



Σχολή Θετικών Επιστημών & Τεχνολογίας

Πληροφορική

Προστασία και Ασφάλεια Συστημάτων Υπολογιστών (ΠΛΗ-35)

3^η Γραπτή Εργασία 2021-22

Προβλήματα στη δημιουργία κλειδιών RSA και κίνδυνοι από την εκμετάλλευση ευπαθών υλοποιήσεων

Καραγκούνης Λεωνίδας Χρήστος ΑΜ: 114163

Τμήμα: ΗΛΕ-42, ΣΕΠ Μαυρίδης Ιωάννης

Πίνακας Περιεχομένων

Πίνακας Εικόνων	ii
1. Εισαγωγή	1
2. Υπόβαθρο.....	2
2.1 Αλγόριθμος Rivest-Shamir-Adleman (RSA)	2
2.2 Λειτουργία αλγορίθμου RSA	2
2.3 Προβλήματα κατά την δημιουργία κλειδίων RSA.....	4
3. Περιγραφή λύσεων	7
3.1 Ερώτημα Α.....	7
3.2 Ερώτημα Β	9
3.3 Ερώτημα Γ.....	13
3.4 Ερώτημα Δ	14
3.4.1 Υποερώτημα 1	14
3.4.2 Υποερώτημα 2	15
3.4.3 Υποερώτημα 3	16
4. Συμπεράσματα	17
Βιβλιογραφία	18
Παράρτημα Α.....	21
Παράρτημα Β.....	25

Πίνακας Εικόνων

Εικόνα 1. Αλγόριθμος RSA (Stallings, 2011, p.280)	3
Εικόνα 2. Παραγοντοποιήσεις για αριθμούς RSA (Buell, 2021, p.154)	4
Εικόνα 3. Παραλλαγή μικρού θεωρήματος Fermat (Wang, 2021).....	5
Εικόνα 4. Κριτήριο πιστοποίησης Miller-Rabin (Burton, 2011, p.369).....	5
Εικόνα 5. Αλγόριθμος των Joye και Paillier (Joye and Paillie, 2006).....	6
Εικόνα 6. Αποτελέσματα συνάρτησης GE3_A	7
Εικόνα 7. Απαιτούμενα bit για x, r, p	8
Εικόνα 8. Διάγραμμα διαφορών σε bit $x-r$	8
Εικόνα 9. Χρόνοι ανά βήμα αρχικού κώδικα	9
Εικόνα 10. Χρόνοι ανά βήμα συνάρτησης fast_gen_prime.....	10
Εικόνα 11. Χρόνοι βημάτων 2-20 συνάρτησης fast_gen_prime	10
Εικόνα 12. Παράλληλο γράφημα χρόνων ανα βήμα	11
Εικόνα 13. Συνάρτηση καλύτερης προσαρμογής αρχικού κώδικα	12
Εικόνα 14. Συνάρτηση καλύτερης προσαρμογής fast_gen_prime	12
Εικόνα 15. Παράγοντες modulus υπ' αριθμόν 7	14
Εικόνα 16. Παράγοντες modulus υπ' αριθμόν 10	14
Εικόνα 17. Αλγόριθμος κοινών παραγόντων.....	15
Εικόνα 18. Αποτελέσματα εύρεσης αποτυπωμάτων ευπάθειας ROCA	16
Εικόνα 19. Χρόνοι ανά βήμα αρχικού κώδικα	26
Εικόνα 20. Χρόνοι ανά βήμα συνάρτησης fast_gen_prime.....	27
Εικόνα 21. Χρόνοι βημάτων 2-20 συνάρτησης fast_gen_prime	28
Εικόνα 22. Συνάρτηση καλύτερης προσαρμογής αρχικού κώδικα	29
Εικόνα 23. Συνάρτηση καλύτερης προσαρμογής fast_gen_prime	30
Εικόνα 24. Γράφημα χρόνων συναρτήσεων ανά βήμα.....	31
Εικόνα 25. Παράλληλο γράφημα συνολικών χρόνων ανα βήμα.....	32
Εικόνα 26. Παράλληλο γράφημα χρόνων ανα βήμα	33

1. Εισαγωγή

Η ανάγκη για ασφάλεια επικοινωνίας, μετάδοσης πληροφοριών και διατήρησης προσωπικών δεδομένων είναι προβλήματα που έχουν απασχολήσει την ανθρωπότητα από την αρχαιότητα μέχρι σήμερα. Με την πάροδο των χρόνων αναπτύχθηκαν διάφορες κρυπτογραφικές μέθοδοι από τις σκυτάλες των αρχαίων Ελλήνων, την κρυπτογράφηση του Ιουλίου Καίσαρα και την μηχανή Enigma των Γερμανών κατά τον Β' Παγκόσμιο Πόλεμο έως τις σημερινές μεθόδους συμμετρικής και ασύμμετρης κρυπτογραφίας (symmetric and asymmetric cryptography), κατακερματισμού (hashing) και την κρυπτογραφία ελλειπτικών καμπυλών (elliptic curve cryptography).

Βασικές ιδιότητες οποιασδήποτε μεθόδου κρυπτογράφησης είναι η αξιοπιστία και η ασφάλειά της. Κατά την προσπάθεια βελτιστοποίησης κρυπτογραφικών διαδικασιών με στόχο την εξοικονόμηση χρόνου και πόρων, η χρήση συντομεύσεων μπορεί να οδηγήσει σε σοβαρά προβλήματα. Στην παρούσα εργασία θα ασχοληθούμε με τέτοιου είδους προβλήματα, συγκεκριμένα με αυτά που εμφανίζονται κατά τη διαδικασία δημιουργίας κλειδιών του αλγορίθμου κρυπτογράφησης RSA.

2. Υπόβαθρο

2.1 Αλγόριθμος Rivest-Shamir-Adleman (RSA)

Ο αλγόριθμος RSA ήταν το πρώτο δημόσια γνωστό κρυπτοσύστημα δημοσίου κλειδιού (public key cryptosystem). Παρουσιάστηκε σε άρθρο του Scientific American από τον Gardner (1977) και η πλήρης ερευνητική εργασία δημοσιεύτηκε ένα χρόνο αργότερα από τους εφευρέτες του Rivest, Shamir και Adleman (1978) ενώ σήμερα είναι το πιο ευρέως χρησιμοποιούμενο ασύμμετρο κρυπτογραφικό σχήμα, παρόλο που οι ελλειπτικές καμπύλες (elliptic curves) και τα κρυπτοσυστήματα διακριτού λογαρίθμου (discrete logarithm cryptosystems) κερδίζουν έδαφος. Πρέπει να σημειωθεί ότι το κρυπτοσύστημα RSA δεν αντικαθιστά συνήθως τα συμμετρικά κρυπτοσυστήματα. Αυτό οφείλεται στους πολλούς υπολογισμούς του αλγορίθμου, όπως θα δούμε παρακάτω. Έτσι, η κύρια χρήση του είναι συνήθως η ασφαλής ανταλλαγή κλειδιού (key exchange) για ένα συμμετρικό κρυπτοσύστημα όπως το AES, το οποίο πραγματοποιεί την κρυπτογράφηση (encryption) των δεδομένων (Paar and Pelzl, 2010, p.174).

2.2 Λειτουργία αλγορίθμου RSA

Το δημόσιο¹ και το ιδιωτικό² κλειδί (public and private key) του αλγορίθμου, τα οποία είναι μεγέθους άνω των 1024 bit ή 309 δεκαδικών ψηφίων, δημιουργούνται με τα παρακάτω βήματα όπως φαίνεται και στην Εικόνα 1:

¹ modulus n και δημόσιος εκθέτης e

² μυστικός εκθέτης d

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Εικόνα 1. Αλγόριθμος RSA (Stallings, 2011, p.280)

1. Επιλέγονται μεγάλοι τυχαίοι πρώτοι αριθμοί p και q διάφοροι μεταξύ τους (ιδιωτικά, επιλεγμένα).
2. Υπολογίζεται το $n = p \cdot q$ και συνάρτηση $\phi(n) = (p-1)(q-1)$ του Euler (δημόσια, υπολογισμένα).
3. Επιλέγεται ακέραιος e έτσι ώστε $1 < e < \phi(n)$ και $e, \phi(n)$ πρώτοι μεταξύ τους (δημόσιο, επιλεγμένο)
4. Υπολογίζεται το $d \equiv e^{-1} \pmod{\phi(n)}$ (ιδιωτικό, υπολογισμένο)
5. Δημόσιο κλειδί είναι το (e, n) και ιδιωτικό κλειδί το (d, n) .

Κατά την κρυπτογράφηση το κείμενο κρυπτογραφείται σε μπλοκ, με κάθε μπλοκ M να έχει δυαδική τιμή μικρότερη από n και μέγεθος i bits, όπου $2^i < n < 2^{i+1}$, με την σχέση $C \equiv M^e \pmod{n}$ (Εικόνα 1). Ανάλογα η αποκρυπτογράφηση (decryption) ενός μπλοκ κρυπτογραφημένου κειμένου C επιτυγχάνεται με την σχέση $M \equiv C^d \pmod{n}$ (Εικόνα 1) (Stallings, 2011, pp. 277-285).

2.3 Προβλήματα κατά την δημιουργία κλειδιών RSA

Το μεγαλύτερο πρόβλημα στην δημιουργία των κλειδιών είναι η επιλογή των τυχαίων πρώτων αριθμών p , q . Όπως αναφέρθηκε και στην ενότητα [2.2](#) τα κλειδιά είναι μεγάλου μεγέθους, κυρίως 2048 bit, ενώ τα τελευταία χρόνια κλειδιά μεγέθους 4096 bit αρχίζουν και γίνονται πιο συνήθη. Η ασφάλεια του RSA βασίζεται στο γεγονός ότι τα p , q επιλέγονται τυχαία με μέγεθος το μισό του μεγέθους του κλειδιού, ώστε να έχουν περίπου την μισή εντροπία (entropy) του, και στο γεγονός ότι η επίλυση του προβλήματος παραγοντοποίησης (factorization problem) μεγάλων αριθμών είναι άγνωστο αν μπορεί να επιλυθεί σε πολυωνυμικό χρόνο με μή κβαντικούς υπολογιστές.

Η επίλυση του προβλήματος της παραγοντοποίησης έχει απασχολήσει αρκετά την ακαδημαϊκή κοινότητα με την εύρεση διαφόρων μεθόδων για την παραγοντοποίηση συγκεκριμένων ομάδων ακεραίων όπως η μέθοδος Pollard rho, η μέθοδος Pollard $p-1$, η μέθοδος συνεχών κλασμάτων (continued fraction), η μέθοδος ελλειπτικής καμπύλης³, το τετραγωνικό κόσκινο (quadratic sieve) και το γενικό κόσκινο πεδίου αριθμών⁴ (general number sieve).

Name	Decimals	Bits	Factoring announced
RSA-129	129	426	April 1994
RSA-130	130	430	April 1996
RSA-140	140	463	February 1999
RSA-150	150	496	2004
RSA-160	160	530	April 2003
RSA-170	170	563	December 2009
RSA-576	174	576	December 2003
RSA-180	180	596	May 2010
RSA-190	190	629	November 2010
RSA-640	193	640	November 2005
RSA-200	200	633	May 2005
RSA-210	210	696	September 2013
RSA-704	212	704	July 2012
RSA-220	220	729	May 2016
RSA-230	230	762	August 2018
RSA-232	232	768	February 2020
RSA-768	232	768	December 2009
RSA-240	240	795	November 2019
RSA-250	250	829	February 2020

Εικόνα 2. Παραγοντοποιήσεις για αριθμούς RSA (Buell, 2021, p.154)

³ Η αλλιώς και παραγοντοποίηση ελλειπτικής καμπύλης Lenstra. Αποτελεί εφαρμογή της προσέγγισης Pollard $p-1$ στην ομάδα σημείων μιας ελλειπτικής καμπύλης

⁴ Θεωρείται ο πιο αποτελεσματικός κλασικός αλγόριθμος παραγοντοποίησης ακεραίων μεγαλύτερων από 10^{100}

Στην Εικόνα 2 παρατηρούμε έναν κατάλογο με αριθμούς που δημοσίευσαν οι Rivest, Shamir και Adleman και τη χρονολογία που επιτεύχθηκε η παραγοντοποίησή τους. Σε πολλές περιπτώσεις, οι υπολογισμοί χρειάστηκαν χρόνια. Εάν οι πρώτοι αριθμοί δεν επιλεγούν σωστά, τότε θα έχουν χαμηλότερη εντροπία από την αναμενόμενη, με αποτέλεσμα ο χρόνος παραγοντοποίησης του modulo n να μειώνεται αισθητά και έτσι η ασφάλεια του αλγορίθμου RSA μπορεί να διακυβευτεί (Buell, 2021, pp. 157-178).

Φυσικά η επιλογή τόσο μεγάλων και τυχαίων πρώτων αριθμών δεν είναι εύκολη υπόθεση. Για αρχή θα πρέπει να προσδιορίσουμε ότι ο τυχαίος αριθμός που επιλέξαμε είναι πρώτος. Αυτό επιτυγχάνεται με διάφορες μεθόδους όπως η παραλλαγή του μικρού θεωρήματος Fermat (Εικόνα 3) και το κριτήριο πιστοποίησης πρώτων Miller-Rabin (Εικόνα 4)⁵.

Theorem 2.6 (Lucas' converse of Fermat's Little Theorem). *Let n be a positive integer. If $a^{n-1} \equiv 1 \pmod{n}$ and there is an integer a for every prime divisor p_i of $n-1$ satisfies $a^{(n-1)/p_i} \not\equiv 1 \pmod{n}$, then n is prime.*

Εικόνα 3. Παραλλαγή μικρού θεωρήματος Fermat (Wang, 2021)

Theorem 16.4. *Let p be an odd prime and $p-1 = 2^h m$, with m odd and $h \geq 1$. Then any integer a ($1 < a < p-1$) satisfies $a^m \equiv 1 \pmod{p}$ or $a^{2^j m} \equiv -1 \pmod{p}$ for some $j = 1, 2, \dots, h-1$.*

Εικόνα 4. Κριτήριο πιστοποίησης Miller-Rabin (Burton, 2011, p.369)

⁵ Τα αναφερθέντα κριτήρια, καθώς και άλλα όπως τα κριτήρια Solovay-Strassen, Baillie-PSW και AKS, αποτελούν πιθανολογικές δοκιμές για να προσδιοριστεί αν ένας αριθμός είναι σύνθετος ή πιθανός πρώτος

Έπειτα θέλουμε να μειώσουμε την χρονική πολυπλοκότητα (time complexity) της διαδικασίας εύρεσης τυχαίων πρώτων αριθμών. Αυτό επιτυγχάνεται με αλγορίθμους που δεν επιλέγουν τυχαία αριθμούς και ελέγχουν αν είναι πρώτοι, αλλά κατασκευάζουν πιθανούς πρώτους αριθμούς με συγκεκριμένα κριτήρια ώστε να περιορίζονται οι αριθμοί που επιλέγονται πριν τον έλεγχο. Ένας τέτοιος αλγόριθμος που θα μας χρησιμεύσει στα ερωτήματα της εργασίας είναι ο αλγόριθμος των Joye και Paillier (2006). Ο αλγόριθμος (Εικόνα 5) έχει ως στόχο την παραγωγή ασφαλών και σχεδόν ασφαλών (safe and quasi-safe) πρώτων αριθμών για τον αλγόριθμο RSA.

Parameters: $l = v\Pi$, $m = w\Pi$, $m' = m/2^r$,
 $t, a \in \text{QR}(m)$ and u as above

Output: a random prime $q \in [q_{\min}, q_{\max}]$ with $(q-1)/2$ prime

1. Randomly choose $\chi \in (\mathbb{Z}/m\mathbb{Z})^*$
2. Set $k \leftarrow 4u\chi^2 + 3m' \pmod{m}$
3. Set $q \leftarrow [(k - t) \pmod{m}] + t + l$
4. If $(\text{T}(q) = \text{false} \text{ or } \text{T}((q-1)/2) = \text{false})$ then
 - (a) Set $k \leftarrow ak \pmod{m}$
 - (b) Go to Step 3
5. Output q

Εικόνα 5. Αλγόριθμος των Joye και Paillier (Joye and Paillie, 2006)

3. Περιγραφή λύσεων

3.1 Ερώτημα Α

Για την επίλυση του ερωτήματος χρησιμοποιούνται οι συναρτήσεις `fast_gen_prime_1` και `fast_gen_prime_2` με κώδικα όπως φαίνεται στο [Παράρτημα Α](#). Με την εκτέλεση της συνάρτησης `GE3_A` έχουμε το αποτέλεσμα⁶ όπως φαίνεται στην Εικόνα 6.

```
D:\Downloads\PLH35_3>python GE3_Final.py
Original Code
Time to generate 100 primes: 3.859699249267578 seconds

Fast_gen_prime - scenario 1
Time to generate 100 primes: 2.1053333282470703 seconds
1.7543659210205078 seconds faster than Original Code
83.33 % faster than Original Code

Fast_gen_prime - scenario 2
Time to generate 100 primes: 1.3274500370025635 seconds
2.5322492122650146 seconds faster than Original Code
190.76 % faster than Original Code
```

Εικόνα 6. Αποτελέσματα συνάρτησης `GE3_A`

Παρατηρούμε πως η συνάρτηση στο σενάριο 1 ήταν 83.33% πιο γρήγορη στον υπολογισμό 100 πρώτων από ότι η συνάρτηση του αρχικού κώδικα, ενώ η συνάρτηση στο σενάριο 2 ήταν 190.76% πιο γρήγορη στον ίδιο υπολογισμό από τον αρχικό κώδικα. Το γεγονός αυτό οφείλεται στην παράμετρο με την οποία πολλαπλασιάζουμε τον τυχαίο αριθμό r . Ο ζητούμενος αριθμός p , μεγέθους 256 bit, παράγεται από τον πολλαπλασιασμό μιας σταθεράς, έστω x , με τον τυχαίο αριθμό r συν 1. Όσο μεγαλώνει η σταθερά x , τόσο λιγότερα bit χρειάζεται ο τυχαίος αριθμός r και επομένως τόσο πιο εύκολο είναι υπολογιστικά να παραχθούν αυτοί οι τυχαίοι αριθμοί ώστε να πληρούν τα απαραίτητα κριτήρια.

Εκτελώντας την συνάρτηση `bit_calc` με τον κώδικα του [Παράρτηματος Α](#), έχουμε μια λίστα των bit που χρειάζονται για το r με x να παίρνει τιμές από 1 έως 19 όπως φαίνεται στην Εικόνα 7.

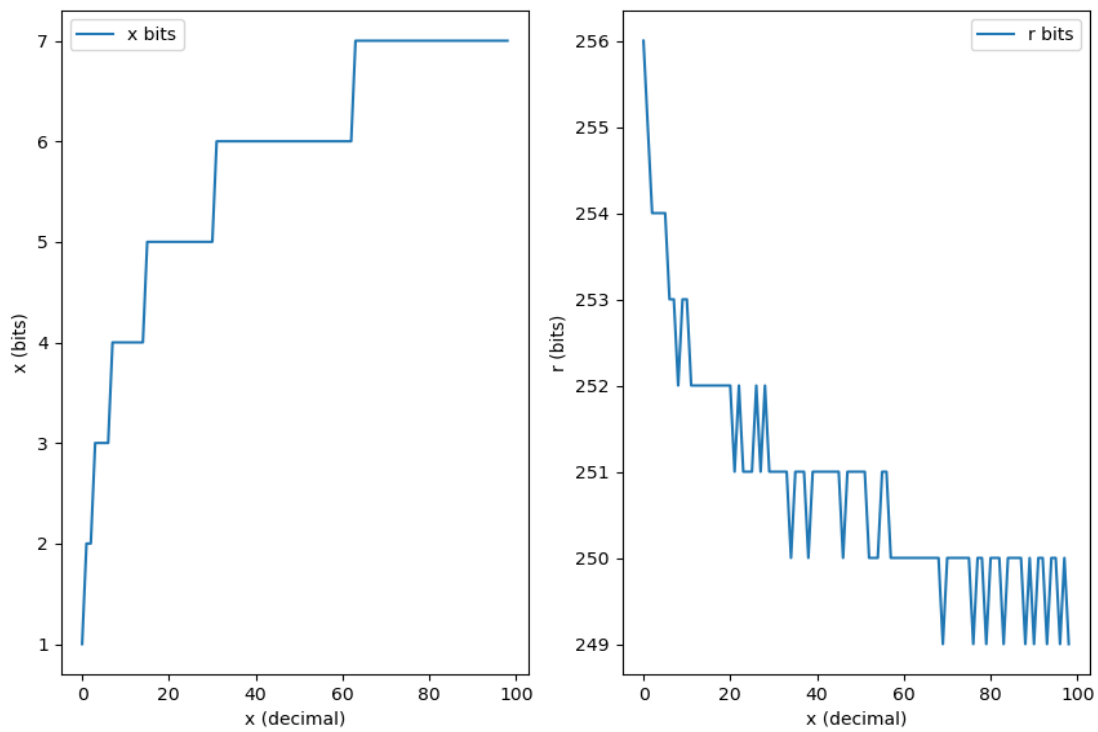
⁶ Τα χρονικά αποτελέσματα ποικίλουν για κάθε εκτέλεση του αλγορίθμου, με τα συγκεκριμένα να αποτελούν αντιπροσωπευτικό δείγμα

Παρατηρούμε επομένως μια αντιστρόφως ανάλογη σχέση των bit της μεταβλητής x και των bit της μεταβλητής r .

```
D:\Downloads\PLH35_3\Code>python bit_calc.py
x = 1 Bits ---> r: 256 -- x: 1 -- p: 256
x = 2 Bits ---> r: 255 -- x: 2 -- p: 256
x = 3 Bits ---> r: 254 -- x: 2 -- p: 256
x = 4 Bits ---> r: 254 -- x: 3 -- p: 256
x = 5 Bits ---> r: 254 -- x: 3 -- p: 256
x = 6 Bits ---> r: 254 -- x: 3 -- p: 256
x = 7 Bits ---> r: 253 -- x: 3 -- p: 256
x = 8 Bits ---> r: 253 -- x: 4 -- p: 256
x = 9 Bits ---> r: 253 -- x: 4 -- p: 256
x = 10 Bits ---> r: 253 -- x: 4 -- p: 256
x = 11 Bits ---> r: 253 -- x: 4 -- p: 256
x = 12 Bits ---> r: 252 -- x: 4 -- p: 256
x = 13 Bits ---> r: 252 -- x: 4 -- p: 256
x = 14 Bits ---> r: 253 -- x: 4 -- p: 256
x = 15 Bits ---> r: 252 -- x: 4 -- p: 256
x = 16 Bits ---> r: 252 -- x: 5 -- p: 256
x = 17 Bits ---> r: 252 -- x: 5 -- p: 256
x = 18 Bits ---> r: 252 -- x: 5 -- p: 256
x = 19 Bits ---> r: 252 -- x: 5 -- p: 256
```

Εικόνα 7. Απαιτούμενα bit για x, r, p

Στην Εικόνα 8 παρατηρούμε επίσης ένα διάγραμμα με τιμές του x στο διάστημα $[1,99]$ και την διαφορά στα bit αναπαράστασης σε σχέση με αυτών της μεταβλητής r .

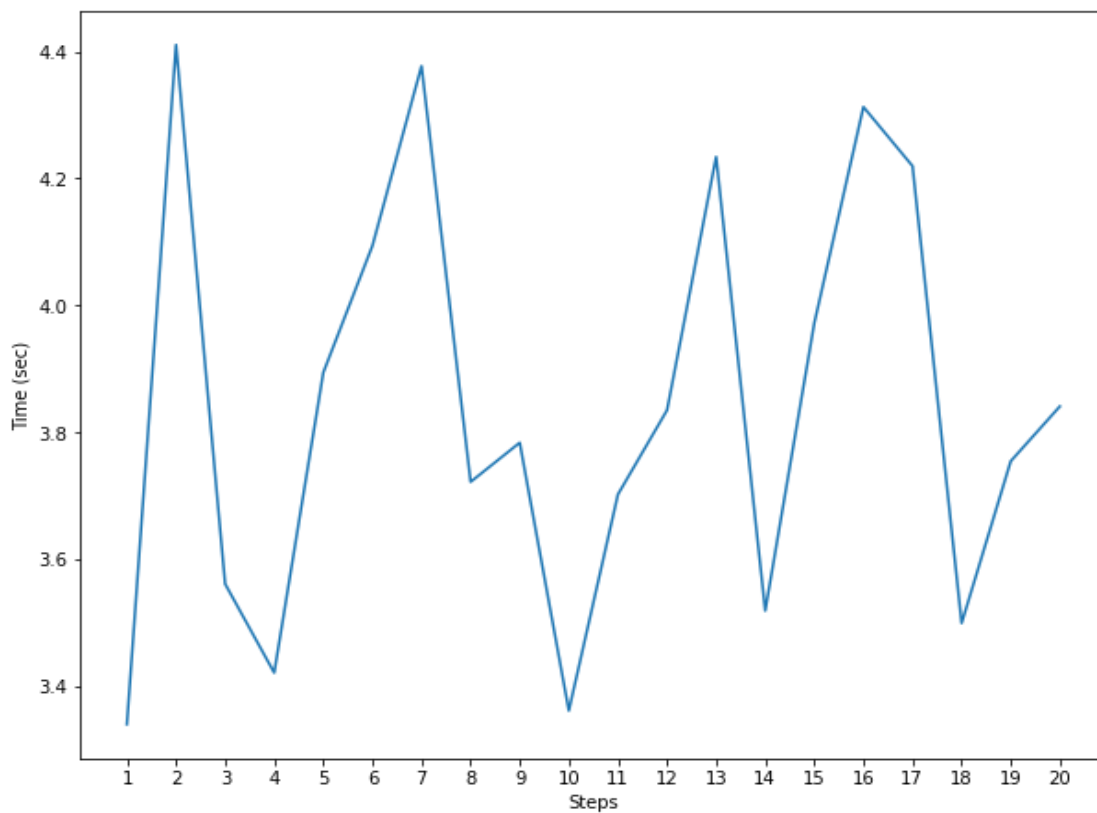


Εικόνα 8. Διάγραμμα διαφορών σε bit $x-r$

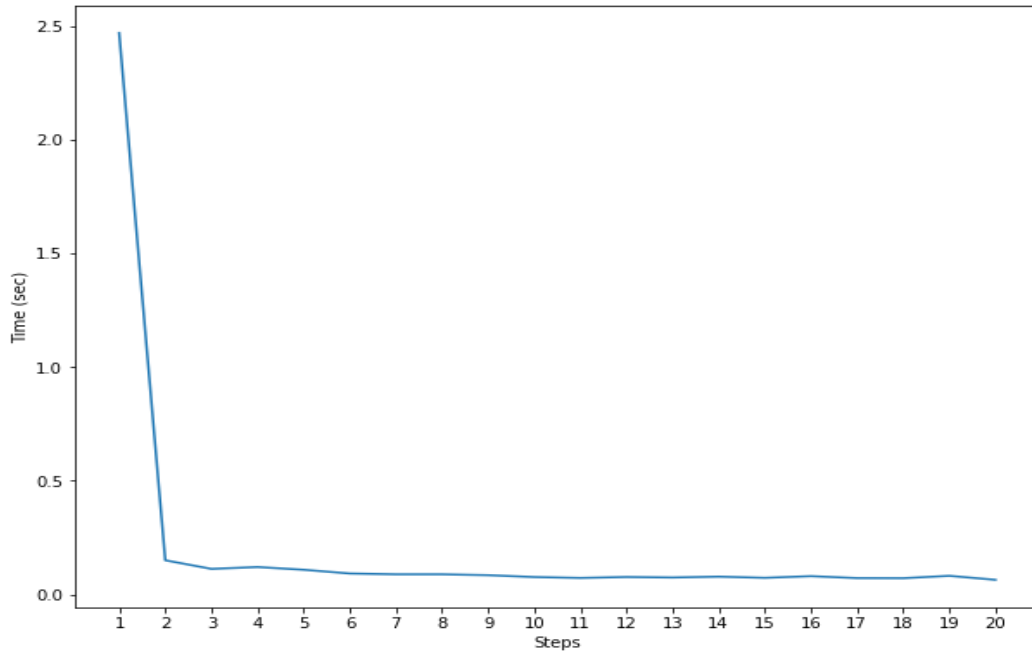
3.2 Ερώτημα Β

Για την επίλυση του ερωτήματος χρησιμοποιείται η συνάρτηση `fast_gen_prime` με κώδικα όπως φαίνεται στο [Παράρτημα Α](#). Για να γίνει μια σύγκριση με τον αρχικό κώδικα θα πρέπει να παράγονται πρώτοι αριθμοί με τα ίδια βήματα. Από τον κώδικα που μας δίνεται στο ερώτημα Β, έχουμε ότι η συνάρτηση `fast_gen_prime` παράγει 100 πρώτους αριθμούς των 256 bit για κάθε τιμή του M σε ένα βρόγχο των 20 συνολικών βημάτων. Επομένως προσαρμόζουμε ανάλογα την παραγωγή των πρώτων αριθμών του αρχικού κώδικα για την σύγκριση. Η διαδικασία φαίνεται στον κώδικα της συνάρτησης `GE3_B` στο [Παράρτημα Α](#). Μια επιμέρους διαδικασία είναι η προσθήκη όλων των χρόνων για κάθε συνάρτηση σε λίστες και η μεταφορά τους σε αρχεία csv. Με τα δεδομένα των αρχείων csv μπορούμε να κάνουμε μια συγκριτική ανάλυση σε Jupyter Notebook. Ακολουθούν τα αποτελέσματα της ανάλυσης. Για τον πλήρη κώδικα ο αναγνώστης παραπέμπεται στο [Παράρτημα Β](#).

Ας δούμε αρχικά τα γραφήματα των χρόνων για κάθε συνάρτηση (Εικόνα 9, Εικόνα 10).

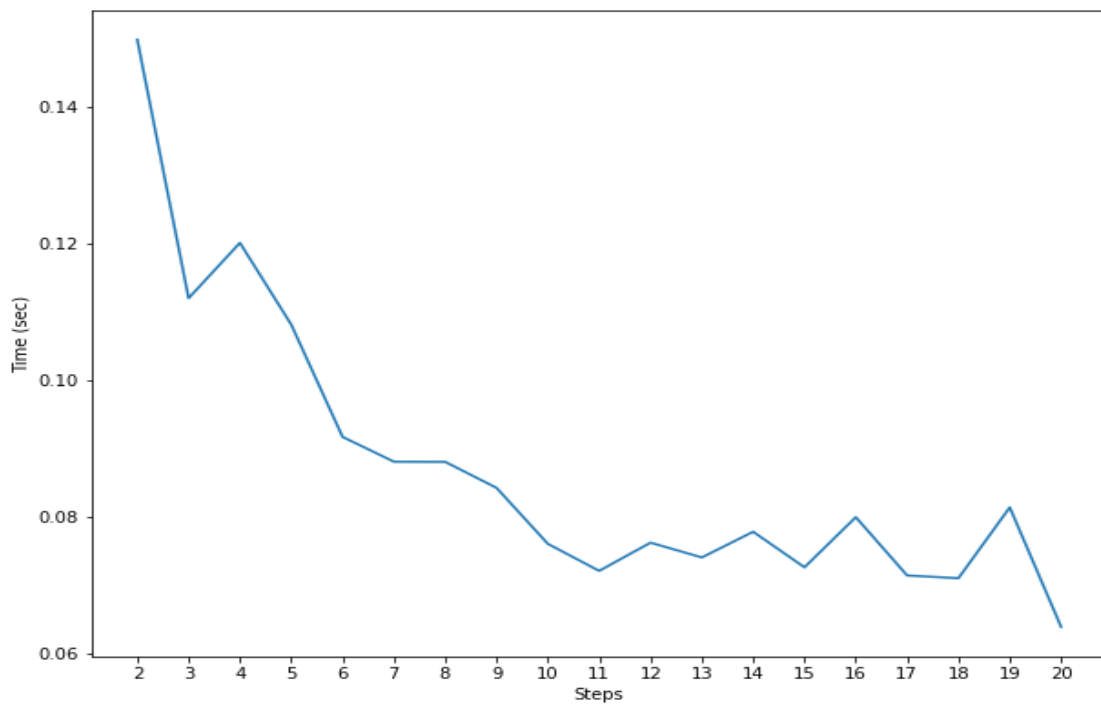


Εικόνα 9. Χρόνοι ανά βήμα αρχικού κώδικα



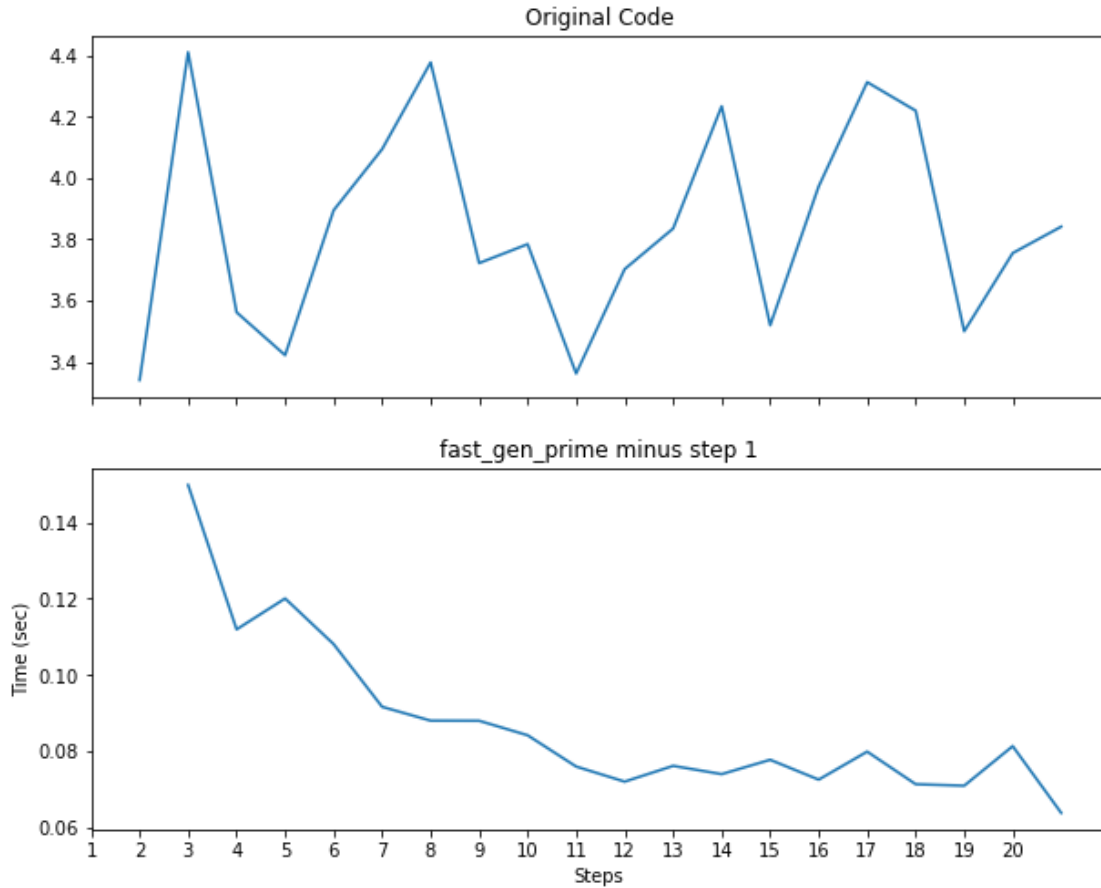
Εικόνα 10. Χρόνοι ανά βήμα συνάρτησης `fast_gen_prime`

Παρατηρούμε επίσης ότι για την περίπτωση της συνάρτησης `fast_gen_prime` δεν έχουμε καθαρή εικόνα των βημάτων 2 έως 20 του παραπάνω γραφήματος. Εξαιρώντας το βήμα 1 έχουμε το γράφημα που φαίνεται στην Εικόνα 11.



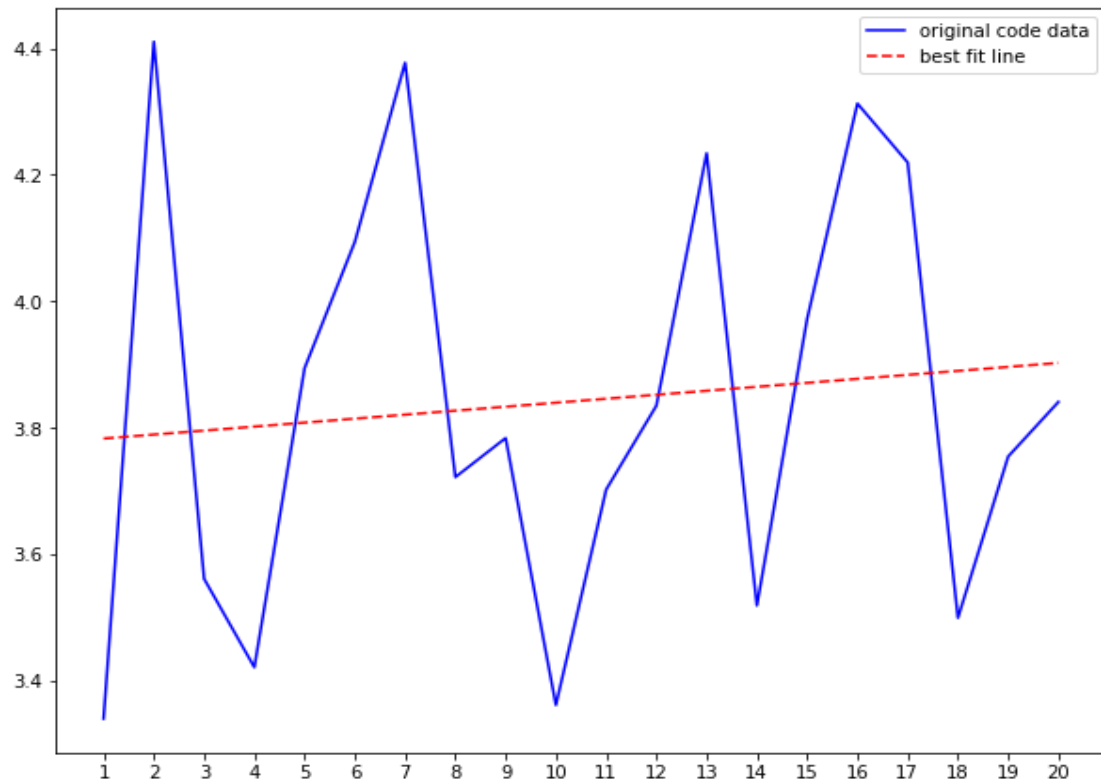
Εικόνα 11. Χρόνοι βημάτων 2-20 συνάρτησης `fast_gen_prime`

Γίνεται αντιληπτό πως υπάρχει μεγάλη διαφορά στους χρόνους των δύο συναρτήσεων όπως παρατηρούμε καθαρά και στο παράλληλο γράφημα (Εικόνα 12), κάτι που ήταν αναμενόμενο καθώς η περίπτωση του ερωτήματος B είναι ουσιαστικά μια διεύρυνση του ερωτήματος A.

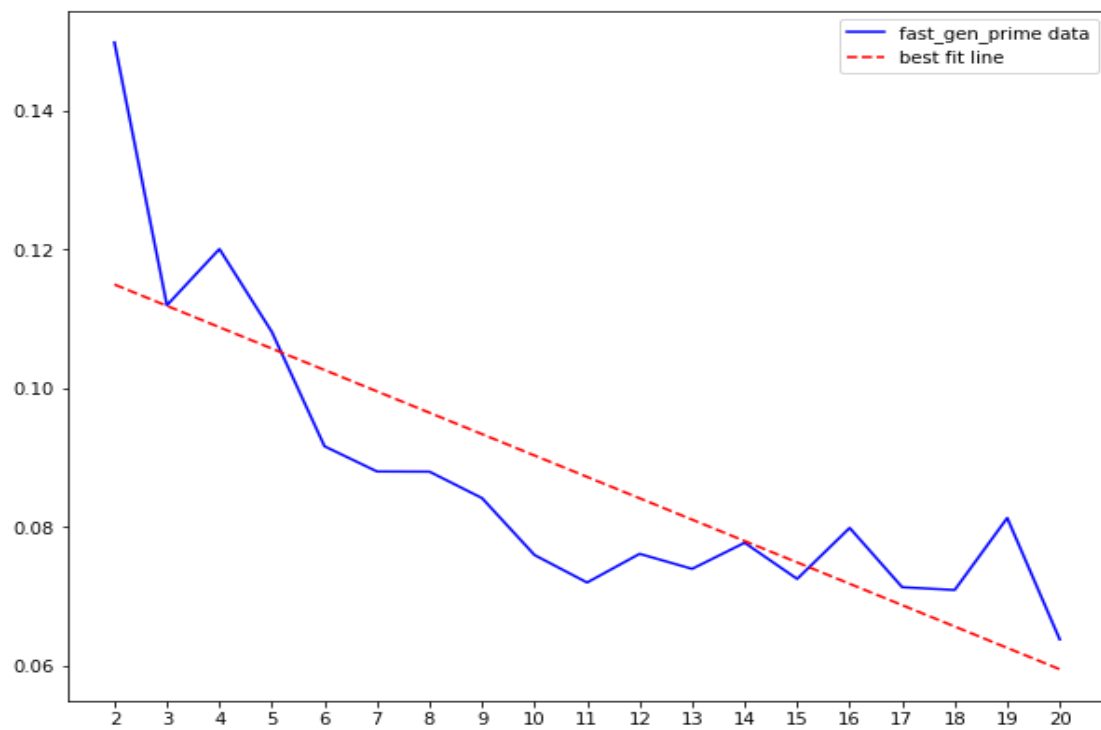


Εικόνα 12. Παράλληλο γράφημα χρόνων ανα βήμα

Κατασκευάζοντας τα γραφήματα συναρτήσεων καλύτερης προσαρμογής για κάθε συνάρτηση μπορούμε να αποφανθούμε για την τάση των χρόνων των δύο συναρτήσεων. Παρατηρούμε ότι η συνάρτηση προσαρμογής του αρχικού κώδικα έχει μια ελαφρά ανοδική κλίση (Εικόνα 13), ενώ για την συνάρτηση προσαρμογής της `fast_gen_prime` παρατηρούμε καθαρή καθοδική κλίση (Εικόνα 14). Όπως αναφέρθηκε και στο ερώτημα A, καθώς ο αριθμός p είναι σταθερού μεγέθους (256 bit) και ο M είναι σταθερός για κάθε βήμα, ο συνολικός χρόνος εύρεσης είναι συνάρτηση του χρόνου εύρεσης τυχαίου αριθμού r . Με τον M να αυξάνεται ανά βήμα, είναι ευκολότερο υπολογιστικά να βρεθεί ο αριθμός r . Επομένως το μέγεθος της μεταβλητής r κάθε βήματος και συνεπώς του χρόνου εύρεσής της (και κατ' επέκταση της συνάρτησης) είναι αντιστρόφως ανάλογο με το μέγεθος της σταθεράς M .



Εικόνα 13. Συνάρτηση καλύτερης προσαρμογής αρχικού κώδικα



Εικόνα 14. Συνάρτηση καλύτερης προσαρμογής fast_gen_prime

3.3 Ερώτημα Γ

Η παραγωγή των κλειδιών ψηφιακών ταυτοτήτων της Εσθονίας πραγματοποιήθηκε από την εταιρία Infineon, η οποία για την κατασκευή των πρώτων αριθμών κλειδιών RSA χρησιμοποίησε μια παραλλαγή του αλγορίθμου των Joye και Paillier. Πιο συγκεκριμένα οι παραγόμενοι πρώτοι αριθμοί είχαν την πολυωνυμική μορφή :

$$p = k \cdot M + (65537^a \bmod M)$$

$$k, a \text{ τυχαίοι ακέραιοι, } M = \prod_{i=1}^r P_i, P_i = 2 \cdot 3 \cdot \dots \cdot P_n, P_n \text{ n-πρώτος}$$

Η κύρια διαφορά με τον αρχικό αλγόριθμο είναι πως η τυχαία μεταβλητή k των Joye και Paillier αντικαθίσταται με την τιμή $65537^a \bmod M$, όπου a μικρός τυχαίος αριθμός. Αυτό έχει ως συνέπεια την μείωση της εντροπίας του παραγόμενου πρώτου.

Οι αναλυτές που ανακάλυψαν την συγκεκριμένη ευπάθεια (vulnerability) με όνομα ROCA⁷, εκμεταλλεύτηκαν την συγκεκριμένη πολυωνυμική μορφή για την παραγοντοποίηση των κλειδιών χρησιμοποιώντας την επέκταση Howgrave-Graham (1998) της επίθεσης Coppersmith (1996), ενώ τα κλειδιά με την συγκεκριμένη ευπάθεια είναι εύκολα αναγνωρίσιμα με αποτύπωμα (fingerprint) βασισμένο στην ύπαρξη διακριτού λογαρίθμου $\log_{65537} N \bmod M$ (Nemec et al., 2017).

.

⁷ Return of the Coppersmith's Attack (CVE-2017-15361)

3.4 Ερώτημα Δ

3.4.1 Υποερώτημα 1

Όπως αναφέρθηκε στην ενότητα [2.3](#) μια από τις δικλίδες ασφάλειας του RSA είναι η πιθανώς μη πολυωνυμικού χρόνου επίλυση του προβλήματος παραγοντοποίησης μεγάλων αριθμών. Καθώς η προσπάθεια παραγοντοποίησης με τις μεθόδους που προαναφέρθηκαν μπορεί να πάρει υπερβολικά μεγάλο χρόνο, υπάρχει ένας τρόπος να ελέγξουμε αν υπάρχουν παράγοντες ενός αριθμού ελέγχοντας βάσεις δεδομένων γνωστών παραγοντοποιήσεων αριθμών όπως η FactorDB⁸. Ελέγχοντας όλα τα δοθέντα moduli N έχουμε τα ακόλουθα δύο⁹ αποτελέσματα όπως φαίνονται στην Εικόνα 15 και Εικόνα 16.

Result:		
status (?)	digits	number
FF	617 (show)	2026303824...27 _{<617>} = 1409689913...13 _{<309>} · 1437411026...79 _{<309>}

Εικόνα 15. Παράγοντες modulus υπ' αριθμόν 7

Result:		
status (?)	digits	number
FF	617 (show)	1625841916...03 _{<617>} = 9697194558...99 _{<308>} · 1676610597...97 _{<309>}

Εικόνα 16. Παράγοντες modulus υπ' αριθμόν 10

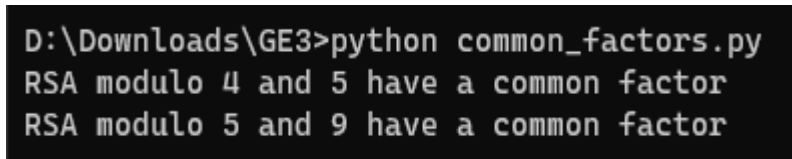
Παρατηρούμε πως υπάρχουν δύο πρώτοι παράγοντες για κάθε modulus με συνέπεια να έχουμε βρεί τα p, q του αλγορίθμου RSA. Ακολουθώντας τα βήματα που περιγράφονται στην ενότητα [2.2](#) μπορούμε να υπολογίσουμε τα d σε πολυωνυμικό χρόνο και επομένως η ασφάλεια του αλγορίθμου έχει παραβιαστεί.

⁸ <http://factordb.com/>

⁹ Για τα υπόλοιπα δεν υπάρχουν καταχωρήσεις γνωστών παραγόντων

3.4.2 Υποερώτημα 2

Αν και η παραγοντοποίηση είναι ένα πολύ δύσκολο πρόβλημα, υπάρχει ένα διαφορετικό πρόβλημα που είναι πολύ πιο εύκολο, η εύρεση του μεγαλύτερου κοινού διαιρέτη δύο αριθμών (greater common divisor – “gcd”). Γνωρίζουμε ότι $\gcd(x, y)$ είναι πάντα 1 αν τα x και y είναι πρώτοι αριθμοί ή είναι πρώτοι μεταξύ τους. Στην περίπτωση των δοθέντων moduli N έχουμε ότι κάθε modulus N_i , $i \in \{1, 2, 3, \dots, 10\}$ είναι της μορφής $N_i = p_i \cdot q_i$, όπου p_i, q_i , με $p_i \neq q_i$, διαφορετικοί πρώτοι $\forall i$. Με εκτέλεση του δοθέντα από την εκφώνηση κώδικα έχουμε το αποτέλεσμα όπως φαίνεται στην Εικόνα 17.



```
D:\Downloads\GE3>python common_factors.py
RSA modulo 4 and 5 have a common factor
RSA modulo 5 and 9 have a common factor
```

Εικόνα 17. Αλγόριθμος κοινών παραγόντων

Ας εξηγήσουμε τι σημαίνει αυτό το αποτέλεσμα παίρνοντας την περίπτωση των modulo 4 και 5 του αποτελέσματος. Τα N_4 και N_5 έχουν κοινό παράγοντα έστω k . Όμως έχουμε κατασκευάσει τα N_i με τέτοιο τρόπο ώστε να είναι γινόμενα δύο διαφορετικών πρώτων αριθμών, που σημαίνει πως $k = p_4 = p_5$ και $N_4 / k = q_4$, $N_5 / k = q_5$ ή $k = q_4 = q_5$ και $N_4 / k = p_4$, $N_5 / k = p_5$. Σε κάθε περίπτωση έχουμε βρεί τα p, q των modulo 4 και 5. Ακολουθώντας τα βήματα που περιγράφονται στην ενότητα [2.2](#) μπορούμε να υπολογίσουμε τα d_4 και d_5 σε πολυωνυμικό χρόνο και επομένως η ασφάλεια του αλγορίθμου έχει παραβιαστεί. Η ίδια διαδικασία ακολουθείται και για τα modulo 5 και 9.

3.4.3 Υποερώτημα 3

Αναφέραμε στην ενότητα [3.3](#) ότι τα κλειδιά που έχουν την ευπάθεια ROCA περιέχουν ένα αποτύπωμα που μπορεί να επαληθευτεί γρήγορα. Από την εκφώνηση μας δίνεται τέτοιος αλγόριθμος ελέγχου της σχετικής ευπάθειας (Centre for Research on Cryptography and Security, 2017). Με την εκτέλεση του αλγορίθμου έχουμε τα αποτελέσματα στην Εικόνα 18.

```
D:\Downloads\GE3>roca-detect ./GE3_keys.txt
2022-01-18 13:08:47 [10620] WARNING Fingerprint found in modulus ./GE3_keys.txt idx 0
{"type": "mod-dec", "fname": "./GE3_keys.txt", "idx": 0, "aux": null, "n": "0x9f66a23b392abaf1a55bf371a5f17066eeb308a53d
b93083112152dae02a1d9b86c33cec1f6baa0712a5985e95ac2f01bf56fb5104021fbaa3b1ee992cf0e671d06bd0902e97c186ca00d96ce932738e3c
95d815b1be1f9c2482488457b1876ec155844bddd548e3c9216787b68aa1777e16561c9dbc9e7c3d761b119adf6c8d64fcd4d91c4c1440e34b2a4dc18
9c3024b390f0d679d864c93693c450c9bd2fa84b3311alacd42f6cal238774f593b4789b9782d6ff4c429b72bc6e51240fab2df0e21d117aa7e5beb7
91625f78de205e753b56acca8c33b8670eccaf2b7642449de3f71560c3f607db9a2879cb18bf40d0d5021446969d5cf674e3222cd0be5b", "marked
": true, "time_years": 85.25100750352632, "price_aws_c4": 37365.516588795595}
2022-01-18 13:08:47 [10620] WARNING Fingerprint found in modulus ./GE3_keys.txt idx 2
{"type": "mod-dec", "fname": "./GE3_keys.txt", "idx": 2, "aux": null, "n": "0xe147c8bea667cc200adaf6b07ab186d9d9af6c89ad
b1bccd9b9581ebd59d8e1a4f1f069149eb3d59a1ab55d135c1416a930ad59e5e98e5305dfa8002b320c1922a35e66cfbe90fc1b2b4c55ee7ca85e724
db26d5749327a0ba3d47bfa171d6a7f10844ed4ac7b4581027dcf363dac94b887412412a05017b0ad1bef8bf412ee3bca4ae5d16f65b14be33aabda1
648c2084e0cb2bc5b063a8748ce66832b3a9714780a3750ca66f0152962100030b065042c70d7a3e4ed1bc51261410735fe8558c58df5d115d87e80c
19d27b68baaaff83389c5f5c5d5df5e9cd3f3df447dbdfcc417afa34d8dd39353d5e37997dd2a63dd005457144b000178d65a6c1562927", "marked
": true, "time_years": 85.25100750352632, "price_aws_c4": 37365.516588795595}
2022-01-18 13:08:47 [10620] WARNING Fingerprint found in modulus ./GE3_keys.txt idx 7
{"type": "mod-dec", "fname": "./GE3_keys.txt", "idx": 7, "aux": null, "n": "0xb712de00aa3ef987e7f73075b553b3bb3997fc944e
e0341f95c7de97bb887cb3ac7e767b83e5abebf06aa01632ec9e819d0481047c2d8b971fc6db9bc9f02cfeb9f37c127b62e172aed8231eb862d20096
8490cbb6bf83ae612aeebcb509cfce907586e84351ddeaf8524515ef4494a00b56baf74fd3e6b0ef9abee104f627c4657a19ee9e81da3f51a067ef
1317944161fb4fec64289f8c415e12efe420fd759a1112b41b40d3d50560b989df51db4d36f7378155d4aac30cf3baf9f95836c51ef911fdddf63aef0
cefc080f222e62974965a81430dff8ad60f02f56f642e5463f38638442891af398f234363e0b87c5cd2afe5563ab2c6188ac2609033fb", "marked
": true, "time_years": 85.25100750352632, "price_aws_c4": 37365.516588795595}
2022-01-18 13:08:47 [10620] INFO ### SUMMARY #####
2022-01-18 13:08:47 [10620] INFO Records tested: 20
2022-01-18 13:08:47 [10620] INFO .. PEM certs: . . . 0
2022-01-18 13:08:47 [10620] INFO .. DER certs: . . . 0
2022-01-18 13:08:47 [10620] INFO .. RSA key files: . . . 0
2022-01-18 13:08:47 [10620] INFO .. PGP master keys: 0
2022-01-18 13:08:47 [10620] INFO .. PGP total keys: 0
2022-01-18 13:08:47 [10620] INFO .. SSH keys: . . . 0
2022-01-18 13:08:47 [10620] INFO .. APK keys: . . . 0
2022-01-18 13:08:47 [10620] INFO .. JSON keys: . . . 0
2022-01-18 13:08:47 [10620] INFO .. LDIF certs: . . . 0
2022-01-18 13:08:47 [10620] INFO .. JKS certs: . . . 0
2022-01-18 13:08:47 [10620] INFO .. PKCS7: . . . . 0
2022-01-18 13:08:47 [10620] INFO Fingerprinted keys found: 3
2022-01-18 13:08:47 [10620] INFO WARNING: Potential vulnerability
2022-01-18 13:08:47 [10620] INFO #####
```

Εικόνα 18. Αποτελέσματα εύρεσης αποτυπωμάτων ευπάθειας ROCA

Μπορούμε να αποφανθούμε πως τα modulus υπ' αριθμόν 1, 3 και 8 είναι ευάλωτα στην ευπάθεια ROCA.

4. Συμπεράσματα

Στην παρούσα εργασία ασχοληθήκαμε με προβλήματα που εμφανίζονται κατά τη διάρκεια μιας προβληματικής διαδικασίας δημιουργίας κλειδιού του αλγορίθμου κρυπτογράφησης RSA. Είδαμε πως η τροποποίηση ενός αλγορίθμου δημιουργίας πρώτων αριθμών οδήγησε στη μείωση της εντροπίας του παραγόμενου RSA modulus. Αυτό δείχνει ότι η σωστή δημιουργία κλειδιών και πιο συγκεκριμένα η δημιουργία μεγάλων πρώτων αριθμών με υψηλή εντροπία είναι υψίστης σημασίας για την ασφάλεια του RSA. Προκειμένου να δημιουργηθούν ανθεκτικά και αξιόπιστα κλειδιά, καθώς η τεχνολογία εξελίσσεται και οι δυνατότητες των σύγχρονων υπολογιστών αυξάνονται, η διαδικασία πρέπει να εξετάζεται διεξοδικά και να ελέγχεται ώστε να αποφεύγονται ενδεχόμενες ευπάθειες.

Βιβλιογραφία

Aumasson, J.P. (2018). *Serious Cryptography: A Practical Introduction to Modern Encryption*. San Francisco: No Starch Press, pp.55–78, 93–98, 261–299.

Buell, D.A. (2021). *Fundamentals of Cryptography: Introducing Mathematical and Algorithmic foundations*. Cham, Switzerland: Springer, pp.3–9, 27–45, 54–56, 149–178.

Burton, D.M. (2011). *Elementary number theory*. New York; Boston: Mcgraw-Hill, pp.85–101, 129–145, 197–218.

Centre for Research on Cryptography and Security (2017). *ROCA detection tool*. [online] GitHub. Available at: <https://github.com/crocs-muni/roca> [Accessed 8 Mar. 2022].

Coppersmith, D. (1996). Finding a Small Root of a Univariate Modular Equation. *Advances in Cryptology — EUROCRYPT '96*, pp.155–165.

Coutinho, S.C. (2019). *The Mathematics of Ciphers: Number Theory and RSA Cryptography*. Boca Raton Crc Press.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, [online] 22(6), pp.644–654. Available at: <http://www.cs.jhu.edu/~rubin/courses/sp03/papers/diffie.hellman.pdf> [Accessed 2 Mar. 2022].

Gardner, M. (1977). MATHEMATICAL GAMES. *Scientific American*, [online] 237(2), pp.120–125. Available at: <https://www.jstor.org/stable/24954008?refreqid=excelsior%3Aa4f11a22f23eaa4e0cf6de517dfc5b26> [Accessed 3 Mar. 2022].

Godfrey Harold Hardy and Edward Maitland Wright (1975). *An introduction to the theory of numbers*. Oxford: Clarendon Press.

Hinek, M.J. (2010). *Cryptanalysis of RSA and its variants*. Boca Raton: Crc Press, pp.3–46, 51–134, 201–219.

Howgrave-Graham, N. (1998). *Computational Mathematics Inspired by RSA* submitted by. [online] Available at: <https://cr.yp.to/bib/1998/howgrave-graham.pdf> [Accessed 7 Mar. 2022].

Joye, M. and Paillier, P. (2006). *Fast Generation of Prime Numbers on Portable Devices: An Update.* [online] citeseerx.ist.psu.edu. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.700.6274&rep=rep1&type=pdf> [Accessed 27 Feb. 2022].

Joye, M., Paillier, P. and Vaudenay, S. (2000). Efficient Generation of Prime Numbers. *Cryptographic Hardware and Embedded Systems - CHES 2000.* [online] Available at: <https://infoscience.epfl.ch/record/99404?ln=en> [Accessed 6 Mar. 2022].

Kilgallin, J. and Vasko, R. (2019). *Factoring RSA Keys in the IoT Era.* [online] IEEE Xplore. Available at: <https://ieeexplore.ieee.org/document/9014350> [Accessed 26 Feb. 2022].

Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T. and Wachter, C. (2012). Ron was wrong, Whit is right. *IACR Cryptol. ePrint Arch.*, [online] 2012. Available at: <https://www.semanticscholar.org/paper/Ron-was-wrong%2C-Whit-is-right-Lenstra-Hughes/1cda554272e08240212e0a5848825d2d6edc3dd8> [Accessed 6 Mar. 2022].

Lenstra, A.K., Lenstra, H.W. and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen*, [online] 261(4), pp.515–534. Available at: https://www.researchgate.net/publication/50863306_Factoring_Polynomials_with_Rational_Coefficients [Accessed 6 Mar. 2022].

Merkle, R.C. (1978). Secure communications over insecure channels. *Communications of the ACM*, 21(4), pp.294–299.

Nemec, M., Sys, M., Svenda, P., Klinec, D. and Matyas, V. (2017). The Return of Coppersmith's Attack. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.*

Paar, C. and Pelzl, J. (2010). *Understanding Cryptography, A Textbook for Students and Practitioners.* Berlin, Heidelberg Springer Berlin Heidelberg, pp.149–200.

Pagourtzis, A. and Zachos, E. (2015). COMPUTATIONAL CRYPTOGRAPHY. *Kallipos.gr*, [online] pp.44–48, 82–105, 183–201. Available at: <https://repository.kallipos.gr/handle/11419/5439> [Accessed 26 Feb. 2022].

Rabin, M.O. (1980). Probabilistic algorithm for testing primality. *Journal of Number Theory*, [online] 12(1), pp.128–138. Available at: <https://www.sciencedirect.com/science/article/pii/0022314X80900840> [Accessed 4 Mar. 2022].

Rassias, I. (1999). *Number Theory*. Athens: Simmetria, pp.211–218.

Republic of Estonia, Information System Authority (2018). *ROCA Vulnerability and eID: Lessons Learned*. [online] Available at: <https://www.ria.ee/sites/default/files/content-editors/kuberturve/roca-vulnerability-and-eid-lessons-learned.pdf> [Accessed 26 Feb. 2022].

Rivest, R.L., Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, [online] 21(2), pp.120–126. Available at: <http://people.csail.mit.edu/rivest/Rsapaper.pdf> [Accessed 1 Mar. 2022].

Stallings, W. (2011). *Cryptography and Network Security: Principles and Practice*. Boston; Montreal: Prentice Hall, pp.243–291.

Sweigart, A. (2013). *Hacking secret ciphers with Python*. North Charleston, South Carolina: Createspace, pp.196–211, 361–422.

Wang, Z. (2021). *Methods of Primality Testing*. [online] Available at: https://www.researchgate.net/publication/348899360_Methods_of_Primality_Testing [Accessed 8 Mar. 2022].

Wong, D. (2015). *Lattice based attacks on RSA*. [online] GitHub. Available at: <https://github.com/mimoo/RSA-and-LLL-attacks> [Accessed 8 Mar. 2022].

Παράρτημα Α

```
import random
from time import time
import gmpy2
import pandas as pd

def isprime(x):
    for i in range(20):
        a = random.randrange(x)
        if pow(a, x, x) != a:
            return False
    return True

def gcd(x, y):
    while(y):
        x, y = y, x % y
    return x

def gen_prime(BITS):
    ub = 2**BITS
    lb = ub // 2
    p = random.randrange(lb, ub)
    while not isprime(p):
        p = random.randrange(lb, ub)
    return p

def fast_gen_prime_1(BITS):
    ub = 2**BITS
    ub = ub / 2
    lb = ub // 2
    r = random.randrange(lb, ub)
    p = 2*r + 1
    while not isprime(p):
        r = random.randrange(lb, ub)
        p = 2*r + 1
    return p

def fast_gen_prime_2(BITS):
```



```

ub = 2**BITS
ub = ub / 6
lb = ub // 2
r = random.randrange(lb, ub)
p = 6*r + 1
while not isprime(p):
    r = random.randrange(lb, ub)
    p = 6*r + 1
return p

primelst = [2]
while primelst[-1] < 166:
    primelst.append(gmpy2.next_prime(primelst[-1]))

M = 1

def fast_gen_prime(BITS):
    ub = 2**BITS
    ub = ub / M
    lb = ub // 2
    r = random.randrange(lb, ub)
    p = M*r + 1
    while not isprime(p):
        r = random.randrange(lb, ub)
        p = M*r + 1
    return p

def GE3_A():
    print("Original Code")
    ts = time()
    for i in range(100):
        p = gen_prime(256)
    tm = time() - ts
    print("Time to generate 100 primes:", tm, "seconds")

    print("\nFast_gen_prime - scenario 1")
    ts1 = time()
    for i in range(100):
        p1 = fast_gen_prime_1(256)
    tm1 = time() - ts1
    print("Time to generate 100 primes:", tm1, "seconds")
    if tm - tm1 > 0:

```

```

        print(tm - tm1, "seconds faster than Original Code")
        print("{:.2f}".format(
            (tm-tm1)/tm1*100), "%", "faster than Original Code")
    elif tm - tm1 < 0:
        print(tm1 - tm, "seconds slower than Original Code")
        print("{:.2f}".format(
            (tm1-tm)/tm1*100), "%", "slower than Original Code")
    else:
        print("Same speed calculation as Original Code")

print("\nFast_gen_prime - scenario 2")
ts2 = time()
for i in range(100):
    p2 = fast_gen_prime_2(256)
tm2 = time()-ts2
print("Time to generate 100 primes:", tm2, "seconds")
if tm - tm2 > 0:
    print(tm - tm2, "seconds faster than Original Code")
    print("{:.2f}".format(
        (tm-tm2)/tm2*100), "%", "faster than Original Code")
elif tm - tm2 < 0:
    print(tm2 - tm, "seconds slower than Original Code")
    print("{:.2f}".format(
        (tm2-tm)/tm2*100), "%", "slower than Original Code")
else:
    print("Same speed calculation as Original Code")

def GE3_B():
    original = []
    for i in range(20):
        ts_1 = time()
        for j in range(100):
            p = gen_prime(256)
            original.append(time() - ts_1)

    list = []
    for i in range(20):
        M *= primelst[i]
        ts = time()
        for j in range(100):
            p = fast_gen_prime(256)
            list.append(time() - ts)

    dict_1 = {'steps': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,

```

```

        12, 13, 14, 15, 16, 17, 18, 19, 20], 'time': original}
dict_2 = {'steps': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                    12, 13, 14, 15, 16, 17, 18, 19, 20], 'time': list}
df_1 = pd.DataFrame(dict_1)
df_2 = pd.DataFrame(dict_2)
df_1.to_csv('original.csv')
df_2.to_csv('fast_gen_prime.csv')

# GE3_A()
# GE3_B()

```

```

def bit_calc(BITS, x):
    ub = 2**BITS
    ub = ub / x
    lb = ub // 2
    r = random.randrange(lb, ub)
    p = x*r + 1
    while not isprime(p):
        r = random.randrange(lb, ub)
        p = x*r + 1
    print("x =", x, "\tBits ---> r:", r.bit_length(), "-- x:",
          x.bit_length(), "-- p:", p.bit_length())
    return p

for i in range(1, 20):
    bit_calc(256, i)

```

Παράρτημα Β

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df1 = pd.read_csv('original.csv')
df2 = pd.read_csv('fast_gen_prime.csv')

df1.head()

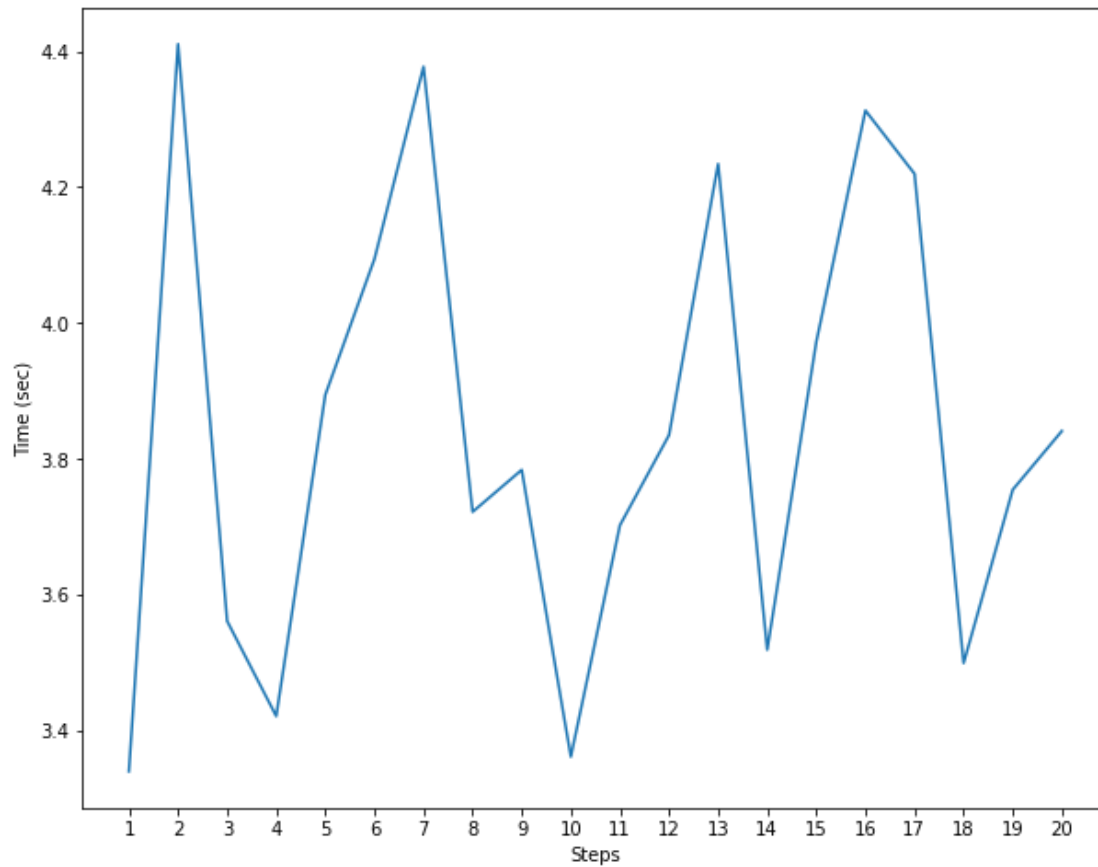
   index  steps    time
0      0      1  3.340640
1      1      2  4.410328
2      2      3  3.561766
3      3      4  3.421869
4      4      5  3.894567

df2.head()

   index  steps    time
0      0      1  2.471590
1      1      2  0.149822
2      2      3  0.111964
3      3      4  0.120070
4      4      5  0.108098

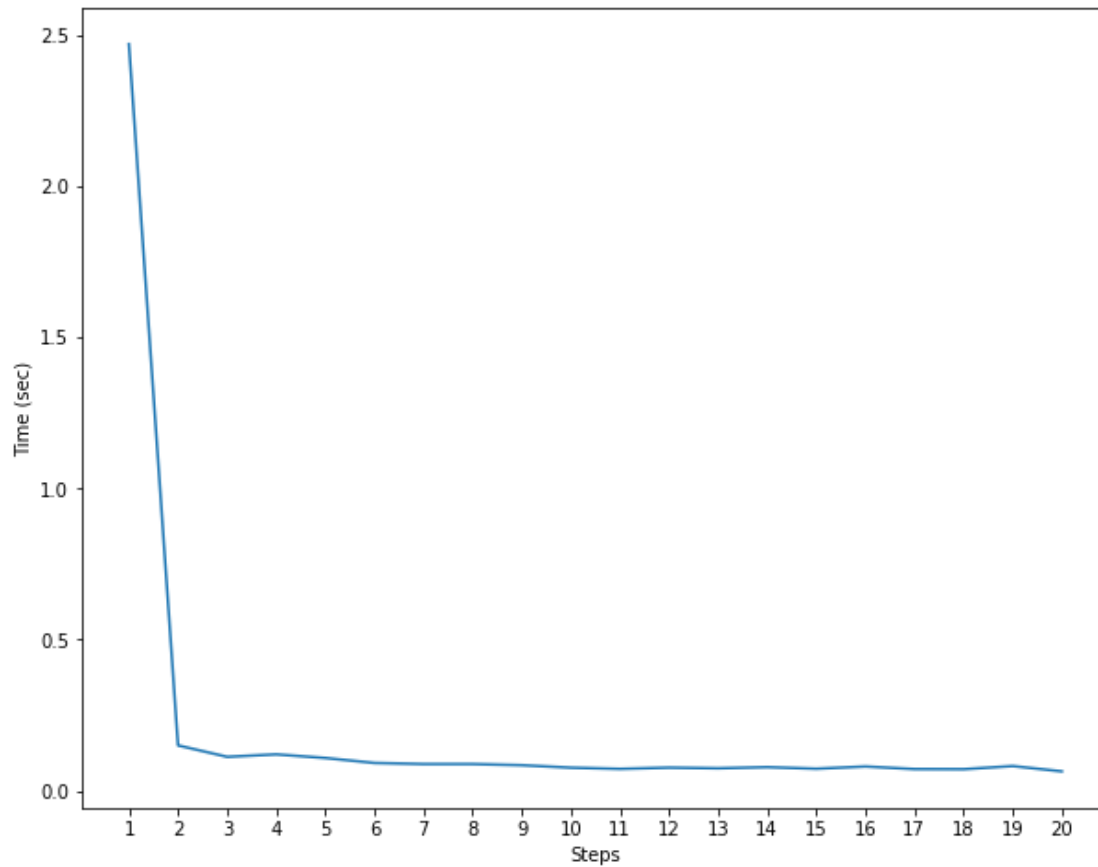
df1 = df1.drop('index',axis=1)
df2 = df2.drop('index',axis=1)
```

```
plt.figure(figsize=(10, 8))  
x = range(len(df1['steps']))  
plt.plot(x, df1['time'])  
plt.xticks(x, df1['steps'])  
plt.xlabel('Steps')  
plt.ylabel('Time (sec)')  
plt.show()
```



Εικόνα 19. Χρόνοι ανά βήμα αρχικού κώδικα

```
plt.figure(figsize=(10, 8))
x = range(len(df2['steps']))
plt.plot(x, df2['time'])
plt.xticks(x, df2['steps'])
plt.xlabel('Steps')
plt.ylabel('Time (sec)')
plt.show()
```

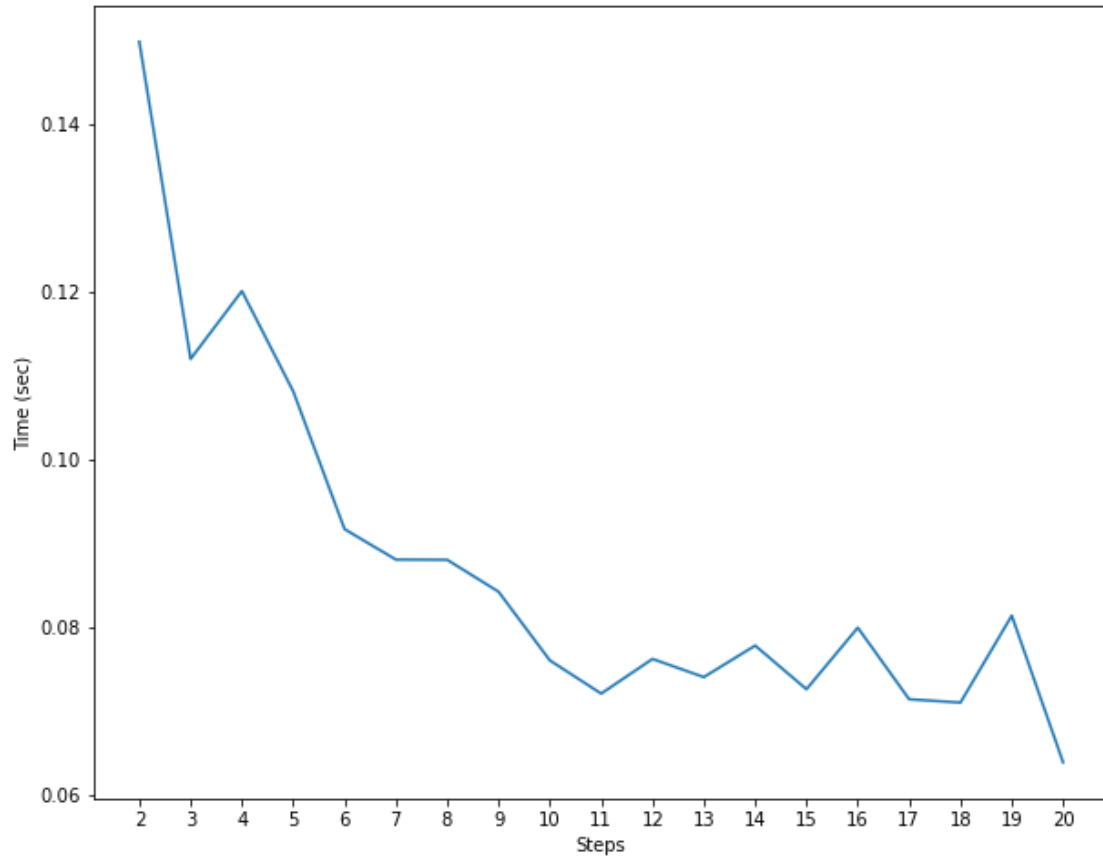


Εικόνα 20. Χρόνοι ανά βήμα συνάρτησης *fast_gen_prime*

```
df3 = df2[1:]
df3.head()
```

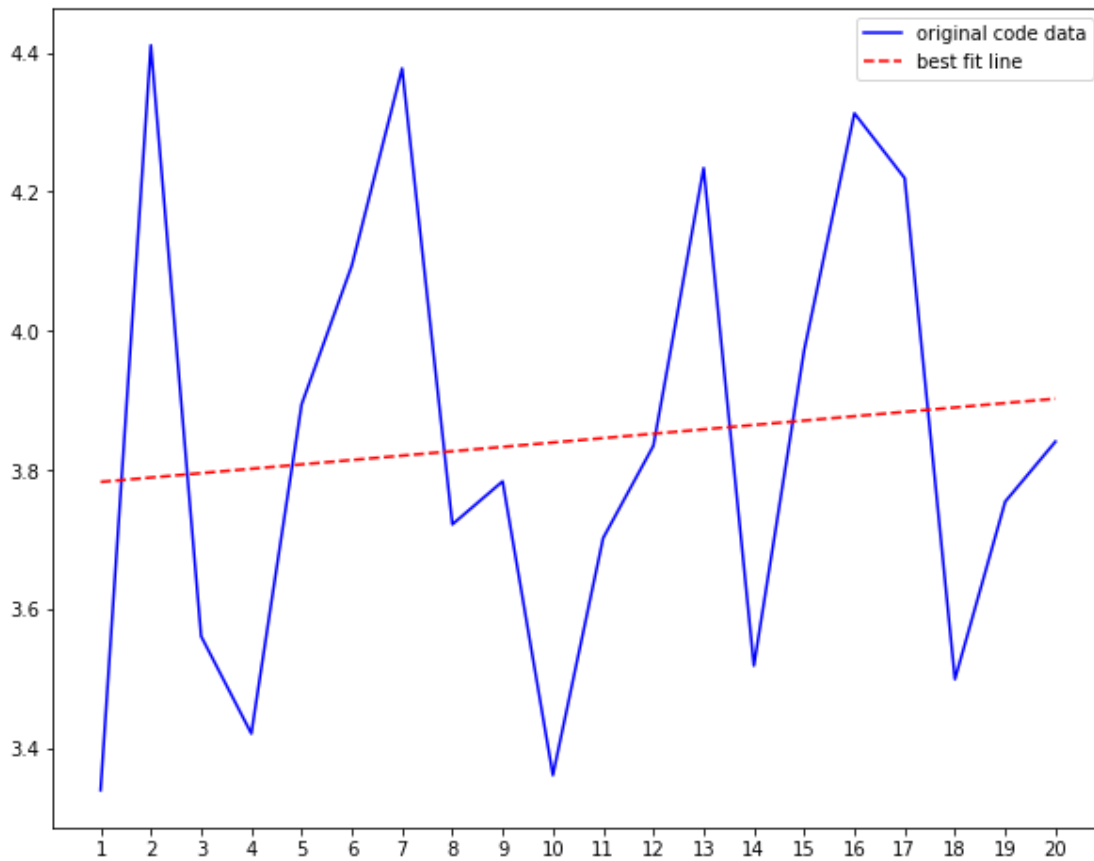
	steps	time
1	2	0.149822
2	3	0.111964
3	4	0.120070
4	5	0.108098
5	6	0.091660

```
plt.figure(figsize=(10, 8))
x = range(len(df3['steps']))
plt.plot(x, df3['time'])
plt.xticks(x, df3['steps'])
plt.xlabel('Steps')
plt.ylabel('Time (sec)')
plt.show()
```



Εικόνα 21. Χρόνοι βημάτων 2-20 συνάρτησης *fast_gen_prime*

```
list1 = df1['time'].to_numpy()
plt.figure(figsize=(10, 8))
x = range(len(df1['steps']))
plt.plot(list1, color='blue', label='original code data')
z1 = np.polyfit(np.arange(len(list1)), list1, 1)
p1 = np.poly1d(z1)
plt.plot(np.arange(len(list1)), p1(np.arange(len(list1))),
         color='red', label='best fit line', linestyle='--')
plt.xticks(x, df1['steps'])
plt.legend()
plt.show()
```

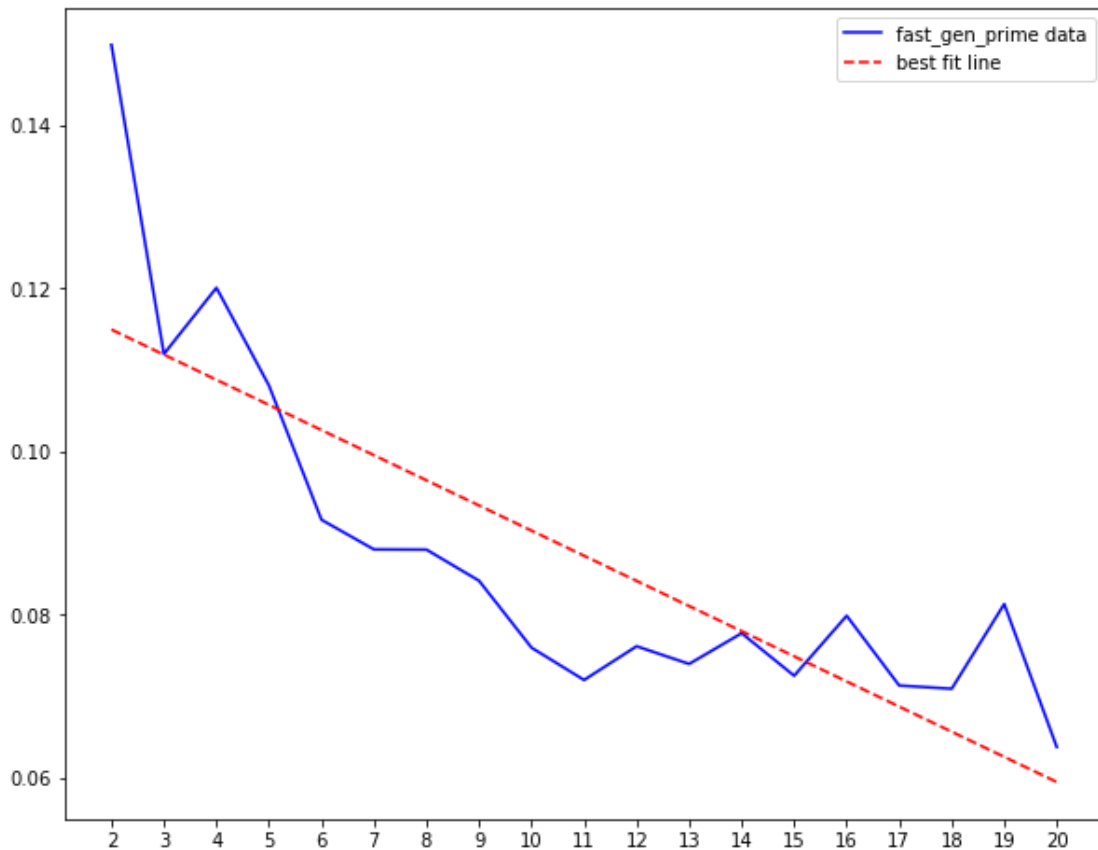


Εικόνα 22. Συνάρτηση καλύτερης προσαρμογής αρχικού κώδικα


```

list2 = df3['time'].to_numpy()
plt.figure(figsize=(10, 8))
x = range(len(df3['steps']))
plt.plot(list2, color='blue', label='fast_gen_prime data')
z = np.polyfit(np.arange(len(list2)), list2, 1)
p = np.poly1d(z)
plt.plot(np.arange(len(list2)), p(np.arange(len(list2))),
         color='red', label='best fit line', linestyle='--')
plt.xticks(x, df3['steps'])
plt.legend()
plt.show()

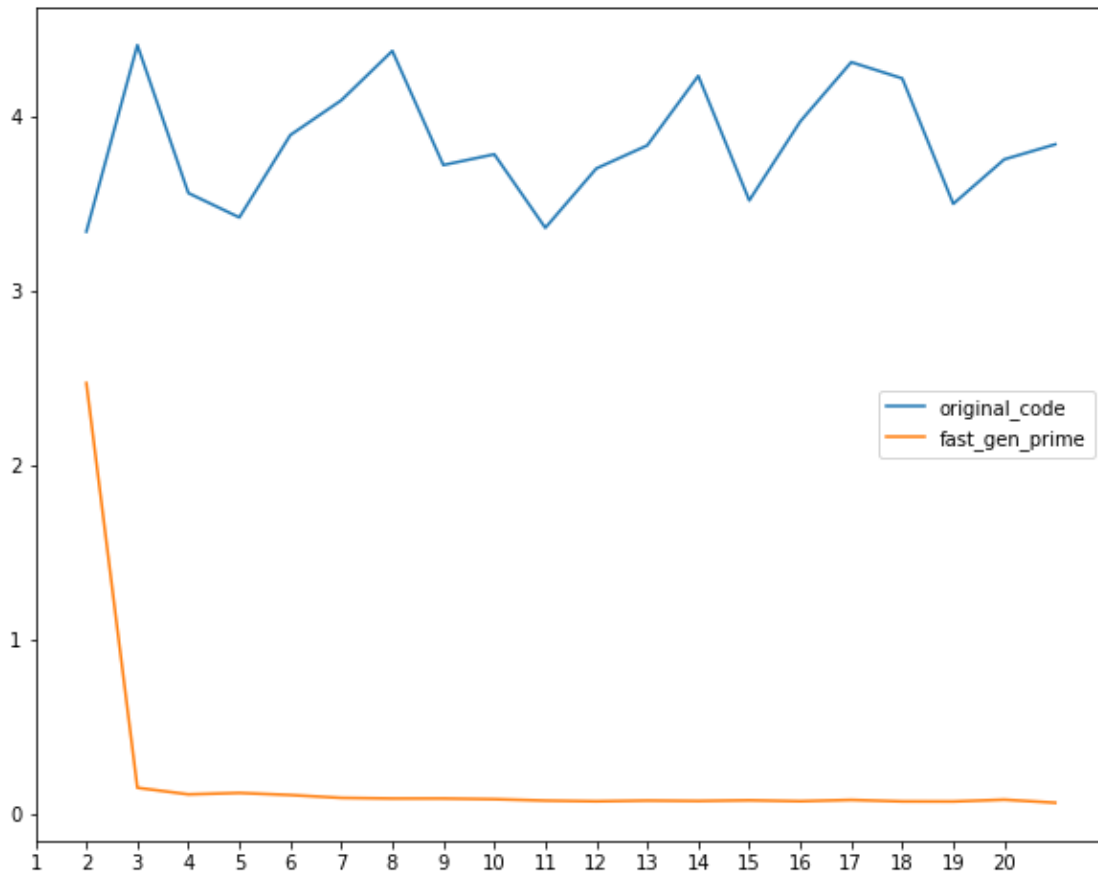
```



Εικόνα 23. Συνάρτηση καλύτερης προσαρμογής fast_gen_prime

```
plt.figure(figsize=(10, 8))
x = range(len(df1['steps']))
plt.plot(df1['steps'], df1['time'], label='original_code')
plt.plot(df2['steps'], df2['time'], label='fast_gen_prime')
plt.xticks(x, df1['steps'])
plt.legend()
```

<matplotlib.legend.Legend at 0x2361c85c3a0>

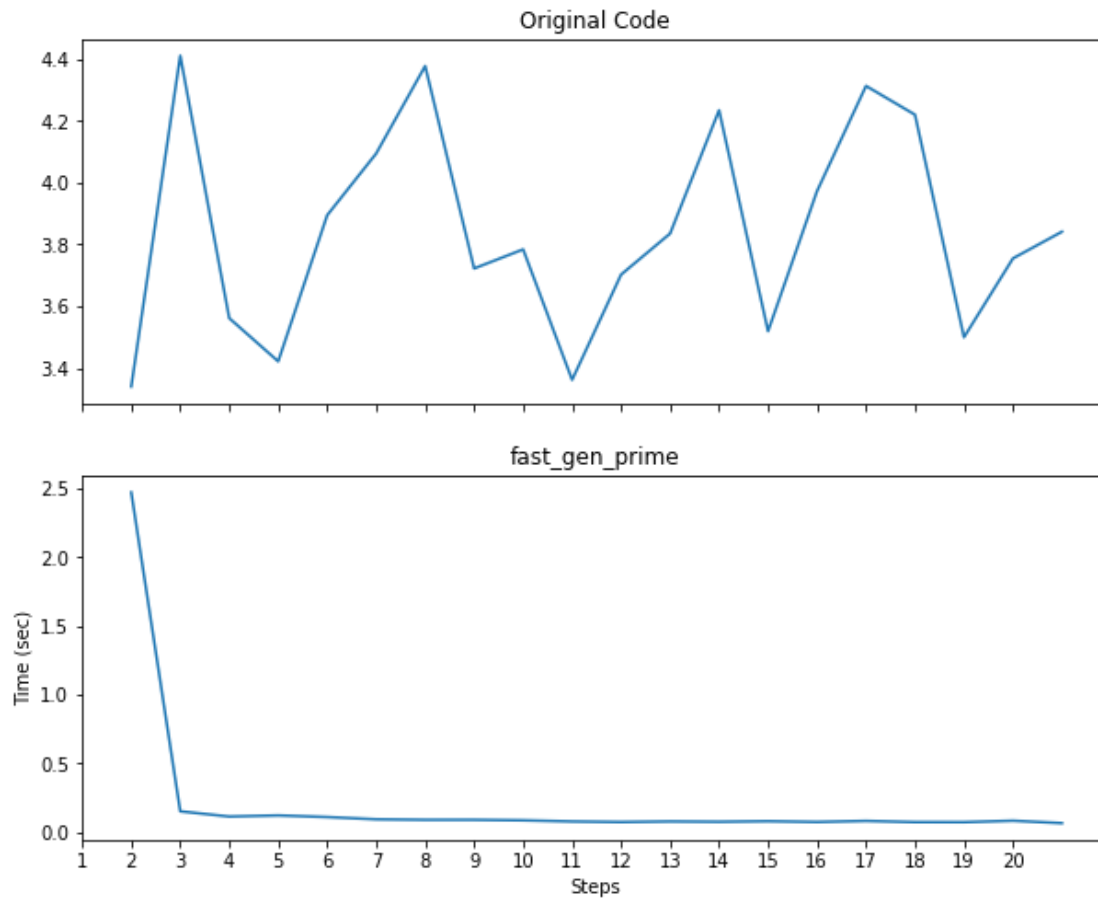


Εικόνα 24. Γράφημα χρόνων συναρτήσεων ανά βήμα.

```

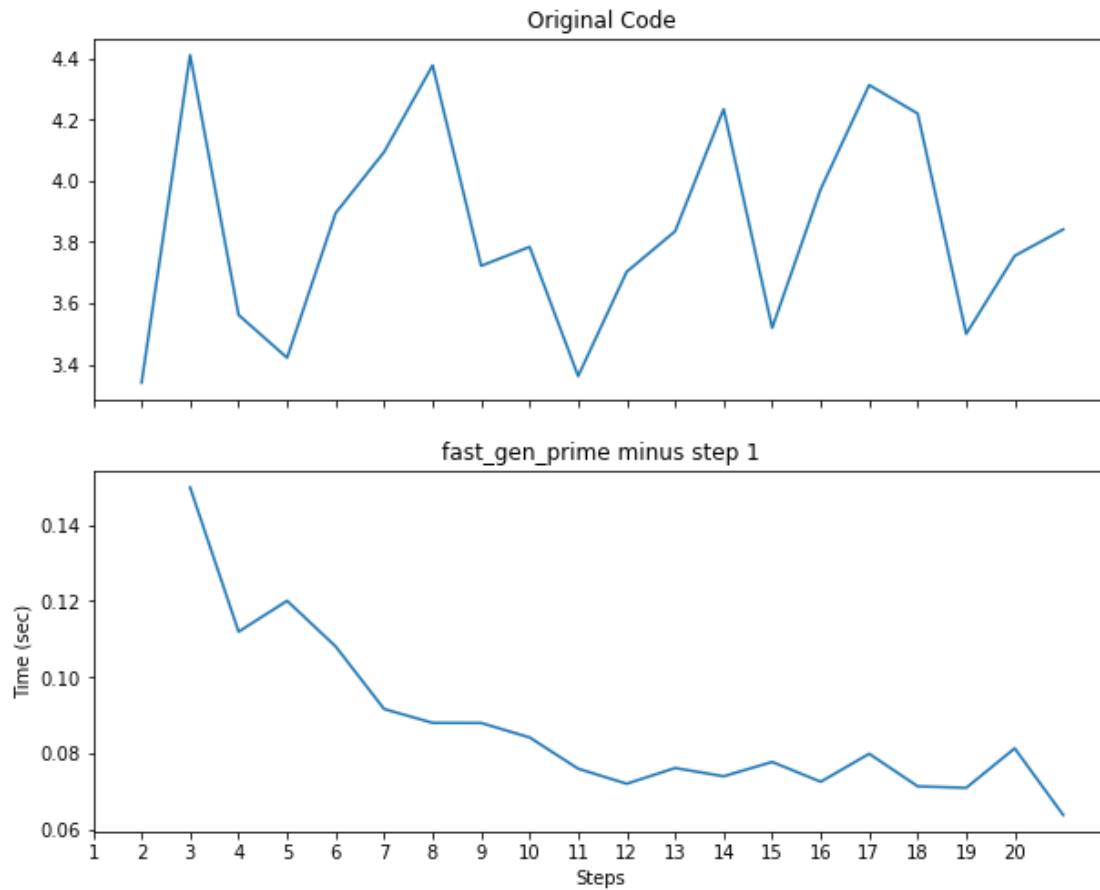
fig, ax = plt.subplots(2,
                        sharex='col', sharey='row', figsize=(10,8))
ax[0].plot(df1['steps'], df1['time'], label='original_code')
ax[0].set(title='Original Code')
ax[1].plot(df2['steps'], df2['time'], label='fast_gen_prime')
ax[1].set(xlabel='Steps', ylabel='Time (sec)',
          title='fast_gen_prime')
x = range(len(df1['steps']))
plt.xticks(x, df1['steps'])
plt.show()

```



Εικόνα 25. Παράλληλο γράφημα συνολικών χρόνων ανα βήμα

```
fig, ax = plt.subplots(2,
                        sharex='col', sharey='row', figsize=(10,8))
ax[0].plot(df1['steps'], df1['time'])
ax[0].set(title='Original Code')
ax[1].plot(df3['steps'], df3['time'])
ax[1].set(xlabel='Steps', ylabel='Time (sec)',
          title='fast_gen_prime minus step 1')
x = range(len(df1['steps']))
plt.xticks(x, df1['steps'])
plt.show()
```



Εικόνα 26. Παράλληλο γράφημα χρόνων ανα βήμα