Információ és kódelmélet
Otthoni feladat megoldás

Ekart Csaba [ZWPMKP]

2019. január 13.

1. Titkosítás

1.1. One-Time Pad

1.1.1. Bevezetés

Ebben a feladatban egy One-Time Pad (OTP) titkosító algoritmust kellett implementálni.

Az OTP különlegessége, hogy tökéletes titkosítást valósít meg, tehát a titkosított szöveg vizsgálata nem juttattja a a vizsgáló személyt új információhoz. Lényege, hogy a titkosítandó szövegből létrejött bitsorozat hosszával megegyező random bitsorozattal XOR-oljuk, majd ugyanezzel a random bitsorozattal visszafejthetjük. [1]

A feladat megoldásához Python nyelvet használtam, de az egyszerűség és a Windows-os kompatibilitás [2] kedvéért csak ASCII karaktereket tud titkosítani, illetve visszafejteni.

1.1.2. Használat

A program indítása után az alábbi opciók közül választhatunk:

1. Kódolás - az 1-es gomb lenyomásával
2. Dekódolás - a 2-es gomb lenyomásával
3. Kilépés - bármelyik másik gomb lenyomásával

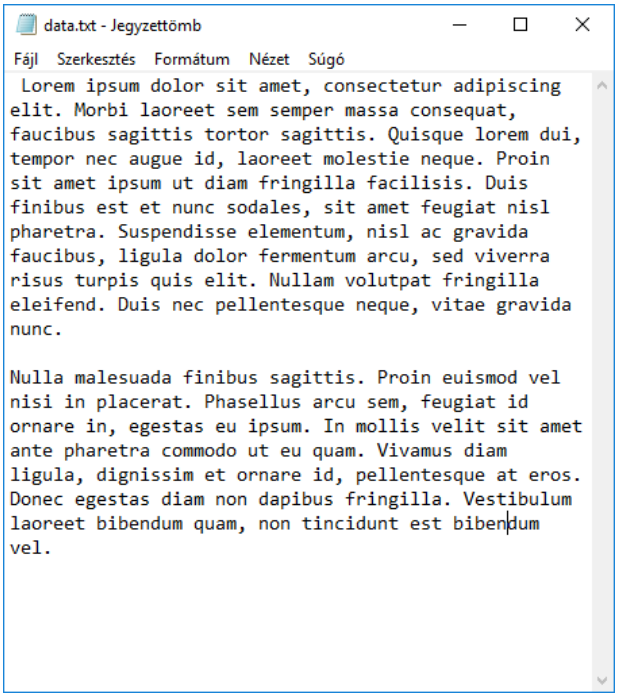
Kódolás választása esetén meg kell adni a programnak a fájlt, amelyet titkosítani szeretnénk. Sikeres futás esetén a kódolt szöveg az *"encrypted.txt"*, míg a visszafejtéshez szükséges kulcs a *"key.txt"* fájlba kerül.

Dekódolás esetén meg kell adni a titkosított fájl nevét, illetve a kulcsot (ez is egy külön fájl a konzisztencia kedvéért). Sikeres futtatás után a visszafejtett szöveg a *"decrypted.txt"* fájlba kerül. Ezután le lehet ellenőrizni, hogy a program működése a várakozásoknak megfelelt e.

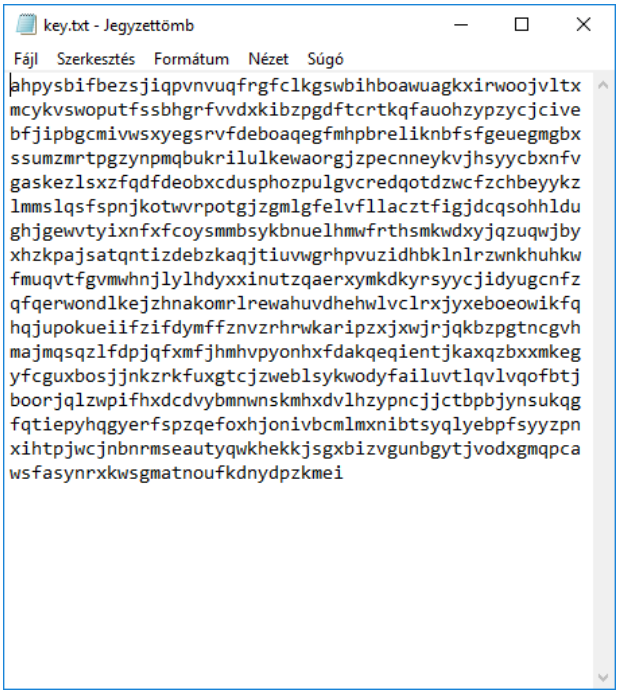
A teljesség kedvéért az alábbi képernyőképeken megtekinthetőek egy futtatás eredményei.

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Int
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\Users\Csaba\Documents\GitHub\one-time-pad\ASCII\onetimepad.py =
Helo egy OTP titkosito program vagyok!
Az alabbi opciok elerhetok:
 (1) kodolas
 (2) dekodolas
 (barmi mas) kilepes
1
Add meg a fajlnevet!
data.txt
A kodolt szoveg az encrypted.txt, a kulcs a key.txt fajlban talalhato.
Az alabbi opciok elerhetok:
 (1) kodolas
 (2) dekodolas
 (barmi mas) kilepes
2
Add meg a fajlnevet!
encrypted.txt
Add meg a kulcs fajlnevet!
key.txt
A dekodolt szoveg a decrypted.txt fajlban talalhato!
Az alabbi opciok elerhetok:
 (1) kodolas
 (2) dekodolas
 (barmi mas) kilepes
3
Viszlat!
>>>
```

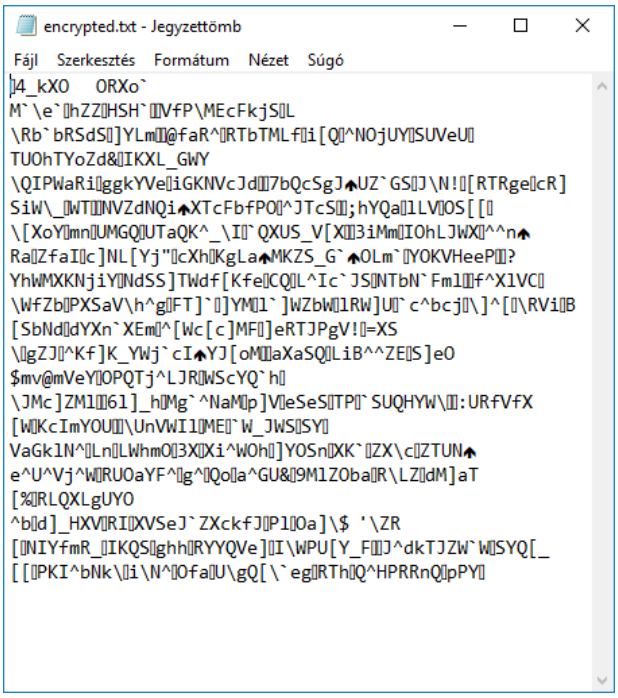
1. ábra. Az OTP program futtatás közben Windows alatt.



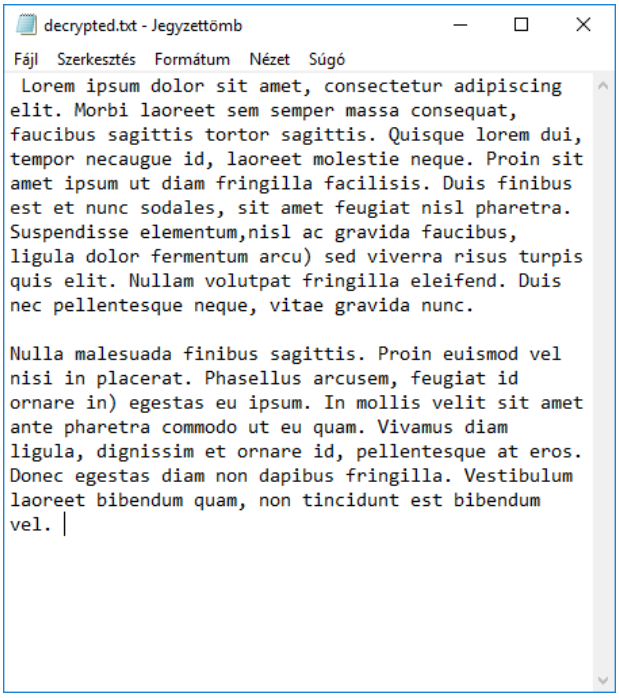
2. ábra. A bemeneti ASCII szöveg



3. ábra. Kulcs



4. ábra. A titkosított szöveg



5. ábra. A visszafejtett eredeti szöveg

1.1.3. Működés

A program működésének két érdekesebb pontja a titkosítás és a visszafejtés.

Az algoritmus alapja, hogy generálunk egy random karakterekből álló, de a bemeneti szöveggel megegyező hosszúságú kulcsot. Az én implementációmban ez a kulcs csak az angol ABC kisbetűiből áll az egyszerűség kedvéért, de könnyen továbbfejleszthető, ha "még titkosabb" kulcsot szeretnénk létrehozni.

```
17 def encrypt_file(filename):
18     with open(filename, 'r') as myfile:
19         data = myfile.read()
20
21     key = ''.join(random.choice(string.ascii_lowercase) for x in range(len(data)))
22
23     encrypted_data_list = list()
24
25     for i in range(len(data)):
26         encrypted_data_list.append(chr((ord(data[i]) + ord(key[i])) % 128))
27
28     print('A kodolt szoveg az encrypted.txt, a kulcs a key.txt fajlban talalhato. ')
29
30     with open('encrypted.txt', 'w') as myfile2:
31         myfile2.write(''.join(encrypted_data_list))
32
33     with open('key.txt', 'w') as myfile3:
34         myfile3.write(key)
35
```

6. ábra. A titkosításért felelős függvény

A bemeneti szöveg és a kulcs azonos pozíción lévő elemeit először számokká kell alakítani, amelyet a

python *ord()* függvényével könnyen meg lehet tenni, majd összeadni. Mivel a Windows terminálja csak ASCII karakterekkel boldogul az egyszerűség kedvéért ennek az összegnek vesszük a 128-cal vett osztási maradékát. majd visszalakítjuk karakterré.[2]

Az így kapott betűk sorozata hozza létre a titkosított szöveget.

A visszafejtés rendkívül hasonlóan működik, csak az összeadás helyett annak inverz műveletét, kivonást végzünk.

```
37 def decrypt_file(filename, key_filename):
38     with open(filename, 'r') as myfile:
39         data = myfile.read()
40 
41     with open(key_filename, 'r') as myfile2:
42         key = myfile2.read()
43 
44     decrypted_data_list = list()
45 
46     for i in range(len(data)):
47         decrypted_data_list.append(chr((ord(data[i]) - ord(key[i])) % 128))
48 
49     print('A dekodolt szoveg a decrypted.txt fajlban talalhato!')
50 
51     with open('decrypted.txt', 'w') as myfile3:
52         myfile3.write(''.join(decrypted_data_list))
53 
```

7. ábra. A visszafejtésért felelős függvény

1.2. RSA-eljárás

1.2.1. Bevezetés

Az RSA-eljárás nyílt kulcsú (vagyis „aszimmetrikus") titkosító algoritmus, melyet 1976-ban Ron Rivest, Adi Shamir és Len Adleman fejlesztett ki (és az elnevezést nevük kezdőbetűiből kapta). Az eljárás a prímtényezős felbontás komplexitásán alapszik.

Az általam írt program alkalmas privát és publikus kulcs generálására és egy maximum 128 karakteres ASCII szöveget képes titkosítani illetve dekódolni.

1.2.2. Használat

A program indítása után az alábbi opciók közül választhatunk:

1. Kulcsgenerálás - az 1-es gomb lenyomásával
2. Kódolás - az 2-es gomb lenyomásával
3. Dekódolás - a 3-es gomb lenyomásával
4. Kilépés - bármelyik másik gomb lenyomásával

Kulcsgenerálás opció esetén a publikus kulcs és a privát kulcs rendre a *"public_key.txt"* és a *"private_key.txt"* fájlba kerül. Ez két számot fog jelenteni vesszővel elválasztva mindkét esetben.

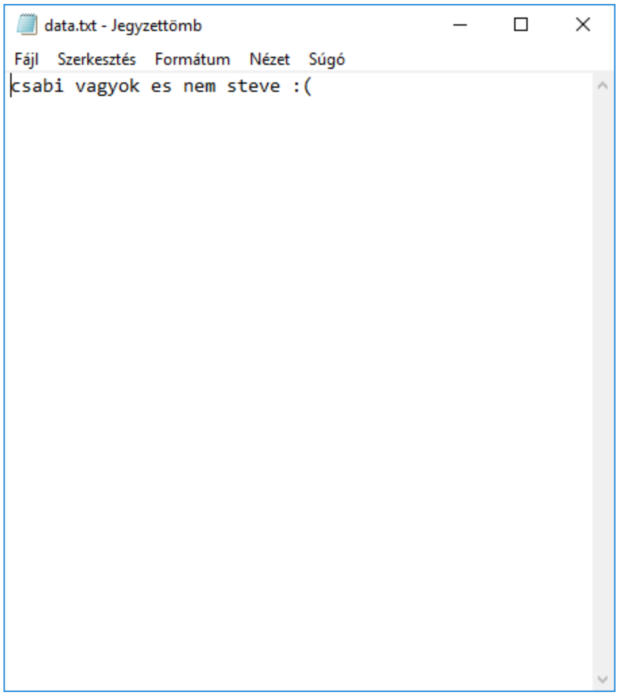
Kódolás választása esetén először a publikus kulcsot tartalmazó fájlt kell megadni, majd a fájlt, amelyet titkosítani szeretnénk. Sikeres futás esetén a kódolt szöveg az *"encrypted.txt"* fájlba kerül.

Dekódolás esetén először a privát kulcsot tartalmazó fájlt kell megadni, majd titkosított fájl nevét. Sikeres futtatás után a visszafejtett szöveg a *"decrypted.txt"* fájlba kerül. Ezután le lehet ellenőrizni, hogy a kódfejtés sikeres volt-e.

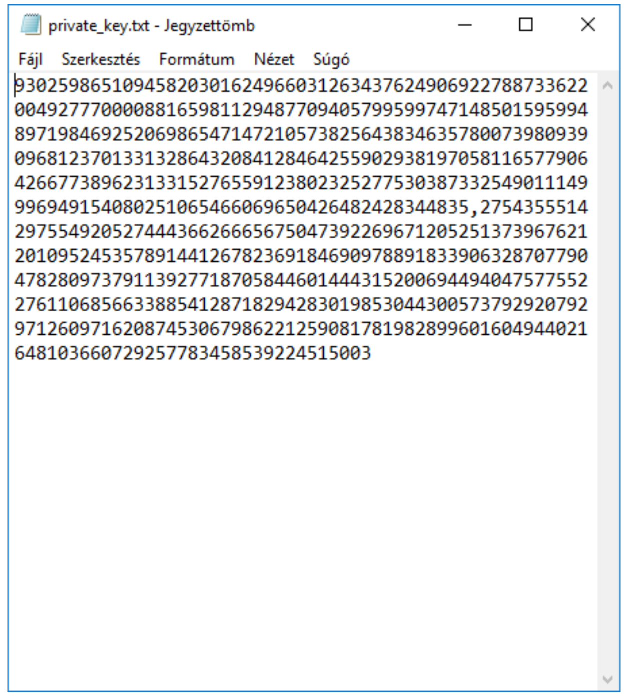
A teljesség kedvéért az alábbi képernyőképeken megtekinthetőek egy futtatás eredményei.

```
>>>
======== RESTART: d:\Users\Csaba\Documents\GitHub\rsa-python\main.py ========
Helo, en egy RSA titkosito program vagyok!
Az alabbi dolgokkal boldogulok:
 (1) kulcsgeneralas
 (2) kodolas
 (3) dekodolas
 (barmi mas) kilepes
1
A publikus kulcs a public_key.txt a privat kulcs a private_key.txt-ben van
Az alabbi dolgokkal boldogulok:
 (1) kulcsgeneralas
 (2) kodolas
 (3) dekodolas
 (barmi mas) kilepes
2
Add meg a publikus kulcsod tartalmazo fajl nevet!public_key.txt
Adj meg egy szoveg fajlt! (max 128 karakter)data.txt
A titkositott adat az encrypted.txt fileban van.
Az alabbi dolgokkal boldogulok:
 (1) kulcsgeneralas
 (2) kodolas
 (3) dekodolas
 (barmi mas) kilepes
3
Add meg a privat kulcsod!private_key.txt
Adj meg a titkositott adatot tartalmazo fajlt!encrypted.txt
A visszafejtett adat a decrypted.txt fileban van.
Az alabbi dolgokkal boldogulok:
 (1) kulcsgeneralas
 (2) kodolas
 (3) dekodolas
 (barmi mas) kilepes
4
Viszlat!
>>>
```

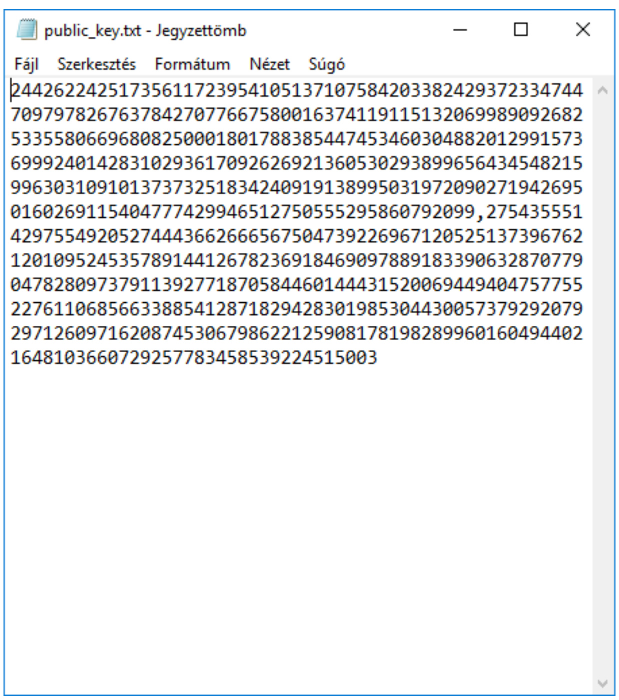
8. ábra. Az RSA program futtatás közben Windows alatt.



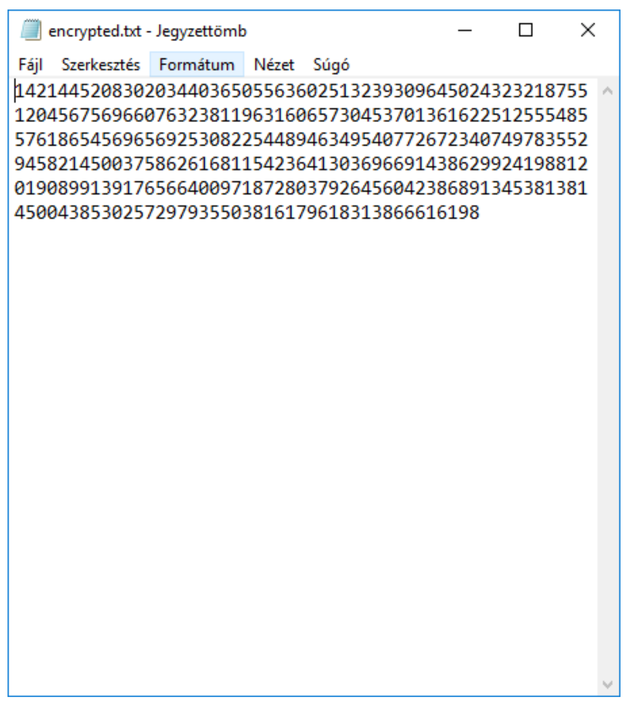
9. ábra. A bemeneti ASCII szöveg



10. ábra. Privát kulcs



11. ábra. Publikus kulcs



12. ábra. A titkosított szöveg

1.2.3. Működés

Az RSA működése lényegesen összetettebb az OTP-nél így a megvalósítása is nehezebb feladat volt.

Az egyszerűség kedvéért a program működését részfunkciói szerint fejtem ki.

Kulcsgenerálás

Az RSA kulcsgenerálásához először két nagy prímszámra van szükségünk. Mivel a prímek gyakorisága kielégítő, ezért használhatunk olyan módszert, mellyel egy véletlenszerű hatalmas számtól addig lépkedek előre, amíg prímet nem találok. Ehhez azonban egy olyan algoritmusra van szükség, amely nagyon gyorsan képes megmondani nagy számokról is, hogy prímek e. Ehhez a Miller-Rabin tesztet használtam, amely elég jó valószínűséggel nagyon gyorsan meg tudja határozni egy nagy számról, hogy prím e.[3] [4] Az élvezhető futási idő érdekében a programomban a generált prímek mérete 512 bit, gyakorlatban ennél nagyobbakat is használnak. Ezen korlátozás miatt alkalmas a program maximum 128 karakteres szöveg kódolására.

A Miller-Rabin teszt implementációja [5] k=40 [6] pontosság esetén.

```python
import random

# test if a large number is prime with Miller-Rabin algorithm
def miller_rabin(n, k = 40):
    if n == 2 or n == 3 :
        return True
    if n % 2 == 0 or n < 2:
        return False
    r = 0
    s = n - 1
    while s % 2 == 0:
        r += 1
        s = s // 2
    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = pow(a, s, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True

```

13. ábra. A Miller-Rabin teszt implementációja

A prímek létrehozása után a tanultak alapján generálhatjuk a privát és publikus kulcsokat. A privát kulcs generálásához a kiterjesztett euklideszi algoritmust használtam. [7]

```python
# the extended euclidean algorithm to find the private key
def extended_euclidean_alg(a, b):
    x, y = a, b
    t0, t1 = 0, 1
    r, t = b, 1
    while (x % y) != 0:
        q = x // y
        r = x % y
        t = (t0 - q*t1) % a
        x = y
        y = r
        t0 = t1
        t1 = t
    return t

```

14. ábra. A kiterjesztett euklideszi algoritmus

A kulcsgenerálás kódja:

```python
# generation of private and public key from two primes
def generate_keys():
    p, q = prime.generate_two_random_prime(512)
    n = p * q
    phi = (p-1)*(q-1)
    e = random.randrange(1, phi)
    while not coprime(e, phi):
        e = random.randrange(1, phi)
    d = extended_euclidean_alg(phi, e)
    return ((e, n), (d, n))

```

15. ábra. Kulcsgeneráló függvény

Titkosítás és visszafejtés

RSA esetén mind a titkosítás, mind a visszafejtés könnyen implementálható. Az egyetlen érdekes rész benne, hogy a szöveget egyértelműen számmá kell alakítanunk a műveletek elvégzéséhez, majd visszafejtés esetén a számot szöveggé.

Ezt az alábbi függvényekkel tudtam könnyen megtenni:

```python

# convert text to integer in one-to-one correspondence
def text_to_integer(text):
    return int.from_bytes(text.encode(), 'big')

# convert intteger to text in one-to-one correspondence
def integer_to_text(n):
    return n.to_bytes((n.bit_length() + 7) // 8, 'big').decode()

```

16. ábra.

Tulajdonképpen a szöveget először bináris kóddá alakítjuk, majd ezt a bináris számot alakítjuk tízes számrendszerbelire, hogy könnyen végezhessünk rajta műveleteket. [9] Visszaalakításnál a számot átváltjuk kettes számrendszerbeli alakra, majd ebből szöveget készítünk.

A titkosítás és visszafejtő függvények az alábbiak: [8]

```
41 # encrypyt max 128 length ascii text
42 def encrypt(public_key, text):
43     e, n = public_key
44     text_number = decimal_string.text_to_integer(text)
45     encoded = pow(text_number, e, n)
46     return encoded
47
```

17. ábra. RSA titkosító függvény

```
48 # decrypt ascii text
49 def decrypt(private_key, text_number):
50     d, n = private_key
51     decoded = pow(text_number, d, n)
52     return decimal_string.integer_to_text(decoded)
53
```

18. ábra. RSA visszafejtő függvény

Hivatkozások

[1] 86-87. oldal
Buttyán Levente, Györfi László, Győri Sándor, Vajda István: Kódolástechnika, 2006

[2] `https://stackoverflow.com/questions/5419/python-unicode-and-the-windows-console/32176732#32176732`

[3] `https://en.wikipedia.org/wiki/Generating_primes#Large_primes`

[4] `https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test`

[5] `https://inventwithpython.com/rabinMiller.py`

[6] urlhttps://stackoverflow.com/questions/6325576/how-many-iterations-of-rabin-miller-should-i-use-for-cryptographic-safe-primes

[7] 99. oldalon található pszeudokód alapján
Buttyán Levente, Györfi László, Győri Sándor, Vajda István: Kódolástechnika, 2006

[8] `https://gist.github.com/JonCooperWorks/5314103`

[9] `http://doctrina.org/How-RSA-Works-With-Examples.html`