \oplus \text{ciphertext} = \text{key}$$

Plaintext	B	A	R	A	C	K	O	B	A	M	A
Plaintext ASCII	01000010	01000001	01010010	01000001	01000011	01001011	01001111	01000010	01000001	01001101	01000001
Ciphertext	01000011	00011011	00010010	00110000	11111000	10100111	10001110	11101001	00010100	00011101	01100100
Keystream + Nonce	00000001	01011010	01000000	01110001	10111011	11101100	11000001	10101011	01010101	01010000	00100101

با توجه به اینکه مقدار nonce ها را می دانیم، کلید استفاده شده در عبارت دوم را به دست می آورم:

Keystream + 1	00000001	01011010	01000000	01110001	10111011	11101100	11000001	10101011	01010101	01010000	00100101
Keystream	00000000	01011001	00111111	01110000	10111010	11101011	11000000	10101010	01010100	01001111	00100100
Keystream + 2	00000010	01011011	01000001	01110010	10111100	11101101	11000010	10101100	01010110	01010001	00100110

برای رمزنگاری متن دوم، دوباره از رابطه ی زیر استفاده می کنم:

$$\text{plaintext} \oplus \text{key} = \text{ciphertext} \quad \rightarrow \quad \text{ciphertext} \oplus \text{key} = \text{plaintext}$$

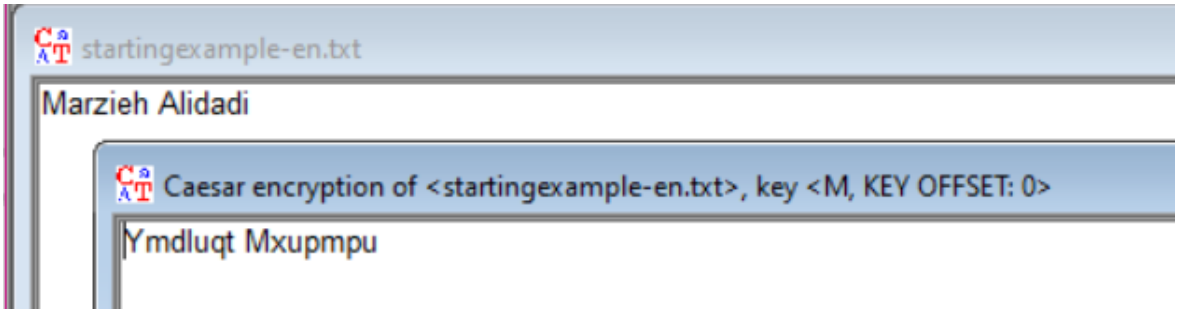
Ciphertext	01000110	00010100	00001111	00110011	11110000	10101001	10010110	11111110	00000011	00011100	01110110
Keystream + 2	00000010	01011011	01000001	01110010	10111100	11101101	11000010	10101100	01010110	01010001	00100110
Plaintext ASCII	01000100	01001111	01001110	01000001	01001100	01000100	01010100	01010010	01010101	01001101	01010000
Plaintext	D	O	N	A	L	D	T	R	U	M	P

حروف نظیر این اعداد اسکی به دست آمده، همان Plaintext مد نظر است و بدین شکل است:

DONALDTRUMP

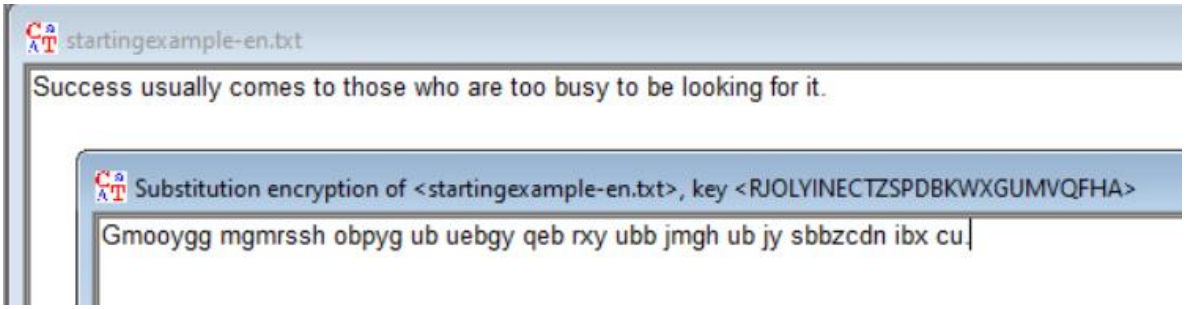
6.

1. The encrypted text: **Ymdluqt Mxupmpu** → the alphabet shifted by **12** letters.



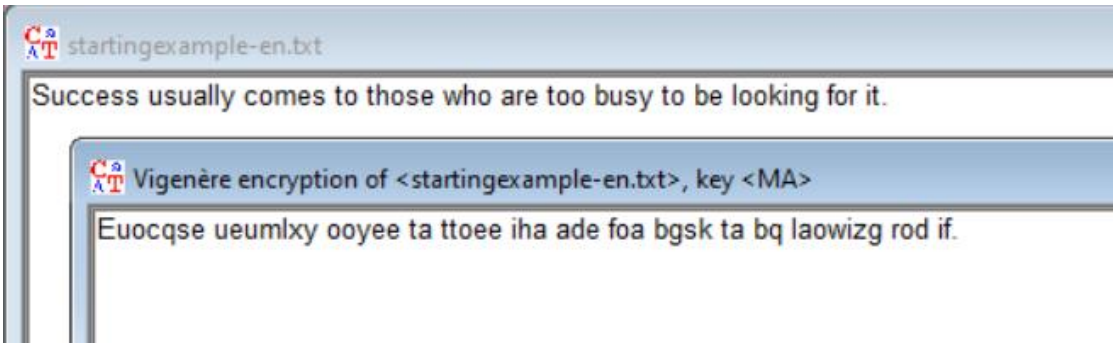
---

2. Offset = 9631983 % 26 = 23 → The encrypted text:
Gmooygg mgmrssh obpyg ub uebgy qeb rxy ubb jmgh ub jy sbbzcdn ibx cu.

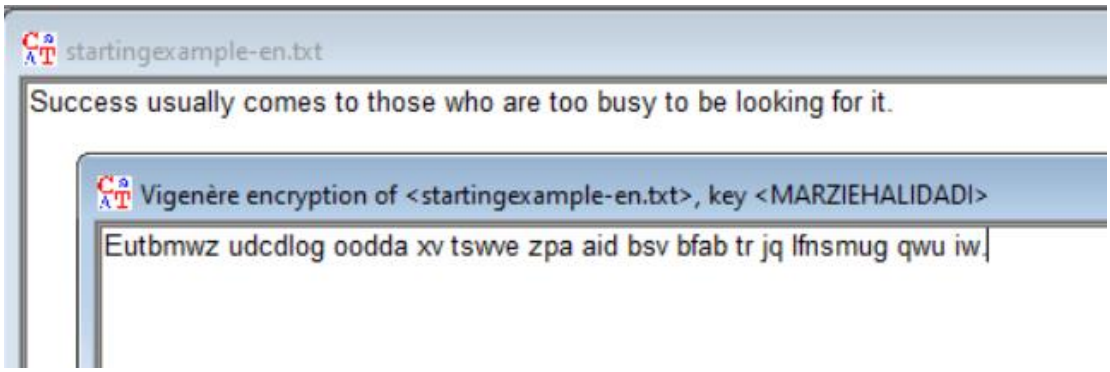


3.

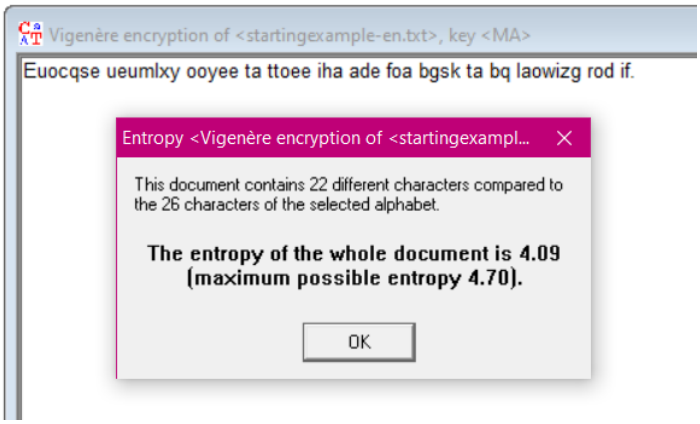
a) The encrypted text:
Euocqse ueumlxy ooyee ta ttoee iha ade foa bgsk ta bq laowizg rod if.



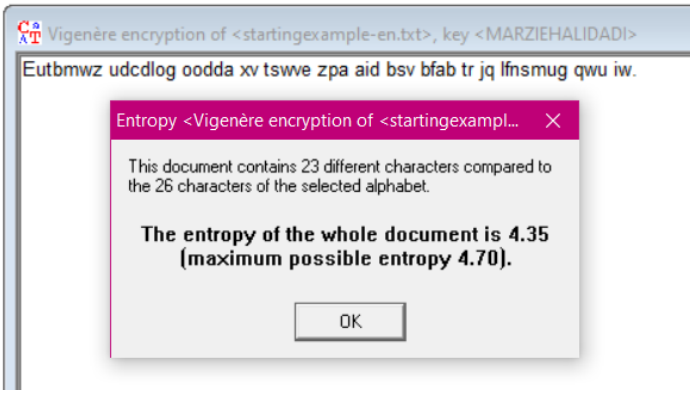
b) The encrypted text:
Eutbmwz udcdlog oodda xv tswve zpa aid bsv bfab tr jq lfnsmug qwu iw.



c) First part with key = MA

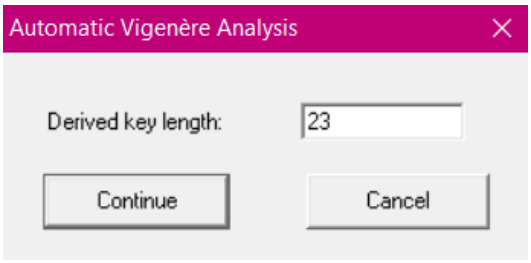


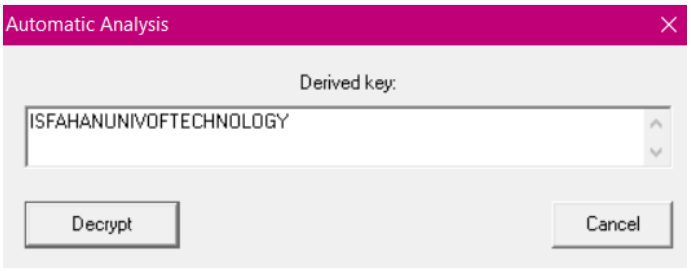
Second part with key = MARZIEHALIDADI

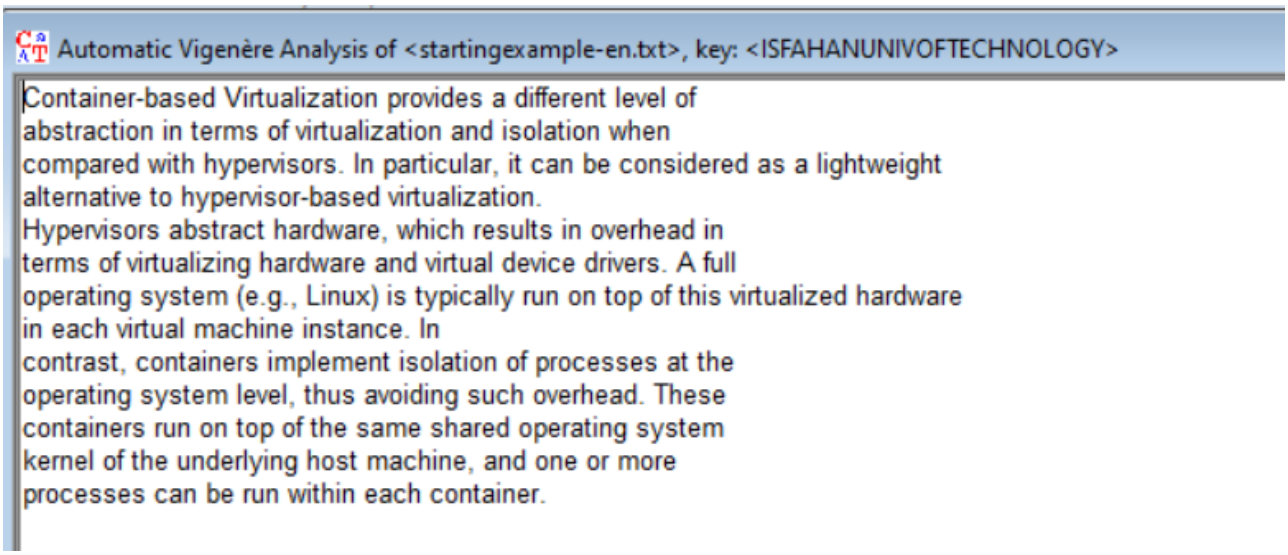


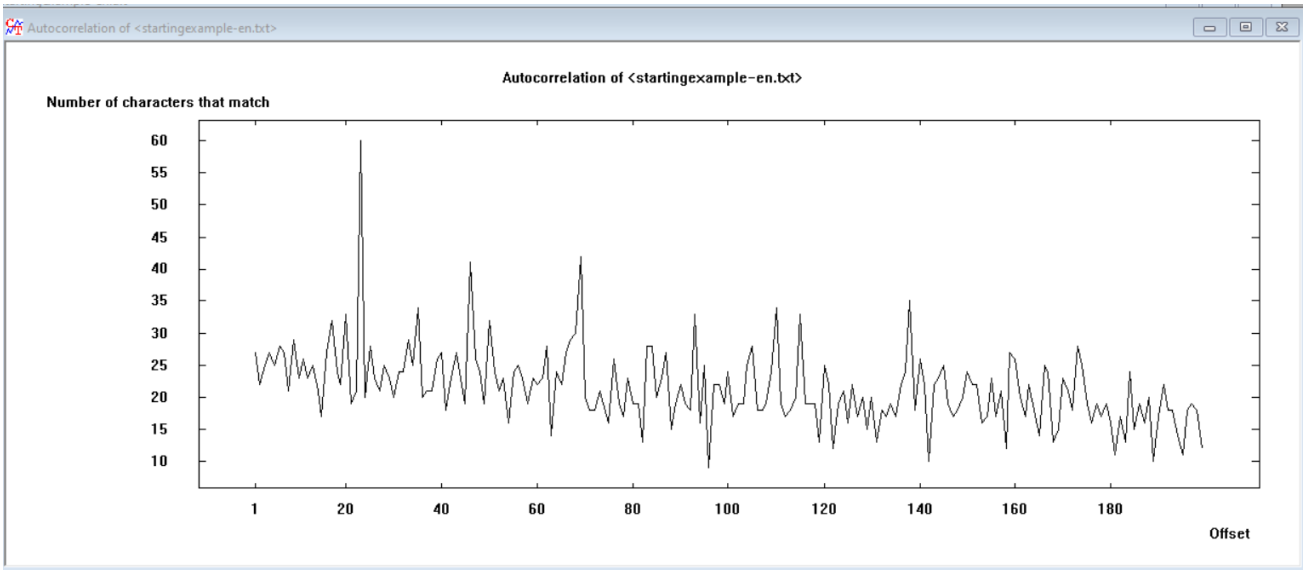
Entropy refers to the randomness collected by a system for use in algorithms that require random data. A lack of good entropy can leave a cryptosystem vulnerable and unable to encrypt data securely.
So, when the entropy of a cryptosystem is better, it is more secure.
In one method, the longer key results a more secure encryption. As we see, when I used "MA" as key, the entropy is less than when I used "MARZIEHALIDADI" as the key of encryption. So it's necessary to use a bigger key.

4.







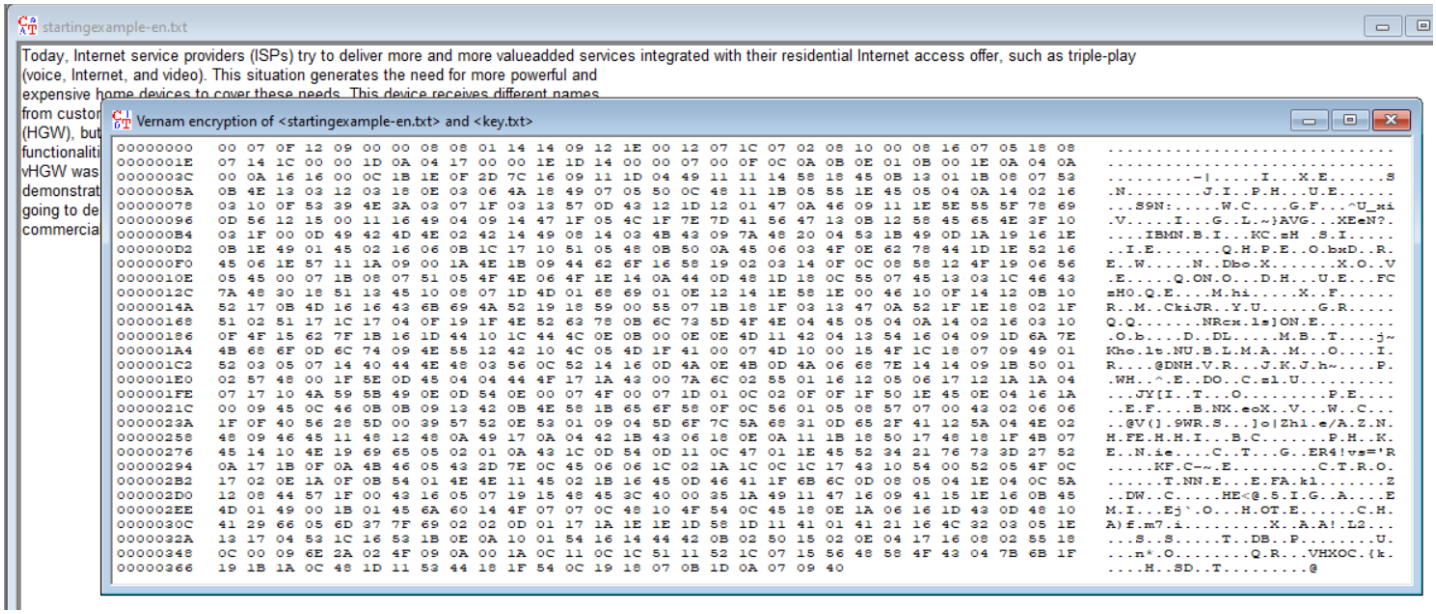


- Diagram is showing the **frequency** graph.
- **Plaintext:** Container-based Virtualization provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. In particular, it can be considered as a lightweight alternative to hypervisor-based virtualization.
  Hypervisors abstract hardware, which results in overhead in

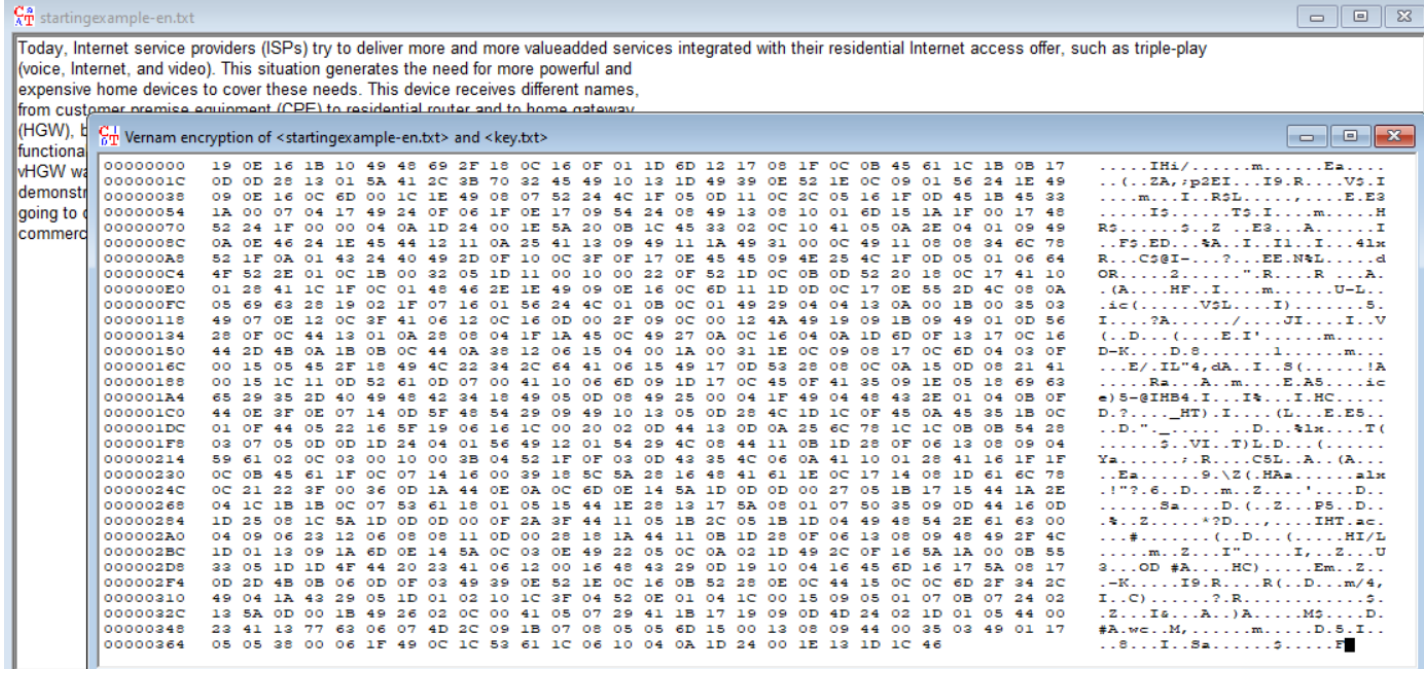
terms of virtualizing hardware and virtual device drivers. A full operating system (e.g., Linux) is typically run on top of this virtualized hardware in each virtual machine instance. In contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine, and one or more processes can be run within each container.

5.

a.

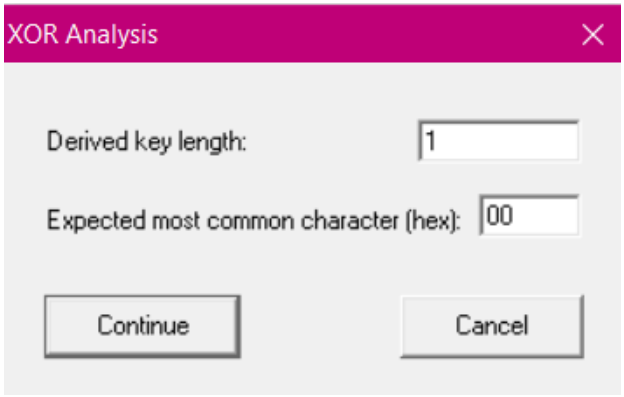


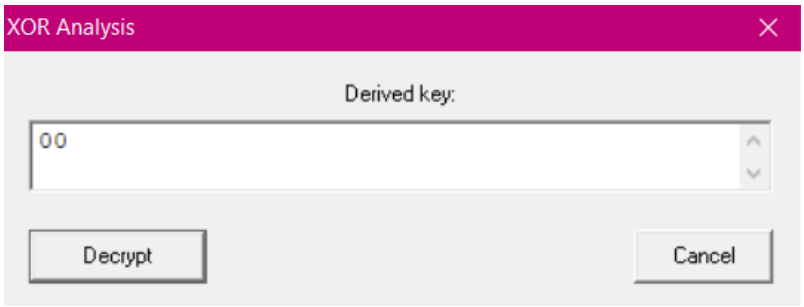
b. Key: **Marzieh Alidadi**

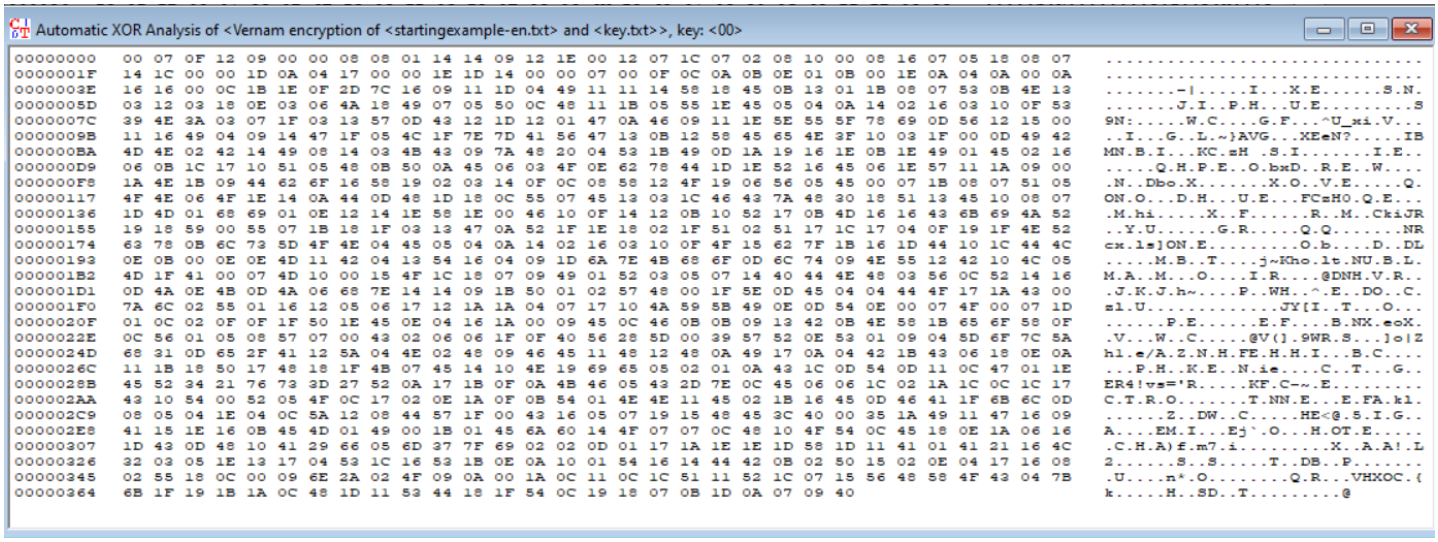


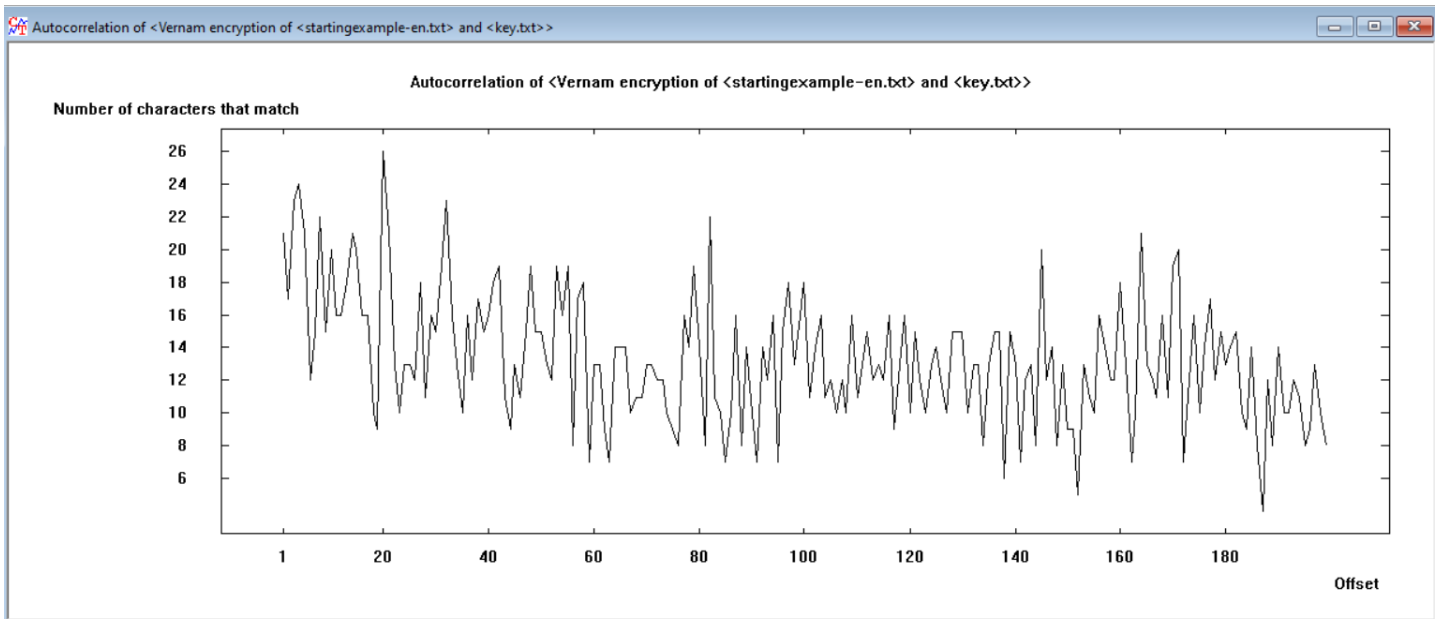
c.

- **Part a (the long key):**

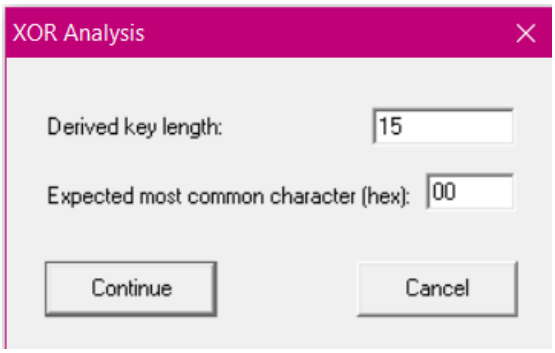


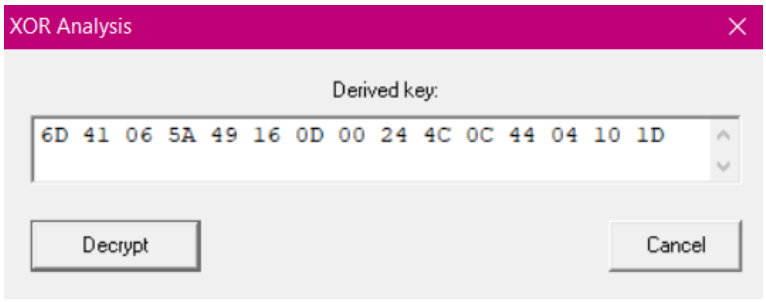


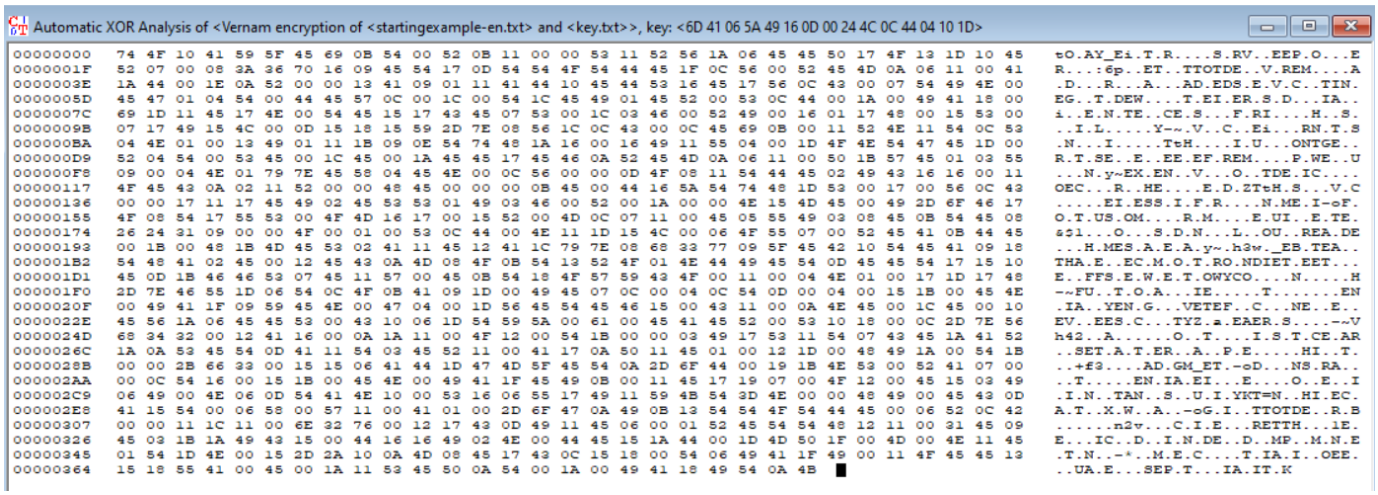


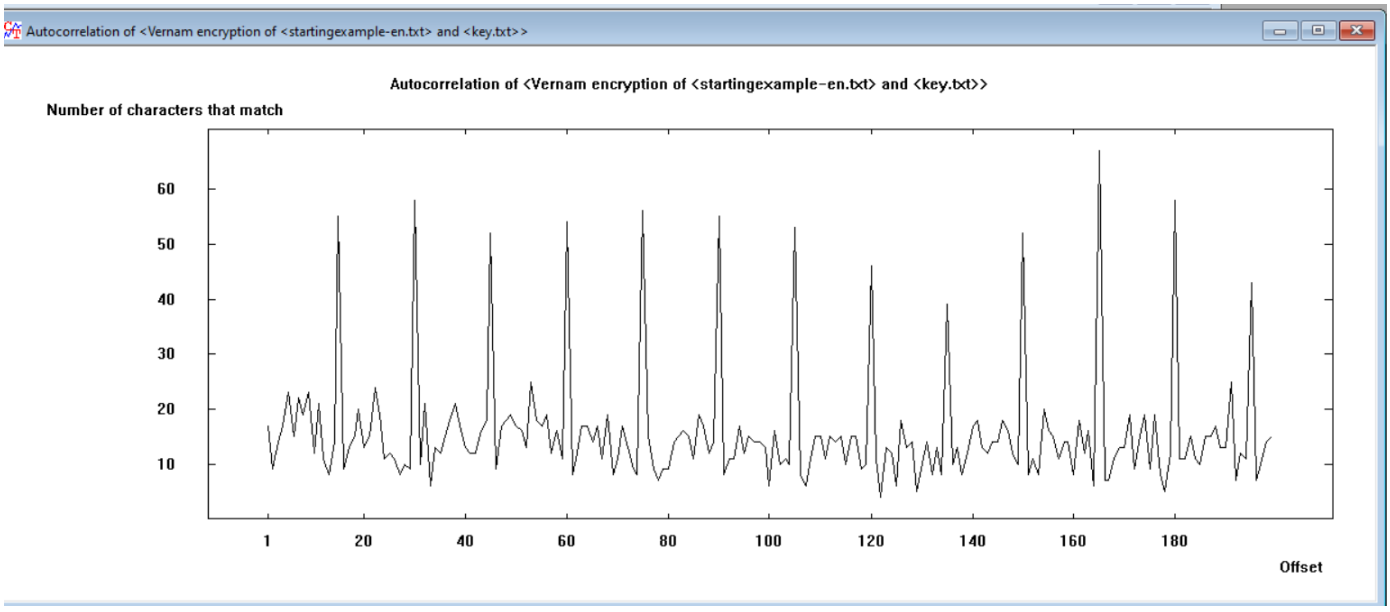


- **Part b (the short key):**









همانطور که مشخص است، در حالت اول که طول کلید بزرگتر مساوی طول plaintext بود، رمز شکسته نشد و طول کلید قابل حدس نبود و متن اصلی به دست نیامد. این نشان دهنده ی امنیت بالا در این حالت است.
در حالت دوم که طول کلید خیلی کوتاه بود نسبت به طول متن، رمز خیلی راحت تر شکسته شد وبخش قابل توجهی از متن اصلی به دست آمد. این نشان دهنده ی امنیت پایین در این حالت است.