Βασίλειος Τσιτώτας 4296 *Project* 2

1 Σημείωση

Καποιοι κώδικες πέρνουν μεχρι και 2 λεπτα να βγαλουν αποτελεσμα και διαφερει αναλογα με την μηχανη σας αλλα τρεχουν και δινουν αποδεκτο αποτελεσμα

Άσκηση 1 (7.4)

Ο παρών αλγόριθμος υπολογίζει την τιμή της παράστασης

$$2^{1000} \cdot 3^{101} \cdot 5^{47} \mod (2^{107} - 1)$$

κάνοντας χρήση της **γρήγορης ύψωσης σε δύναμη** και του **πολλαπλασιασμού** $Karatsuba$.

- Η $karatsuba(x, y)$ υλοποιεί τον αναδρομικό αλγόριθμο $Karatsuba$, μειώνοντας την υπολογιστική πολυπλοκότητα του πολλαπλασιασμού μεγάλων αριθμών.
- Η $fast\_mod\_exp(base, exponent, modulus)$ υλοποιεί τη γρήγορη ύψωση σε δύναμη με επαναληπτικό τρόπο, χρησιμοποιώντας $Karatsuba$ αντί για τον απλό τελεστή `*`.
- Οι τρεις επιμέρους εκφράσεις `2^1000`, `3^101` και `5^47` υπολογίζονται ξεχωριστά $mod\ 2^{107} - 1$.
- Τέλος, τα αποτελέσματα πολλαπλασιάζονται μεταξύ τους με χρήση $Karatsuba$ και λαμβάνεται το τελικό υπόλοιπο ως αποτέλεσμα.

Άσκηση 2 (8.4)

Θεώρημα. Για κάθε ακέραιο $m \geq 2$, ο αριθμός 2^m δεν μπορεί να εκφραστεί ως άθροισμα της μορφής

$$a + (a + 1) + (a + 2) + \cdots + (a + k - 1),$$

για κανένα $a \in \mathbb{N}$ και $k \geq 2$.

Απόδειξη. Έστω προς άτοπο ότι υπάρχει $a \in \mathbb{N}$ και $k \geq 2$ ώστε

$$2^m = a + (a+1) + \cdots + (a+k-1).$$

Το άθροισμα k διαδοχικών θετικών ακεραίων δίνεται από τον τύπο:

$$a + (a+1) + \cdots + (a+k-1) = \frac{k(2a+k-1)}{2}.$$

Άρα:

$$2^m = \frac{k(2a+k-1)}{2} \quad \Rightarrow \quad k(2a+k-1) = 2^{m+1}.$$

Δηλαδή, το γινόμενο $k(2a+k-1)$ είναι δύναμη του 2.

Επειδή $k \geq 2$ και διαιρεί το 2^{m+1}, το k είναι δύναμη του 2. Ομοίως, για να είναι και το $2a+k-1$ ακέραιος θετικός και το γινόμενο να παραμένει δύναμη του 2, πρέπει ο παράγοντας $2a+k-1$ να είναι και δύναμη του 2.

Όμως, παρατηρούμε ότι:

- Το $2a$ είναι άρτιος.
- Το k είναι άρτιος (αφού είναι δύναμη του 2).
- Άρα, $2a+k$ είναι άρτιος.
- Επομένως, $2a+k-1$ είναι περιττός αριθμός.

Καμία δύναμη του 2 (πέραν του $2^0 = 1$) δεν είναι περιττός αριθμός. Άρα ο αριθμός $2a+k-1$ δεν μπορεί να είναι δύναμη του 2, και έτσι καταλήγουμε σε άτοπο.

Άρα ο αριθμός 2^m, με $m \geq 2$, δεν μπορεί να γραφεί ως άθροισμα διαδοχικών θετικών ακεραίων.

Άσκηση 3 (8.21)

Θεώρημα 1. *Έστω p και q διαδοχικοί πρώτοι αριθμοί τέτοιοι ώστε $3 < p < q$. Τότε, κάθε ακέραιος $r < q$ διαιρεί το $p!$.*

Απόδειξη. *Θεωρούμε έναν ακέραιο $r < q$ και διακρίνουμε δύο περιπτώσεις:*

1η περίπτωση: *Αν $r \leq p$, τότε r είναι μικρότερος ή ίσος του p, και άρα περιλαμβάνεται στο γινόμενο*

$$p! = 1 \cdot 2 \cdot 3 \cdots p,$$

οπότε προφανώς $r \mid p!$.

2η περίπτωση: *Αν $p < r < q$, τότε ο r βρίσκεται ανάμεσα σε δύο διαδοχικούς πρώτους: τον p και τον q. Εφόσον δεν υπάρχουν άλλοι πρώτοι μεταξύ τους, ο r δεν μπορεί να είναι πρώτος. Άρα είναι σύνθετος, και γράφεται ως γινόμενο πρώτων:*

$$r = \pi_1^{\alpha_1} \cdot \pi_2^{\alpha_2} \cdots \pi_k^{\alpha_k},$$

όπου κάθε π_i είναι πρώτος μικρότερος του r και συνεπώς μικρότερος του q. Αλλά επειδή q είναι ο πρώτος αμέσως μετά τον p, θα ισχύει ότι $\pi_i \leq p$ για κάθε i.

Άρα όλοι οι πρώτοι παράγοντες του r είναι μικρότεροι ή ίσοι του p και επομένως περιέχονται στο $p!$. Συνεπώς:

$$r \mid p!.$$

Αρα για κάθε ακέραιο $r < q$, ισχύει $r \mid p!$.

Άσκηση 4 (8.33)

1. Νδο $\gcd(ac, b) = \gcd(c, b)$ για κάθε $c \in \mathbb{Z}$

Απόδειξη: Έστω $d = \gcd(c, b)$. Τότε υπάρχουν ακέραιοι c', b' τέτοιοι ώστε:

$$c = d \cdot c', \quad b = d \cdot b', \quad \text{με } \gcd(c', b') = 1$$

Επομένως:

$$ac = a(dc') = d(ac') \Rightarrow \gcd(ac, b) = \gcd(dac', db') = d \cdot \gcd(ac', b')$$

Επειδή $\gcd(a, b) = 1$ και $\gcd(c', b') = 1$, προκύπτει $\gcd(ac', b') = 1$. Άρα:

$$\gcd(ac, b) = d = \gcd(c, b)$$

Άρα $\gcd(ac, b) = \gcd(c, b)$

2. Νδο $\gcd(a + b, a - b) \in \{1, 2\}$, και αν a, b περιττοί, τότε $\gcd(a + b, a - b) = 2$

Απόδειξη: Έστω $d = \gcd(a + b, a - b)$. Τότε:

$$d \mid (a + b) - (a - b) = 2b \Rightarrow d \mid 2b$$

και

$$d \mid (a + b) + (a - b) = 2a \Rightarrow d \mid 2a$$

Άρα:

$$d \mid \gcd(2a, 2b) = 2 \cdot \gcd(a, b) = 2 \cdot 1 = 2 \Rightarrow d \in \{1, 2\}$$

Ειδική περίπτωση: Αν a, b περιττοί, τότε $a + b$ και $a - b$ είναι άρτιοι, άρα $2 \mid \gcd(a + b, a - b)$, δηλαδή η *gcd* είναι διαιρετή με το 2. Επειδή μόνο οι τιμές 1 και 2 είναι δυνατές, θα είναι αναγκαστικά ίση με 2:

$$\gcd(a + b, a - b) = 2$$

3. Νδο Αν $M_p = 2^p - 1, M_q = 2^q - 1$, με p, q πρώτους και $p \neq q$, τότε $\gcd(M_p, M_q) = 1$

Απόδειξη: Ισχύει η γενική ταυτότητα:

$$\gcd(2^m - 1, 2^n - 1) = 2^{\gcd(m,n)} - 1$$

Αν p, q είναι διαφορετικοί πρώτοι, τότε $\gcd(p, q) = 1$, οπότε:

$$\gcd(2^p - 1, 2^q - 1) = 2^1 - 1 = 1$$

Άρα:

$$\gcd(M_p, M_q) = 1$$

Άσκηση 5 (8.50)

Ψλοποίηση ενός προγράμματος σε *Python* που επαληθεύει πειραματικά την ανισότητα

$$\frac{\sigma(n)}{n} < \frac{e^\gamma}{2} \ln \ln n + \frac{0.74}{\ln \ln n}$$

για όλους τους περιττούς θετικούς ακέραιους $n < 220$, όπου $\sigma(n)$ είναι το άθροισμα των θετικών διαιρετών του n και γ η σταθερά του *Euler Mascheroni*

Περιγραφή Κώδικα

Ο κώδικας υπολογίζει για κάθε περιττό $n < 220$:

- Την τιμή της συνάρτησης $\sigma(n)$ ως το άθροισμα όλων των διαιρετών του n.
- Το πηλίκο $\frac{\sigma(n)}{n}$.
- Τη δεξιά πλευρά της ανισότητας, υπολογίζοντας το $\frac{e^\gamma}{2} \ln \ln n + \frac{0.74}{\ln \ln n}$.
- Συγκρίνει τα δύο μέρη και εκτυπώνει αν η ανισότητα ισχύει ή όχι.

Παράλληλα, ο κώδικας χειρίζεται περιπτώσεις όπου το $\ln \ln n$ δεν είναι ορισμένο (για πολύ μικρά n).

Αποτελέσματα

Η εκτέλεση του προγράμματος δείχνει ότι για όλους τους περιττούς ακέραιους $n < 220$ η ανισότητα ισχύει, επιβεβαιώνοντας πειραματικά το θεωρητικό πλαίσιο.

Επομένως, για το συγκεκριμένο πεπερασμένο σύνολο αριθμών, το πηλίκο $\frac{\sigma(n)}{n}$ δεν υπερβαίνει το συγκεκριμένο άνω φράγμα.

2 Άσκηση 6(8.76)

Ο παρακάτω αλγόριθμος υπολογίζει τον επόμενο τέλειο αριθμό μεγαλύτερο του 8128, αξιοποιώντας τον γνωστό τύπο του Ευκλείδη για τέλειους αριθμούς:

$$\text{Τέλειος αριθμός} = 2^{p-1} \cdot (2^p - 1)$$

όπου $(2^p - 1)$ είναι πρώτος αριθμός $Mersenne$. Ο αλγόριθμος περιλαμβάνει τα εξής βασικά στάδια:

- **Έλεγχος πρωτογονίας:** Συνάρτηση $is - prime$ που ελέγχει αν ένας αριθμός είναι πρώτος, περιορίζοντας τις διαιρέσεις μέχρι τη τετραγωνική ρίζα του αριθμού και αποφεύγοντας περιττούς ελέγχους.
- **Εύρεση αριθμών** $Mersenne$**:** Ξεκινώντας από $p = 17$, υπολογίζεται ο αριθμός $Mersenne$ $2^p - 1$ και ελέγχεται αν είναι πρώτος.
- **Υπολογισμός τέλειου αριθμού:** Αν ο αριθμός $Mersenne$ είναι πρώτος, υπολογίζεται ο αντίστοιχος τέλειος αριθμός και ελέγχεται αν είναι μεγαλύτερος από το 8128.
- **Μέτρηση χρόνου εκτέλεσης:** Χρησιμοποιείται η $time.perf-counter()$ για ακριβή μέτρηση του χρόνου εκτέλεσης της διαδικασίας.

Ο αλγόριθμος είναι αποδοτικός διότι περιορίζει την αναζήτηση μόνο στους υποψήφιους εκθέτες p που οδηγούν σε αριθμούς $Mersenne$, και δεν εξετάζει όλους τους φυσικούς αριθμούς σειριακά.

Αποτελέσματα

Ο επόμενος τέλειος αριθμός ' 8128 είναι: 8589869056 Χρόνος εκτέλεσης: 0.0000111000 δευτερόλεπτα

3 Άσκηση 7 (9.29)

Περιγραφή Κώδικα

Ο παρακάτω κώδικας υλοποιεί το **τεστ του** $Fermat$ για πιθανολογικό έλεγχο πρώτων αριθμών, αξιοποιώντας:

- τη βιβλιοθήκη $gmpy2$ για γρήγορους υπολογισμούς με μεγάλους αριθμούς ακριβείας (mpz),
- το $ProcessPoolExecutor$ από τη βιβλιοθήκη $concurrent.futures$ για την παράλληλη εκτέλεση των ελέγχων με διαφορετικές βάσεις a,
- και το γεγονός ότι με k ανεξάρτητες επαναλήψεις, η πιθανότητα να χαρακτηριστεί λανθασμένα ένας σύνθετος αριθμός ως πρώτος είναι το πολύ 2^{-k}.

Η βασική συνάρτηση $is\_probable\_prime(n, k)$ δέχεται έναν θετικό ακέραιο n και έναν αριθμό επαναλήψεων k. Σε κάθε επανάληψη επιλέγεται τυχαία μια βάση a στο διάστημα $[2, n-2]$ και ελέγχεται αν ισχύει:

$$a^{n-1} \mod n = 1$$

Εάν η παραπάνω σχέση ισχύει για όλες τις επιλεγμένες βάσεις, τότε ο αριθμός θεωρείται **πιθανώς πρώτος**. Διαφορετικά, απορρίπτεται ως σύνθετος. Οι υπολογισμοί αυτοί διαμοιράζονται σε πολλούς πυρήνες του επεξεργαστή για καλύτερη απόδοση.

Η συνάρτηση $test\_large\_number()$ εφαρμόζει τον έλεγχο σε έναν πολύ μεγάλο αριθμό της μορφής:

$$n = 3714089895285 \cdot 2^{60000} - 1$$

Εάν αυτός περάσει το τεστ του $Fermat$ με k επαναλήψεις, τότε εξετάζεται και ο αριθμός:

$$2n + 1$$

για να διαπιστωθεί αν και αυτός είναι πιθανώς πρώτος. Αν ναι, τότε ο n χαρακτηρίζεται ως **πιθανός** *Sophie Germain prime*, δηλαδή πρώτος για τον οποίο ο αριθμός $2n + 1$ είναι επίσης πρώτος.

Ανάλυση Τεστ του *Fermat* και Ιδιότητας *Sophie Germain*

Το **τεστ του Φερματ** βασίζεται στο μικρό θεώρημα του Φερματ, σύμφωνα με το οποίο: Αν n είναι πρώτος και a ακέραιος με $2 \leq a \leq n-2$, τότε:

$$a^{n-1} \equiv 1 \mod n$$

Ωστόσο, υπάρχουν σύνθετοι αριθμοί (οι λεγόμενοι *ψευδοπρώτοι του Fermat*) που ικανοποιούν αυτή τη σχέση για πολλές ή όλες τις βάσεις a. Για τον λόγο αυτό, το τεστ του $Fermat$ θεωρείται **πιθανολογικό** και όχι απόλυτα αξιόπιστο.

Συμπέρασμα. Από την εκτέλεση του κώδικα για:

$$n = 3714089895285 \cdot 2^{60000} - 1$$

με $k = 40$ επαναλήψεις του τεστ $Fermat$, προέκυψαν τα εξής αποτελέσματα:

- Ο αριθμός n πέρασε με επιτυχία όλους τους ελέγχους.
- Η πιθανότητα σφάλματος είναι περίπου $2^{-40} \approx 10^{-12}$, άρα το αποτέλεσμα χαρακτηρίζεται ως:

Πιθανώς ΠΡΩΤΟΣ (με πιθανότητα σφάλματος $\sim 10^{-12}$)

- Ακολούθως υπολογίστηκε το μέγεθος του $2n+1$ (περίπου 60043 *bits*) και εκτελέστηκε το ίδιο τεστ.
- Το αποτέλεσμα ήταν επίσης θετικό:

Είναι πιθανώς *Sophie Germain prime*!

Τελικό συμπέρασμα: Ο αριθμός:

$$n = 3714089895285 \cdot 2^{60000} - 1$$

είναι **πιθανώς πρώτος** και ταυτόχρονα **πιθανώς** *Sophie Germain prime*, σύμφωνα με 40 ανεξάρτητες δοκιμές του τεστ του *Fermat* για κάθε αριθμό.

Άσκηση 8 (9.44)

Ο κώδικας αποτελείται από τρία βασικά μέρη:

1. **Συνάρτηση ελέγχου πρώτου αριθμού:** $is\_prime(n)$
   - Ελέγχει αν ο αριθμός n είναι πρώτος.
   - Επιστρέφει $False$ για $n < 2$.
   - Επιστρέφει $True$ για $n = 2$ ή 3.
   - Επιστρέφει $False$ αν ο n είναι ζυγός (διαιρείται με 2).
   - Ελέγχει αν ο n έχει διαιρέτες από το 3 μέχρι $\sqrt{n}$ με βήμα 2 (μόνο περιττοί).
   - Αν βρεθεί διαιρέτης επιστρέφει $False$, αλλιώς $True$.
2. **Συνάρτηση ελέγχου ισχυρού ψευδοπρώτου:** $is\_strong\_pseudoprime(n, a)$
   - Εφαρμόζει το τεστ$Miller - Rabin$ ως εξής:
   - Διασπά το $n-1$ σε $2^r \cdot d$ όπου d περιττός.
   - Υπολογίζει $x = a^d \pmod{n}$.
   - Αν $x \equiv 1 \pmod{n}$ ή $x \equiv n-1 \pmod{n}$, ο αριθμός περνάει το τεστ (επιστρέφει $True$).
   - Αλλιώς, τετραγωνίζει επαναληπτικά x έως $r-1$ φορές:

     $$x \leftarrow x^2 \pmod{n}$$

     και ελέγχει αν σε κάποιο βήμα $x \equiv n-1 \pmod{n}$. Αν ναι, επιστρέφει $True$.
   - Αν κανένα από τα παραπάνω δεν ισχύει, επιστρέφει $False$.
3. **Βρόχος αναζήτησης του μικρότερου ισχυρού ψευδοπρώτου**

- Ξεκινά από τον αριθμό $n = 3$ και αυξάνει κατά 2 (μόνο περιττοί αριθμοί).
- Για κάθε n, πρώτα ελέγχει αν είναι σύνθετος (δηλαδή όχι πρώτος).
- Αν είναι σύνθετος, ελέγχει αν είναι ισχυρός ψευδοπρώτος ως προς τη βάση $a = 32$.
- Μόλις βρεθεί τέτοιος αριθμός, τον εκτυπώνει και σταματάει.

Με αυτόν τον τρόπο βρίσκουμε τον *μικρότερο σύνθετο αριθμό* που περνάει το τεστ $Miller-Rabin$ για τη βάση 32, δηλαδή τον **μικρότερο ισχυρό ψευδοπρώτο αριθμό ως προς τη βάση 32 και ειναι ο 25.**

Άσκηση 9 (10.1)

Ο παρακάτω κώδικας σε *Python* υλοποιεί τον αλγόριθμο δοκιμαστικής διαίρεσης (*trial division*) για την παραγοντοποίηση των αριθμών $2^{62} - 1$ και $2^{102} - 1$.

- Η συνάρτηση $trial\_division(n)$ δέχεται έναν ακέραιο αριθμό n και επιστρέφει μια λίστα με τους πρώτους παραγοντές του.
- Ελέγχεται πρώτα αν ο αριθμός είναι μικρότερος του 2 (μη παραγοντοποιήσιμος).
- Γίνεται ξεχωριστός έλεγχος για τον διαιρέτη 2 (πρώτος άρτιος αριθμός).
- Στη συνέχεια, δοκιμάζονται όλοι οι περιττοί αριθμοί από το 3 έως και $\sqrt{n}$ για το αν διαιρούν το n.
- Όταν βρεθεί διαιρέτης, ο αριθμός διαιρείται και ο διαιρέτης προστίθεται στη λίστα. Το όριο max_factor ενημερώνεται επειδή το n μειώνεται.
- Τέλος, αν απομείνει ένας πρώτος αριθμός μεγαλύτερος του 1, προστίθεται και αυτός στους παραγοντές.

Το πρόγραμμα υπολογίζει και εκτυπώνει τους πρώτους παραγοντές για τους αριθμούς:
$n_1 = 2^{62} - 1 = 4611686018427387903$
`[3, 715827883, 2147483647]`
$n_2 = 2^{102} - 1 = 5070602400912917605986812821504$
`[3, 3, 7, 103, 307, 2143, 2857, 6529, 11119, 43691, 131071]`
Ο αλγόριθμος είναι απλός αλλά αργός για πολύ μεγάλους αριθμούς

Άσκηση 10 (10.20)

Για την επίλυση της εξίσωσης $g^x \equiv y \pmod{p}$ με $p = 3571$, $g = 2$, και $y = 2763$, υλοποιήθηκε ο αλγόριθμος ρ του *Pollard*.

Ο αλγόριθμος στηρίζεται στον εντοπισμό κύκλου (*cycle detection*) μέσω της μεθόδου του *Floyd* με στόχο την εύρεση σύγκρουσης $X_i = X_{2i}$ σε κάποια επανάληψη.

Ορίζεται μια συνάρτηση $f(x, a, b)$ που μεταβάλλει τις τιμές x, a, b ως εξής:

$$f(x,a,b) = \begin{cases} (x^2 \bmod p,\ 2a \bmod (p-1),\ 2b \bmod (p-1)), & \text{αν } x \bmod 3 = 0 \\ (xg \bmod p,\ a+1 \bmod (p-1),\ b), & \text{αν } x \bmod 3 = 1 \\ (xy \bmod p,\ a,\ b+1 \bmod (p-1)), & \text{αν } x \bmod 3 = 2 \end{cases}$$

Όταν εντοπιστεί σύγκρουση $X = Y$, ισχύει η σχέση:

$$g^a y^b \equiv g^A y^B \pmod{p} \Rightarrow g^{a-A} \equiv y^{B-b} \pmod{p} \Rightarrow g^{a-A} \equiv g^{x(B-b)} \pmod{p}$$

Από την οποία προκύπτει:

$$x \equiv (a-A)(B-b)^{-1} \mod (p-1)$$

Αν $B - b$ είναι αντιστρέψιμο $mod\ p-1$, τότε υπολογίζεται το x. Αν όχι, ο αλγόριθμος επαναλαμβάνεται με νέες τυχαίες αρχικές τιμές έως ότου βρεθεί έγκυρη λύση.

Στο συγκεκριμένο πρόβλημα, ο αλγόριθμος βρήκε επιτυχώς τη λύση $x = 900$, για την οποία ισχύει:

$$2^{900} \equiv 2763 \pmod{3571}$$

Άσκηση 11 (10.21)

Ο αλγόριθμος του *Pollard* ρ είναι μια στοχαστική μέθοδος παραγοντοποίησης φυσικών αριθμών. Η παρακάτω υλοποίηση χρησιμοποιεί το *multiprocessing module* για την παράλληλη εκτέλεση του αλγορίθμου σε πολλούς πυρήνες.

1. Συνάρτηση *pollard_worker*

Πρόκειται για μια συνάρτηση που εκτελεί μία ανεξάρτητη προσπάθεια παραγοντοποίησης του αριθμού N. Χρησιμοποιεί τη συνάρτηση $f(x) = (x^2 + 1) \mod N$ για να δημιουργήσει μια ψευδοτυχαία ακολουθία.

- Ξεκινάει με τυχαία τιμή $x \in [2, N-2]$
- Υπολογίζει ακολουθίες x_i και y_i με χρήση της τεχνικής *Floyd cycle detection*
- Όταν βρει $\gcd(|x-y|, N) > 1$ και $< N$, επιστρέφει τον διαιρέτη

2. Συνάρτηση *parallel_pollard*

Αυτή η συνάρτηση ξεκινάει **πολλαπλές διεργασίες** (όχι *threads*), που εκτελούν ταυτόχρονα τον *Pollard* ρ αλγόριθμο με διαφορετικές αρχικές τιμές.

- Δημιουργεί έναν $multiprocessing.Queue$ για να διαβάσει τον πρώτο επιτυχημένο διαιρέτη
- Δημιουργεί $num\_processes$ διεργασίες, η καθεμία με ανεξάρτητο x_0
- Μόλις μια διεργασία βρει διαιρέτη και τον βάλει στην ουρά, οι υπόλοιπες τερματίζονται

3.$Main$

- Ορίζεται ο αριθμός $N = 2^{257} - 1$
- Μετριέται ο χρόνος εκτέλεσης με $time.time()$
- Καλείται η $parallel\_pollard(N, 8)$ για χρήση 8 πυρήνων
- Εκτυπώνεται ο πρώτος μη τετριμμένος διαιρέτης και ο συνολικός χρόνος

4. Αποτέλεσμα Εκτέλεσης

Ο αλγόριθμος εκτελέστηκε επιτυχώς χρησιμοποιώντας 8 διεργασίες. Μετά από τυχαίες ανεξάρτητες προσπάθειες παραγοντοποίησης, βρέθηκε ο παρακάτω μη τετριμμένος διαιρέτης:

Ένας μη τετριμμένος διαιρέτης του $N = 2^{257} - 1$ είναι:
535006138814359

Ο συνολικός χρόνος εκτέλεσης ήταν:

17.584000 δευτερόλεπτα

Αυτό καταδεικνύει την αποδοτικότητα της παράλληλης εκτέλεσης του αλγορίθμου $Pollard$ καθώς ο υπολογισμός έγινε σε σχετικά μικρό χρονικό διάστημα, παρά το μέγεθος του αριθμού N.

Άσκηση 12 (11.3)

Δεδομένου του δημόσιου κλειδιού $(N, e) = (11413, 19)$, και της υπόθεσης ότι το αρχικό μήνυμα κωδικοποιήθηκε σε ΑΣΚΙΙ χαρακτήρες, ακολουθείται η εξής διαδικασία για την εύρεση του ιδιωτικού κλειδιού και την αποκρυπτογράφηση του μηνύματος:

1. Παραγοντοποίηση του N

Το πρώτο βήμα είναι να παραγοντοποιηθεί ο αριθμός N:

$$N = 11413 = 113 \cdot 101$$

Άρα:

$$p = 113, \quad q = 101$$

2. Υπολογισμός του $\phi(N)$

$$\phi(N) = (p-1)(q-1) = (113-1)(101-1) = 112 \cdot 100 = 11200$$

3. Υπολογισμός του Ιδιωτικού Εκθέτη d

Ο ιδιωτικός εκθέτης d είναι:

$$d \equiv e^{-1} \mod \phi(N)$$

Υπολογίζεται με τον εκτεταμένο αλγόριθμο του Ευκλείδη $extended\_gcd(a, b)$:
Για $e = 19$ και $\phi(N) = 11200$, προκύπτει:

$$d = 3539$$

4. Αποκρυπτογράφηση Μηνύματος

Το κρυπτογραφημένο μήνυμα είναι ένας πίνακας C από αριθμούς:

$$C = [3203, 909, 3143, \ldots, 4443]$$

Η αποκρυπτογράφηση κάθε $block$ c_i γίνεται ως εξής:

$$m_i = c_i^d \mod N$$

Το m_i στη συνέχεια μετατρέπεται σε χαρακτήρα $ASCII$.

5. Συμπεράσματα

Μέσω του ιδιωτικού κλειδιού $(N, d) = (11413, 3539)$ καταφέραμε να αποκρυπτογραφήσουμε το αρχικό μήνυμα. Η χρήση $ASCII$ επιτρέπει την άμεση μετατροπή των αποκρυπτογραφημένων αριθμών σε χαρακτήρες.
Έξοδος: Ιδιωτικό κλειδί: (N, δ) = (11413, 1179)

Αποκρυπτογραφημένο μήνυμα: *welcowe to the real world*

Άσκηση 13 (12.2)

Περιγραφή της επίθεσης

Η επίθεση του $Wiener$ στοχεύει στο να βρει τον ιδιωτικό εκθέτη d ενός RSA ζεύγους (N, e), στην περίπτωση που d είναι ¨μικρός¨, δηλαδή $d < \frac{1}{3}N^{1/4}$. Η μέθοδος βασίζεται στην αναπαράσταση του e/N ως συνεχές κλάσμα και στην εύρεση των συγκλινόντων του, τα οποία παρέχουν υποψήφιες τιμές για τον d.

Υλοποίηση της επίθεσης

Ο ακόλουθος *Python* κώδικας υλοποιεί την επίθεση του $Wiener$:

- $rational\_to\_contfrac(x, y)$: Επιστρέφει την αναπαράσταση συνεχούς κλάσματος του x/y.
- $convergents\_from\_contfrac(frac)$: Επιστρέφει τις συγκλίνουσες του κλάσματος.
- $wiener\_attack(e, N)$: Δοκιμάζει κάθε υποψήφιο (k, d) για να ελέγξει αν ικανοποιεί τη συνθήκη:
$$\phi(N) = \frac{ed - 1}{k}$$
και αν οι ρίζες της εξίσωσης $x^2 - sx + N = 0$ είναι ακέραιες (επαληθεύει ότι το d είναι σωστό).
- $rsa\_decrypt(c, d, N)$: Υπολογίζει το $m = c^d \mod N$.
- $decode\_ascii\_from\_int(m)$: Μετατρέπει τον ακέραιο m πίσω σε $ASCII$ χαρακτήρες.

Αποκρυπτογράφηση Κειμένου

Το κρυπτογράφημα δόθηκε σε κωδικοποίηση $base64$ και αντιστοιχούσε σε ένα σύνολο $RSA\ blocks$. Το κείμενο αποκωδικοποιήθηκε ως εξής:

1. Μετατράπηκε από $base64$ σε $UTF-8 string$.
2. Εξήχθησαν όλοι οι ακέραιοι αριθμοί με χρήση κανονικής έκφρασης.
3. Κάθε $block\ c_i$ αποκρυπτογραφήθηκε με $m_i = c_i^d \mod N$.
4. Κάθε m_i μετατράπηκε σε $bytes$ και στη συνέχεια σε $ASCII$ χαρακτήρες.

Αποτελέσματα

Η επίθεση ολοκληρώθηκε επιτυχώς. Το ιδιωτικό κλειδί που προέκυψε είναι:

$$d = 20881$$

Το τελικό αποκρυπτογραφημένο μήνυμα είναι:

> *Just because you are a character doesn't mean that you have character*

Αυτό επιβεβαιώνει ότι το (N, e) ήταν ευάλωτο στην επίθεση $Wiener$ λόγω μικρής τιμής του d και ότι η ακολουθούμενη διαδικασία αποκρυπτογράφησης ήταν ορθή.

Άσκηση 14 (15.2)

Η μετάβαση από το TLS 1.2 στο TLS 1.3 έφερε σημαντικές αλλαγές, με στόχο τη βελτίωση της ασφάλειας, της απόδοσης και της απλότητας του πρωτοκόλλου. Παρακάτω παρουσιάζονται οι βασικότερες διαφορές, όπως καταγράφονται στο $Textbook$

$Ciphersuites$

Στην έκδοση TLS 1.2 υποστηρίζεται ένας μεγάλος αριθμός από $ciphersuites$ — περίπου 200 διαφορετικές επιλογές. Πολλές από αυτές, όμως, είναι πλέον ξεπερασμένες ή θεωρούνται ανασφαλείς. Για παράδειγμα:

- $TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256$: θεωρείται ακόμα ασφαλής,
- $TLS\_KRB5\_WITH\_3DES\_EDE\_CBC\_MD5$: βασίζεται σε παλιούς και ευάλωτους αλγόριθμους,
- $TLS\_NULL\_WITH\_NULL\_NULL$: δεν προσφέρει καμία απολύτως ασφάλεια.

Αντίθετα, το TLS 1.3 έχει σχεδιαστεί ώστε να περιλαμβάνει μόνο σύγχρονες και ασφαλείς επιλογές. Έτσι, ο αριθμός των διαθέσιμων $ciphersuites$ έχει μειωθεί δραστικά και όλες χρησιμοποιούν ισχυρούς μηχανισμούς, όπως:

- $TLS\_AES\_128\_CCM\_8\_SHA256$, που ενδείκνυται για $embedded$ συσκευές.

Ανταλλαγή Κλειδιών

Ένα βασικό σημείο διαφοροποίησης είναι ο τρόπος με τον οποίο γίνεται η ανταλλαγή κλειδιών. Στο TLS 1.2, αυτή μπορεί να γίνει είτε με RSA είτε με$Diffie-Hellman$ ($Static$ ή $Ephemeral$), ανάλογα με τις δυνατότητες του $client$ και του $server$.

Στο TLS 1.3, η ανταλλαγή βασίζεται αποκλειστικά σε εφήμερα κλειδιά μέσω του $ECDHE$ ($Ephemeral\ Elliptic\ Curve\ Diffie\text{-}Hellman$). Αυτό σημαίνει ότι εξασφαλίζεται πάντα $forwardsecrecy$, δηλαδή ακόμα κι αν κάποια στιγμή παραβιαστεί ένα μακροχρόνιο κλειδί, το περιεχόμενο παλαιότερων επικοινωνιών δεν μπορεί να αποκαλυφθεί.

$Handshake$

Το TLS 1.2 χρησιμοποιεί ένα εκτενές $handshake$, το οποίο περιλαμβάνει αρκετά μηνύματα και φάσεις, όπως: $ClientHello$, $ServerHello$, $Certificate$, $ServerKeyExchange$ κ.ά. Το γεγονός αυτό το καθιστά πιο ευάλωτο σε επιθέσεις (όπως οι $BEAST$, $CRIME$, $Lucky13$, κ.λπ.) και αυξάνει τον χρόνο για την εγκαθίδρυση ασφαλούς σύνδεσης.

Το TLS 1.3 απλοποιεί σημαντικά τη διαδικασία. Το νέο *handshake* είναι μικρότερο σε μήκος, απαιτεί λιγότερα μηνύματα και προσφέρει αυξημένη ασφάλεια εξαρχής. Επιπλέον, επιτρέπει τη χρήση $0-RTT$ ($Zero\ Round\ Trip\ Time$) για ακόμα ταχύτερες συνδέσεις σε επαναλαμβανόμενες επικοινωνίες.

Προεπιλεγμένη Ασφάλεια

Στο TLS 1.2, οι επιλογές παραμέτρων (αλγόριθμοι, *modes*, κ.λπ.) αφήνονται σε μεγάλο βαθμό στον *client* και τον *server*, πράγμα που μπορεί να οδηγήσει σε κακές επιλογές. Η δυνατότητα διαπραγμάτευσης σημαίνει ότι ενδέχεται να επιλεγούν αδύναμοι ή ξεπερασμένοι μηχανισμοί.

Το TLS 1.3 εξαλείφει αυτόν τον κίνδυνο ενσωματώνοντας εξ ορισμού μόνο ασφαλείς επιλογές. Έτσι, ακόμη κι αν κάποιος από τους δύο συμμετέχοντες δεν έχει σωστά ρυθμισμένο σύστημα, η σύνδεση δεν θα γίνει με επισφαλή τρόπο.

Υποστήριξη από Προγράμματα Περιήγησης

Και οι δύο εκδόσεις υποστηρίζονται από όλους τους σύγχρονους *browsers*. Ωστόσο, η TLS 1.2 παραμένει η πιο διαδεδομένη σήμερα, κυρίως λόγω παλαιότερων συστημάτων. Παρ' όλα αυτά, η TLS 1.3 υιοθετείται σταδιακά και αναμένεται να γίνει το νέο πρότυπο, καθώς προσφέρει καλύτερη ασφάλεια και ταχύτερες επιδόσεις.

Συνοψίζοντας

Η έκδοση TLS 1.3 αποτελεί ένα σημαντικό βήμα προόδου σε σχέση με την 1.2. Εξαλείφει πολλές παλιές αδυναμίες, επιταχύνει τη διαδικασία σύνδεσης και ενσωματώνει μοντέρνους και ασφαλείς αλγορίθμους. Όλα αυτά την καθιστούν ιδανική επιλογή για τις απαιτήσεις του σύγχρονου διαδικτύου.

Άσκηση 15 (15.4)

Πρώτα, εισήγαγα το δημόσιο κλειδί του παραλήπτη από το αρχείο $pk.asc$ με την εντολή:

$$gpg --import pk.asc$$

Έφτιαξα το μήνυμα που ήθελα να στείλω σε ένα αρχείο κειμένου ($message.txt$).

Μετά, κρυπτογράφησα το μήνυμα χρησιμοποιώντας το δημόσιο κλειδί του παραλήπτη ($keyID\ 0xEB1185F82713D6DF$) με την εντολή:

$$gpg --encrypt --armor --recipient\ 0xEB1185F82713D6DF\ message.txt$$

Αυτό δημιούργησε το αρχείο $message.txt.asc$, που είναι το κρυπτογραφημένο μήνυμα.

Τέλος, έστειλα το αρχείο αυτό μέσω εμαιλ στο *drazioti@csd.auth.gr* μαζι με το δημόσιο κλειδί μου και ελαβα ως απαντηση κρυπτογραφημενο κειμενο που αποκρυποτγράφησα και έλεγε : $2db8d27ba23ec57547784e2cbb3f7ae4$

Σημείωση

Για την ευκολία της συγγραφής και την αισθητική παρουσίαση της αναφοράς, χρησιμοποιήθηκε σε μικρό βαθμό η βοήθεια του *ChatGPT*, κυρίως:

- για τη μορφοποίηση του αρχείου LaTeX ,
- για τη βελτίωση κάποιων κομματιών κώδικα, όπως οι περιττοί και χρονοβόροι έλεγχοι τα *labels* κλπ.
- για εύρεση επιπρόσθετων πληροφοριών

Η κατανόηση και η επίλυση των ασκήσεων έγιναν από εμένα και η χρήση του εργαλείου έγινε απλώς για διευκόλυνση και όχι για αντικατάσταση της διαδικασίας μάθησης.