

Κρυπτογραφία

Γέροντας Αλέξανδρος – 321/2015029

2^η Άσκηση

Σημείωση: Όλα τα ζητήματα υλοποιήθηκαν σε λειτουργικό Windows 10 με το gnuMP library εγκαταστημένο από το Msys2. Στα linux δεν έγιναν αρκετές δοκιμές αλλά ο κώδικας τρέχει κανονικά.

Ζήτημα 1 – Αλγόριθμος κρυπτογράφησης RSA

Αρχικά δημιουργούμε μία συνάρτηση (convert_plaintext_to_ascii) η οποία μετατρέπει το αρχικό μήνυμα σε έναν ακέραιο. Συγκεκριμένα η συνάρτηση παίρνει σαν όρισμα το plaintext και έναν ακέραιο mpz με όνομα ascii_plaintext. Για κάθε χαρακτήρα του plaintext, μετατρέπουμε τον χαρακτήρα σε ascii με την συνάρτηση sprintf και τον αποθηκεύουμε σε έναν προσωρινό δείκτη τύπου char. Αφού μετατρέψουμε όλους τους χαρακτήρες σε ascii, μετατρέπουμε τον δείκτη σε ακέραιο mpz και τον αποθηκεύουμε στην μεταβλητή ascii_plaintext.

Στην συνέχεια δημιουργούμε μια συνάρτηση η οποία δημιουργεί δημόσια και ιδιωτικά κλειδιά. Πρώτα δημιουργεί έναν τυχαίο αριθμό 512 bit (Τα bit δίνονται σαν όρισμα στην συνάρτηση) και με βάση τους επόμενους πρώτους από αυτόν δημιουργούμε τους ακέραιους p,q. Στη συνέχεια υπολογίζουμε το n και το φ, και βρίσκουμε έναν ακέραιο e ο οποίος να αντιστρέφεται στο Ζφ – γίνεται επανάληψη μέχρι να βρεθεί και υπολογίζεται και ο αντίστροφος d.

Για την κρυπτογράφηση του μηνύματος μετατρέπουμε το μήνυμα σε ακολουθία ascii (m) με την χρήση της συνάρτησης convert_plaintext_to_ascii. Ο υπολογισμός του ciphertext c γίνεται με τον τύπο $c = m^e \bmod n$.

Στην αποκρυπτογράφηση υπολογίζουμε το ascii_plaintext με τον τύπο $\text{ascii_plaintext} = c^d \bmod n$. Μετατρέπουμε το ascii_plaintext πίσω στο αρχικό μήνυμα με την συνάρτηση convert_ascii_to_plaintext η οποία ακολουθεί την αντίστροφη διαδικασία της convert_plaintext_to_ascii.

Κατά την εκτέλεση το πρόγραμμα τυπώνει τα ιδιωτικά και δημόσια κλειδιά το αρχικό κείμενο το αρχικό κείμενο σε ascii, το κρυπτοκείμενο και το αρχικό κείμενο μετά την αποκρυπτογράφηση.

```
Run - untitled1
Run: program x Rsa x
"C:\Users\Alexander\Desktop\Crypt tmp\untitled1\cmake-build-debug\rsa.exe"
-----
public key -----
n: 532570074847251358641279556850206384437708232508033552231931248877658164457444245506814420357205291442331452251439722
751099915968457029399222054020887917134902544605412951868036365528650684783532976083829022954981004070816329931744432681
4635226726607127918338161885464344531306206570832928174041468659875943
e: 17

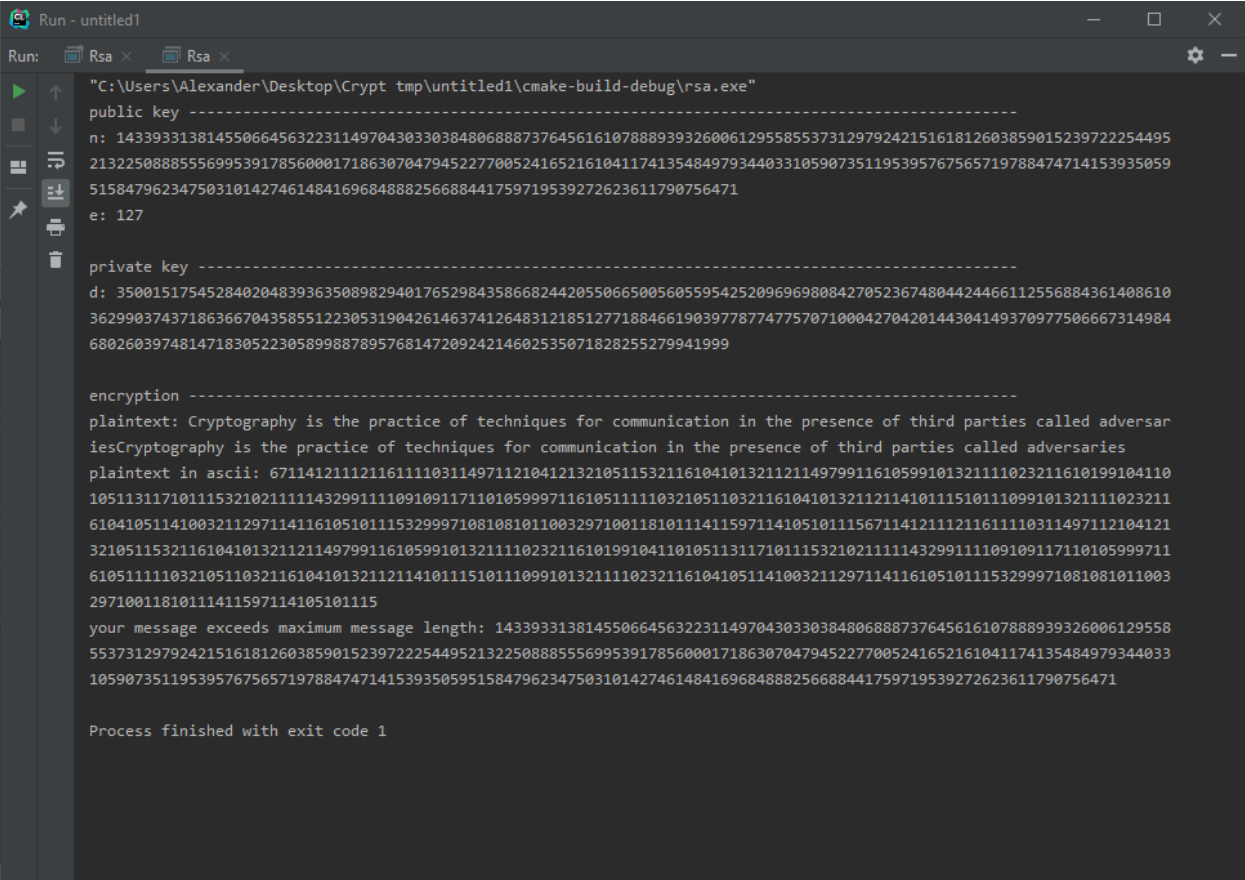
private key -----
d: 281948863154427189868912706567756321172904358386605998240434190582289616477470482915372340189108683704763710015468088
515288190806830192034882263893411250003539796107659701225497353349895696206609122154586823853197386499170401083541215563
2551456509011496751066091613071539691691436908991045082634996523947689

encryption -----
plaintext: Cryptography is the practice of techniques for communication in the presence of third parties called adversar
ies
plaintext in ascii: 671141211121161110311497112104121321051153211610410132112114979911610599101321111023211610199104110
10511311710111532102111143299111109109117110105999711610511111032105110321161041013211211410111510111099101321111023211
61041051141003211297114116105101115329997108108101100329710011810111411597114105101115
ascii plaintext length in bits: 1016
encrypted message: 35608427301801339642266669521686013812169398450990134922204690382654460018195248106966797445765508691
946202040843291464916661342340642626396700782415079828058803260403858228065693120809749951326300496905922271561631642793
41430701362941271703124853824235291857604475283575875942376803415619799009152771690259

decryption -----
ascii plaintext after decryption: 6711412111211611103114971121041213210511532116104101321121149799116105991013211110232
1161019910411010511311710111532102111143299111109109117110105999711610511111032105110321161041013211211410111510111099101321111023211
0132111102321161041051141003211297114116105101115329997108108101100329710011810111411597114105101115
plaintext after decryption: Cryptography is the practice of techniques for communication in the presence of third partie
s called adversaries

Process finished with exit code 0
|
```

Εάν δοθεί κείμενο του οποίου η ascii μορφή να ξεπερνάει το διάστημα 0, n-1 το πρόγραμμα τυπώνει μήνυμα αποτυχίας και τερματίζει.



Σημείωση: Οι συναρτήσεις convert_plaintext_to_ascii και convert_ascii_to_plaintext χρησιμοποιούνται σε όλα τα παρακάτω ζητήματα.

Ζήτημα 2 – Αλγόριθμος κρυπτογράφησης El Gamal

Με την συνάρτηση gen_keys δημιουργούμε το δημόσιο κλειδί p,a,d και το ιδιωτικό d. Συγκεκριμένα βρίσκουμε τον επόμενο πρώτο ενός τυχαίου αριθμού για να υπολογίσουμε το p. Καθώς το p είναι πρώτος η συνάρτηση φ(p) μας δίνει πάντα p-1. Για τον υπολογισμό του γεννήτορα του p κάνουμε επανάληψη με μία while μέχρι να βρεθεί ένας ακέραιος a τέτοιος ώστε $a^{\phi(p)} \equiv 1 \pmod p$. Στη συνέχεια δημιουργούμε έναν ακέραιο d 6 bit και υπολογίζουμε το a^d .

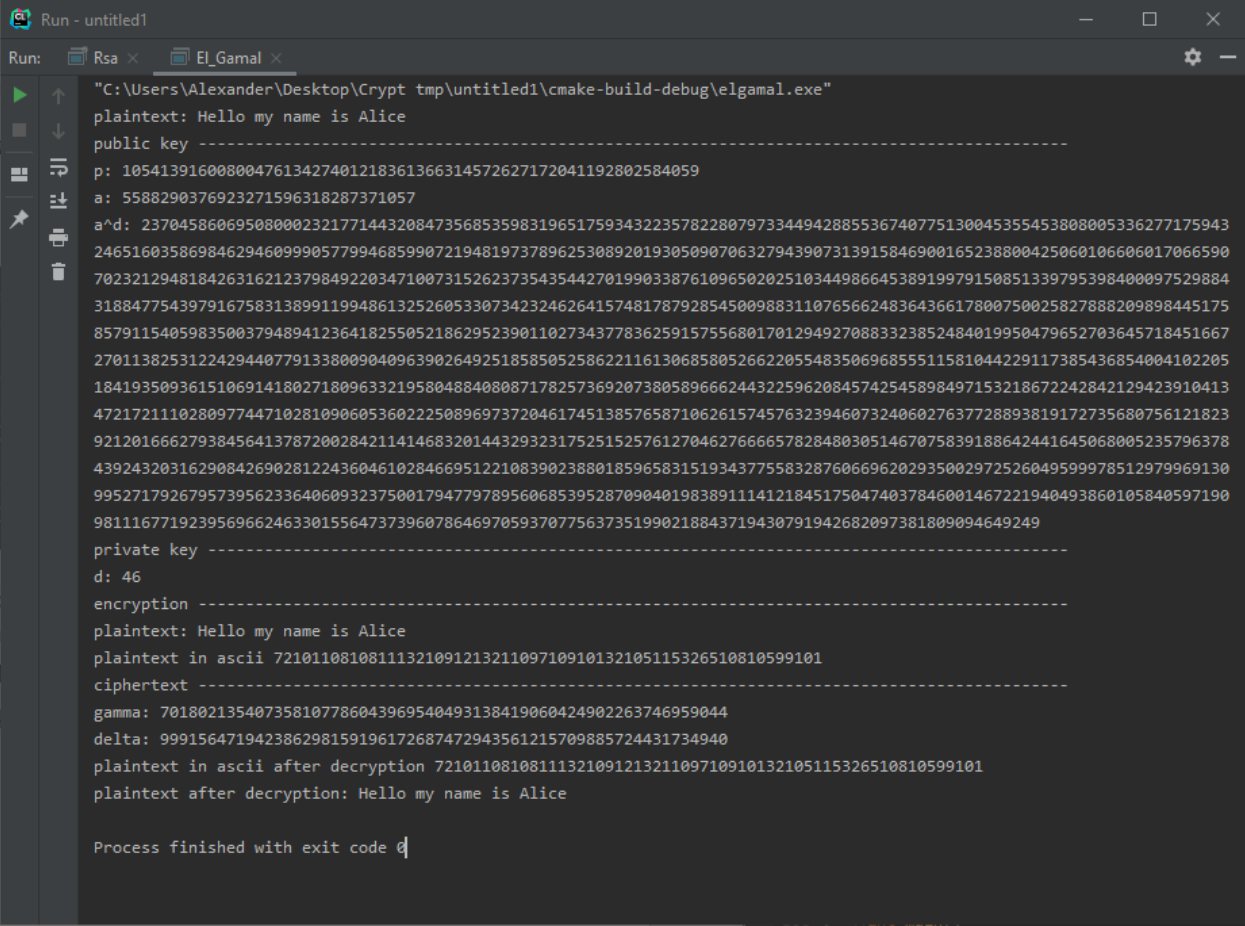
Για την κρυπτογράφηση του μηνύματος με την συνάρτηση encryptMsg μετατρέπουμε πρώτα το αρχικό μήνυμα σε ascii με την συνάρτηση convert_plaintext_to_ascii. Εάν το μήνυμα ascii ξεπερνάει το p στο δεκαδικό το πρόγραμμα τυπώνει μήνυμα αποτυχίας και τερματίζει. Δημιουργούμε έναν τυχαίο k 7 bit και υπολογίζουμε το κρυπτοκείμενο γ,δ ως εξής:

```
// γ = α^k mod p.
mpz_powm(gamma, a, k, p);

// tmp = α^dk
mpz_pow_ui(tmp, a_d, mpz_get_ui(k));
mpz_mul(tmp, tmp, ascii_plaintext);

// δ = m*α^dk mod p
mpz_mod(delta, tmp, p);
```

Στην συνάρτηση decryptMsg αποκρυπτογραφούμε το μήνυμα. Υπολογίζουμε το $c = \gamma^{p-1-d} \pmod p$ και το $m = c * \delta \pmod p$ για να πάρουμε το αρχικό κείμενο σε ascii. Τέλος με την συνάρτηση convert_ascii_to_plaintext μετατρέπουμε το αρχικό μήνυμα σε ascii στο αρχικό μήνυμα.



Ζήτημα 3 – Αλγόριθμος Κρυπτογράφησης Rabin

Για την δημιουργία των κλειδιών φτιάχνουμε μία random μεταβλητή 200 bit και παίρνουμε τους επόμενους πρώτους (p, q) μετά από αυτήν. Καθώς για τους p και q πρέπει να ισχύει ότι $p \equiv 3 \pmod 4$ και $q \equiv 3 \pmod 4$ αυτή η διαδικασία γίνεται σε επανάληψη μέχρι να τηρηθεί ο περιορισμός. Στη συνέχεια υπολογίζεται το n -> $n = p \cdot q$. (συνάρτηση gen_keys)

Για την κρυπτογράφηση του κειμένου με την συνάρτηση πρώτα μετατρέπουμε το αρχικό μήνυμα σε ascii με την συνάρτηση convert_plaintext_to_ascii. Μετατρέπουμε το ascii μήνυμα στο δυαδικό, και το τροποποιούμε έτσι ώστε να επαναλαμβάνονται τα τελευταία 6 bit του. Στην συνέχεια το μετατρέπουμε ξανά σε δεκαδικό, ελέγχουμε αν ξεπερνάει το n (αν ναι το πρόγραμμα τερματίζει) και υπολογίζουμε την τιμή $c = m^2 \pmod n$, όπου m το τροποποιημένο ascii plaintext. (συνάρτηση encryptMsg)

Στον υπολογισμό των ριζών βρίσκουμε αρχικά δύο αριθμούς a,b έτσι ώστε $ap + bq = 1$ με την χρήση της έτοιμης συνάρτησης mpz_gcdext και υπολογίζουμε τις τέσσερις ρίζες με την χρήση των τύπων:

1.

Βρίσκει ακεραίους a και b τέτοιους ώστε $ap + bq = 1$.
2.

Υπολογίζει τις τιμές $r = c^{(p+1)/4} \pmod p$ και $s = c^{(q+1)/4} \pmod q$.
3.

Υπολογίζει $x = (aps + bqr) \pmod n$.
4.

Υπολογίζει $y = (aps - bqr) \pmod n$.
5.

Οι 4 ρίζες του $c \pmod n$ είναι οι x, $-x \pmod n$, y και $-y \pmod n$.

Αφού τις υπολογίζουμε τις καταχωρούμε σε έναν πίνακα. (συνάρτηση calculate_roots).

Τέλος για την αποκρυπτογράφηση του μηνύματος μετατρέπουμε τις ρίζες στο δυαδικό και ελέγχουμε τα τελευταία τους bit. Αυτό γίνεται σε επανάληψη μέχρι να βρεθεί μία ρίζα όπου τα τελευταία 6 bit της να επαναλαμβάνονται. Όταν βρεθεί αυτή η ρίζα αφαιρούμε τα bit και την μετατρέπουμε σε δεκαδικό. Μετατρέπουμε την ρίζα στο δεκαδικό και με την συνάρτηση convert_ascii_to_plaintext παίρνουμε το αρχικό μήνυμα.(συνάρτηση decryptMsg)

```
Run - untitled1
Run:  Rsa  Rabin
"C:\Users\Alexander\Desktop\Crypt tmp\untitled1\cmake-build-debug\rabin.exe"
public key -----
n: 869374308404148268232094918039778093487244218101277495895937694480591880112374854179370242535119635776994273258417185
981
private key -----
p: 932402439080973376360686584583531262348978714530618924556511
q: 932402439080973376360686584583531262348978714530618924556771

encryption -----
plaintext:
Hello my name is Alice
My best friend is Bob
plaintext in ascii:
7210110810811132109121321109710910132105115326510810599101307712132981011151163210211410510111010032105115326611198
ascii plaintext in binary:
1011101101100001010100001110100011100111001001001110000001110110101101000110010011011001111100011111101110111110000
000000011000110010110010110111101000001010110001011001111110111111000010110000010101010001011000101010001000010011
11001011101100101011000011110011001100000010011011011101101000011111111010001100011000000111000101101001001111100100111
001011111111011111110
ascii plaintext in binary with extra bits:
101110110110000101010000111010001110011100100100111000000111011010110100011001001101100111110001111111011101111110000
000000011000110010110010110111101000001010110001011001111110111111000010110000010101010001011000101010001000010011
1100101110110010101100001111001100110000001001101101101101000011111111010001100011000000111000101101001001111100100111
0010111111110111111011110
ascii plaintext in decimal with extra bits:
461447091891912454983764551021498248454727380896691878342483693576510784713674445453530272647104642054727380903116734
ciphertext:
571124570981579383014567347790497002308945099628044461394781501425714482522827091772372583625977904904558905303353665518

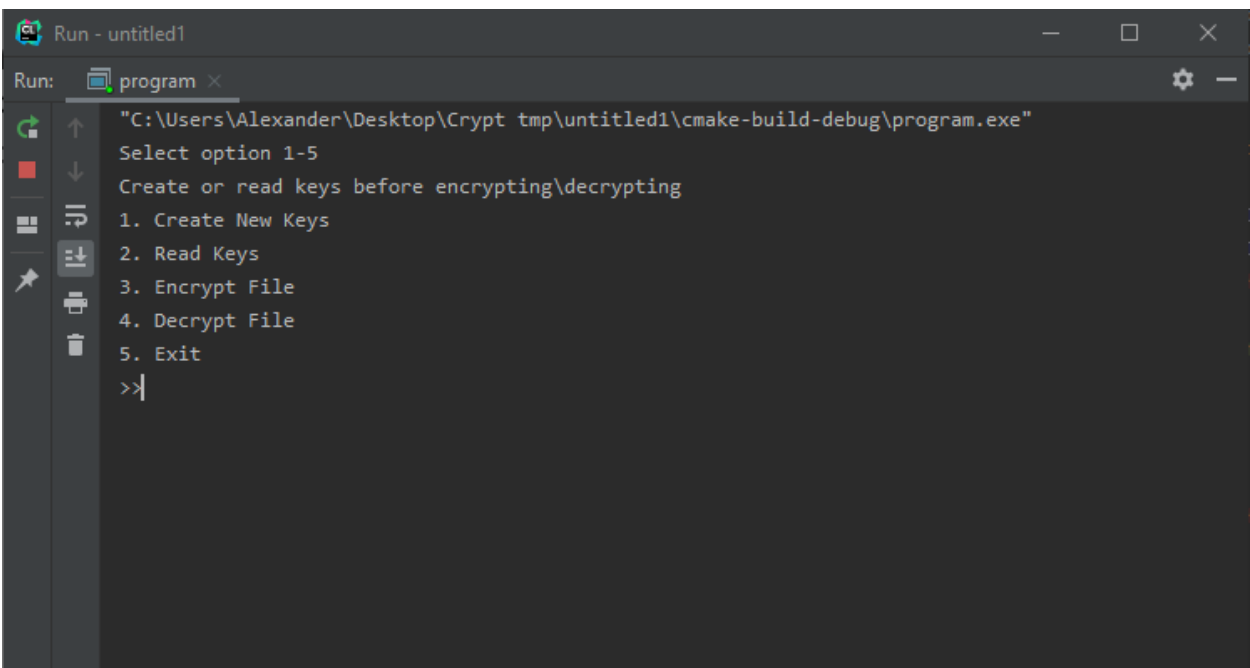
decryption -----
last 6 bits of root 3 are replicated
root in binary:
10111011011000010101000011101000111001110010010011100000011101101011010001100100110110011111000111111011101111110000
000000011000110010110010110111101000001010110001011001111110111111000010110000010101010001011000101010001000010011
1100101110110010101100001111001100110000001001101101101101000011111111010001100011000000111000101101001001111100100111
0010111111110111111011110
last bits 1: 111110
last bits 2: 111110
decrypted plaintext in ascii:
7210110810811132109121321109710910132105115326510810599101307712132981011151163210211410510111010032105115326611198
plaintext after decryption: Hello my name is Alice
My best friend is Bob

Process finished with exit code 0
```


Ζήτημα 4 – Κρυπτογράφηση αρχείου κειμένου

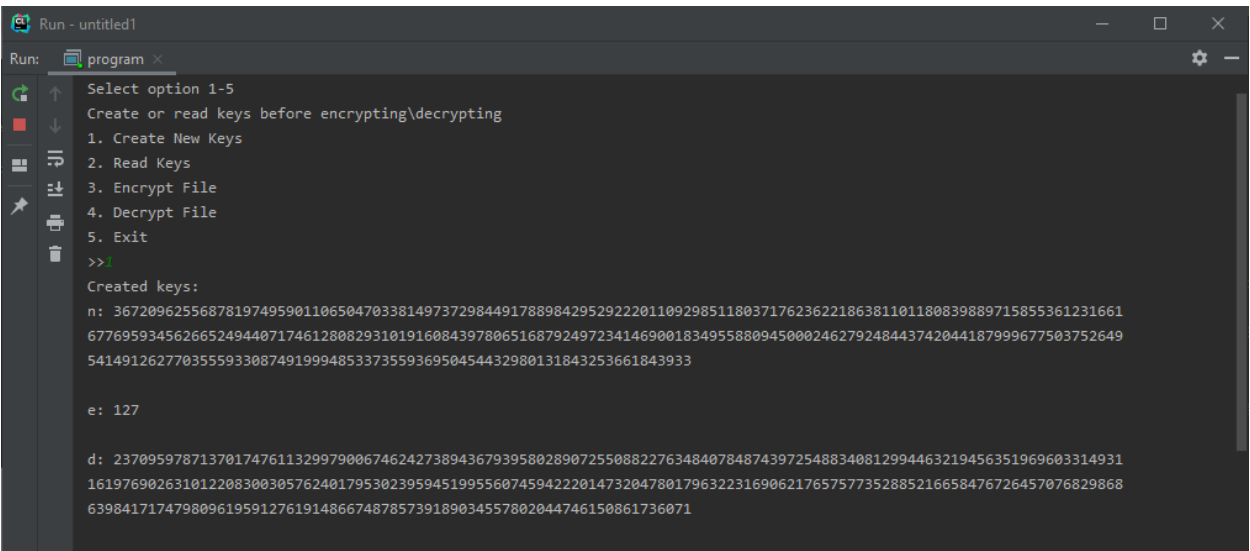
Στο 4^ο Ζήτημα για το πρόγραμμα που κρυπτογραφεί ένα αρχείο κειμένου επέλεξα τον αλγόριθμο RSA. Επίσης χρειάστηκε να υλοποιήσω ακόμα μία συνάρτηση. Η `getLine` δέχεται σαν όρισμα έναν δείκτη σε αρχείο και επιστρέφει μία γραμμή κειμένου. Διαβάζει από το αρχείο έναν-έναν τους χαρακτήρες και τους καταχωρεί στον δείκτη `line`. Με την χρήση της `realloc` προσαρμόζει το μέγεθος του `line` δίνοντας του περισσότερη μνήμη όποτε χρειάζεται. Η συνάρτηση `getLine` καλείται επαναληπτικά από το πρόγραμμα για να διαβάσει το `plaintext` ή το `ciphertext` και τα αρχείο με τα κλειδιά.

Το πρόγραμμα δίνει στον χρήστη τέσσερις επιλογές:



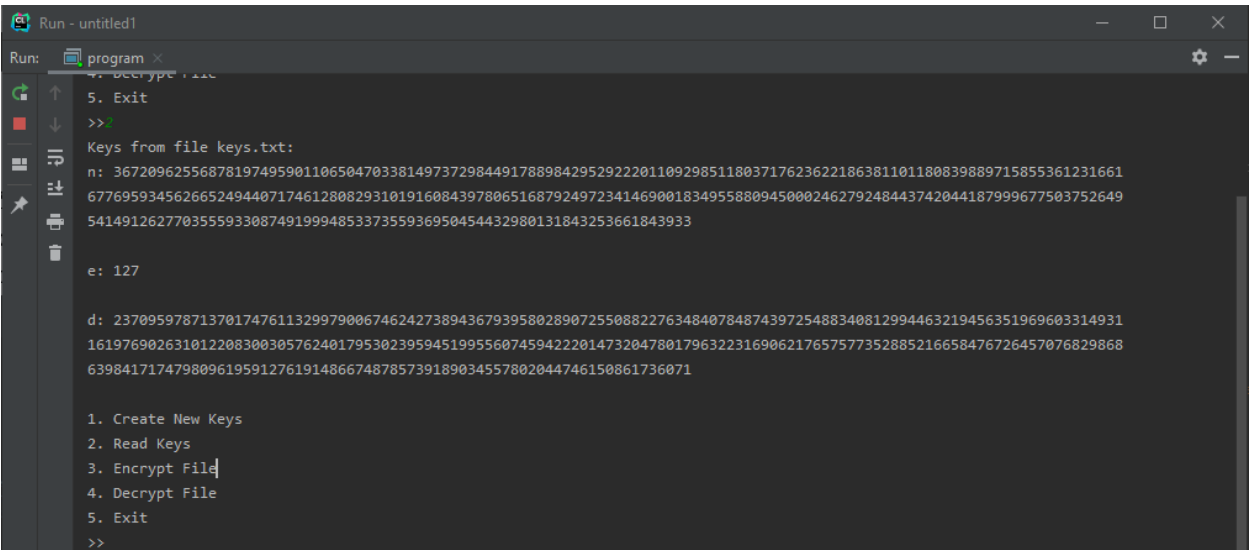
- Create New Keys

Το πρόγραμμα δημιουργεί τα ιδιωτικά και δημόσια κλειδιά `n`, `e`, `d`. Μετά την δημιουργία τους καταχωρούνται στο αρχείο `keys.txt` το οποίο βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο.



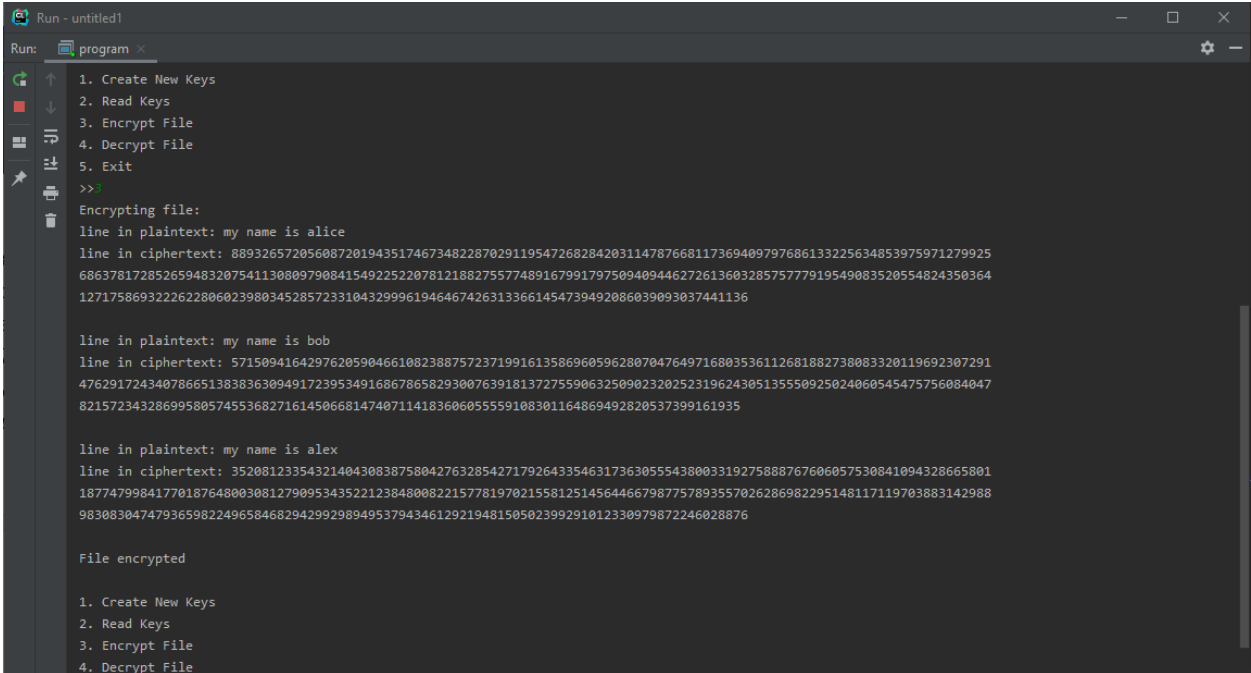
- Read Keys

Το πρόγραμμα διαβάζει από το αρχείο `keys.txt` τα κλειδιά που έχουν δημιουργηθεί στην 1^η επιλογή και τα καταχωρεί στις αντίστοιχες μεταβλητές `n`, `e`, `d`.



- Encrypt File

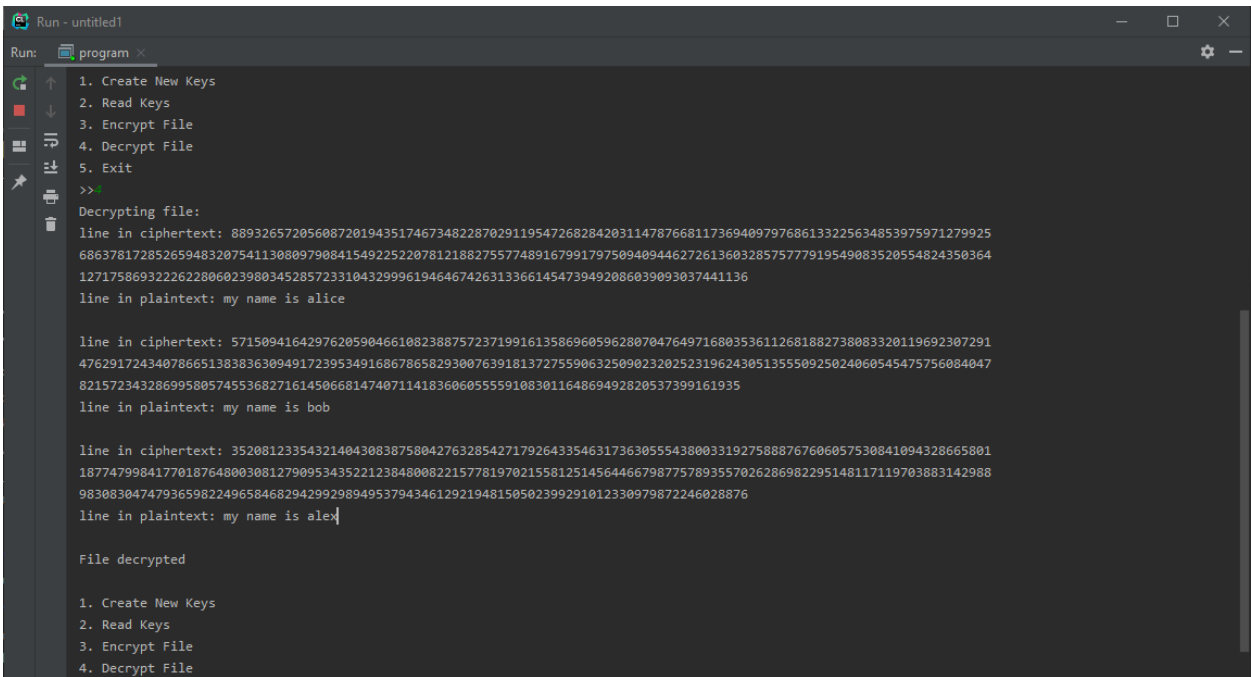
Το πρόγραμμα διαβάζει το αρχείο `plaintext.txt`, κρυπτογραφεί την κάθε γραμμή του και την καταχωρεί στο αρχείο `ciphertext.txt`. Το αρχείο `plaintext.txt` πρέπει να δημιουργηθεί από τον χρήστη στον ίδιο φάκελο με το εκτελέσιμο (Στα `linux` πρέπει να ονομαστεί "`.\plaintext.txt`"). Επίσης ο χρήστης θα πρέπει πρώτα να έχει δημιουργήσει τα κλειδιά έχοντας επιλέξει μία από τις παραπάνω επιλογές.



- Decrypt File

Το πρόγραμμα αποκρυπτογραφεί το αρχείο ciphertext.txt και καταχωρεί το αποκρυπτογραφημένο κείμενο στο αρχείο decrypted_plaintext.txt το οποίο επίσης βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο. Για την αποκρυπτογράφηση απαιτούνται φυσικά τα ίδια κλειδιά που δημιουργήθηκαν με την 1^η ή 2^η επιλογή αλλιώς η αποκρυπτογράφηση δεν γίνεται σωστά.

Αποκρυπτογράφηση με σωστά κλειδιά



Αποκρυπτογράφηση με λάθος κλειδιά

