Project II
Κρυπτογραφία Δημοσίου Κλειδιού

Ασημάκης Κύδρος
ΑΕΜ: 3881
asimakis@csd.auth.gr

10 Ιουνίου 2023

Άσκηση 1:
Εφαρμόστε το πρωτόκολλο ανταλλαγής Diffie-Hellman για $g = 3, p = 101, a = 77, b = 91$. Υπολογίστε, δηλαδή, το κοινό κλειδί. Υλοποιείστε τον αλγόριθμο γρήγορης ύψωσης σε δύναμη για να το πετύχετε.

Η Alice και ο Bob έχουν συμφωνήσει στους αριθμούς $g = 3, p = 101$. Επιλέγουν μυστικά κλειδιά 77 και 91 αντιστοίχως και δημοσιεύουν τα $3^{77}, 3^{91}$ αντιστοίχως. Επομένως και οι δύο καταλήγουν στο κοινό κλειδί

$$K = (3^{77})^{91} \ \ mod \ \ 101 = (3^{91})^{77} \ \ mod \ \ 101 \Rightarrow$$
$$K = 66$$

Το παραπάνω σκηνικό υλοποιείται στο task 1.py. Για την γρήγορη ύψωση σε δύναμη χρησιμοποιείται η συνάρτηση **mod_pow** που ορίζεται στο myfuncs.py και ακολουθεί πιστά τον ψευδοκώδικα των σημειώσεων.

Άσκηση 2:
Υλοποιείστε τον αλγόριθμο της γρήγορης ύψωσης σε δύναμη και χρησιμοποιείστε τον για να υπολογίσετε το $5^{77} \ mod \ 19$.

Όπως και στην προηγούμενη άσκηση, ο αλγόριθμος ορίζεται στο myfuncs.py. Τρέχοντας το τεστ του task 2.py βλέπουμε πως

$$5^{77} \ mod \ 19 = 9$$

Η αποτελεσματικότητα αυτού του αλγορίθμου υπογραμμίζεται με την αστραπιαία ταχύτητα που υπολογίζει τόσο μεγάλα (και ακόμα μεγαλύτερα) εκθετικά.

Άσκηση 3:
Να αποδείξετε ότι ο n–οστός πρώτος p_n ικανοποιεί την ανισότητα $p_n < 2^{2^n}$

Γνωρίζουμε πως

$$p_{n+j} < p_1 p_2 \dots p_n + 1 \text{ για } j > 0$$

άρα

$$p_{n+1} \leq p_1 p_2 \dots p_n + 1 \tag{1}$$

Θα αποδείξουμε το ζητούμενο με επαγωγή. Παρατηρούμε πως για $n = 1$ ισχύει:

$$p_1 = 2 < 2^{2^1} \Rightarrow^{?} 2 < 4 \ \checkmark$$

Έστω ότι ισχύει για τυχαίο n. Θα αποδείξουμε ότι ισχύει και για $n+1$. Εφαρμόζοντας την υπόθεση στην (1) έχουμε:

$$\begin{aligned} &p_{n+1} \leq p_1 p_2 \dots p_n + 1 \Rightarrow \\ &p_{n+1} \leq 2^{2^1} 2^{2^2} \dots 2^{2^n} + 1 \Rightarrow \\ &p_{n+1} \leq 2^{2+4+\dots+2^n} + 1 \Rightarrow \\ &p_{n+1} \leq 2^{2(1+2+\dots+2^{n-1})} + 1 \end{aligned}$$

Είναι γνωστό πως

$$1 + 2 + 4 + \dots + 2^{n-1} = 2^n - 1$$

άρα

$$\begin{aligned} &p_{n+1} \leq 2^{2(2^n-1)} + 1 \Rightarrow \\ &p_{n+1} \leq 2^{2^{n+1}-2} + 1 \end{aligned}$$

Επίσης παρατηρούμε πως

$$1 < 3 \cdot 2^{2^{n+1}-2} \ \ \forall n > 0$$

επομένως

$$p_{n+1} < 4 \cdot 2^{2^{n+1}-2} \Rightarrow$$
$$p_{n+1} < 2^{2^{n+1}} \quad \checkmark$$

Άρα τελικά όντως ισχύει $p_n < 2^{2^n} \quad \forall n > 0$

Άσκηση 4:
Έστω a, b θετικοί ακέραιοι. Αν $gcd(a,b) = 1$, **τότε δείξτε ότι:**

1. $gcd(ac, b) = gcd(c, b) \ \ \forall c \in \mathbb{Z}$
2. $gcd(a+b, a-b) \in \{1, 2\}$ **και συγκεκριμένα ότι είναι 2 αν a, b περιττοί.**
3. $gcd(2^a - 1, 2^b - 1) = 1$
4. $gcd(M_p, M_q) = 1$ **όπου** $M_p = 2^p - 1$ **Mersenne πρώτοι** $(p \neq q)$**.**

1. Έστω $gcd(ac, b) = d_1$. Άρα ισχύει ότι

$$d_1|ac, \ \ d_1|b$$

Εφόσον $gcd(a, b) = 1$, πηγάζουμε από τα παραπάνω πως

$$d_1 \not| a, d_1|c$$

Έστω τώρα ότι το $d_1 \neq gcd(c, b)$, άρα

$$gcd(c, b) = d_2 > d_1$$

άρα έχουμε πως

$$d_2|c, d_2|b \Rightarrow d_2|ac + b$$

(ο ΚΔ δύο ακεραίων διαιρεί και κάθε γραμμικό συνδυασμό τους). Εφόσον όμως $d_2 > d_1$ (εξορισμού) τότε έχουμε πως τελικά

$$gcd(ac, b) = d_2$$

και άρα

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

2. Έστω $gcd(a+b, a-b) = d$. Εφόσον το d διαιρεί τα $a+b, a-b$, θα διαιρεί και το άθροισμα και τη διαφορά τους. Συγκεκριμένα, έστω

$$a+b = c_1 d, \quad a-b = c_2 d$$

για αυθαίρετα c_1, c_2.

Τότε:

$$(a+b)+(a-b) = c_1 d + c_2 d \Rightarrow 2a = (c_1+c_2)d$$

$$(a+b)-(a-b) = c_1 d - c_2 d \Rightarrow 2b = (c_1-c_2)d$$

Άρα ισχύει

$$d|2a, \quad d|2b$$

και επειδή $gcd(a,b) = 1$ πρέπει

$$d|2 \Rightarrow d = \{1, 2\}$$

Έστω ότι τώρα a, b είναι επιπλέον περιττοί, μπορούμε δηλαδή να τους γράψουμε

$$a = 2k_1 + 1, \quad b = 2k_2 + 1, \quad k_1, k_2 \in \mathbb{Z}$$

Άρα

$$a+b = 2k_1+1+2k_2+1 = 2k_1+2k_2+2 = 2(k_1+k_2+1) = 2m$$

$$a-b = 2k_1+1-2k_2-1 = 2k_1-2k_2 = 2(k_1-k_2) = 2n, \quad m, n \in \mathbb{Z}$$

Είναι και τα δύο άρτιοι, άρα $2|(a+b), \ \ 2|(a-b)$ και επομένως

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

3. Έστω $gcd(2^a-1, 2^b-1) = d$, και επειδή $gcd(a,b) = 1$ τότε από την ταυτότητα Bezout:

$$\exists x, y \in \mathbb{Z} \ni ax + by = 1$$

επίσης

$$2^a - 1 = c_1 d \Rightarrow 2^a = c_1 d + 1$$
$$2^b - 1 = c_2 d \Rightarrow 2^b = c_2 d + 1$$

για αυθαίρετα c_1, c_2.

Τότε:

$$2^1 = 2^{ax+by} = 2^{ax}2^{by} = (c_1 d + 1)^x (c_2 d + 1)^y$$

Χρησιμοποιούμε την **διωνυμική ταυτότητα** για να υπολογίσουμε τις προκύπτουσες δυνάμεις:

$$2^1 = \sum_{k=0}^{x} \binom{x}{k} (c_1 d)^k \cdot \sum_{k=0}^{y} \binom{y}{k} (c_2 d)^k \Rightarrow$$
$$2^1 = (\sum_{k=1}^{x} \binom{x}{k} (c_1 d)^k + 1)(\sum_{k=1}^{y} \binom{y}{k} (c_2 d)^k + 1) \Rightarrow$$
$$2^1 = \sum_{k=1}^{x} \binom{x}{k} (c_1 d)^k \cdot \sum_{k=1}^{y} \binom{y}{k} (c_2 d)^k +$$
$$\sum_{k=1}^{x} \binom{x}{k} (c_1 d)^k + \sum_{k=1}^{y} \binom{y}{k} (c_2 d)^k + 1$$

Κάθε άθροισμα είναι ένας γραμμικός συνδυασμός των δυνάμεων του d ξεκινώντας από την πρώτη δύναμη. Άρα, στην τελική σχέση, ο πρώτος όρος θα έχει κοινό παράγοντα το d^2 και οι άλλοι δύο το d.

Επομένως, για ακεραίους m, n, l, k, g καταλήγουμε στο

$$2^1 = d^2 \cdot m + d \cdot n + d \cdot l + 1 = d(k + n + l) + 1 \Rightarrow$$

$$1 = d \cdot g$$

Άρα ισχύει $d|1$ και άρα το $d = 1$ (d, g θετικά).

4. Εφόσον $p \neq q$ τότε $gcd(p, q) = 1$ και άρα ισχύει το ζητούμενο με εφαρμογή του από πάνω.

Άσκηση 5:
Έστω $a, b, c \in \mathbb{Z}$ και $\delta = a^2 - 4bc^2 \neq 0$. Δείξτε ότι το $gcd(\delta, 4c^2)$ είναι τετράγωνο.

Ισχύει γενικά πως

$$gcd(a^n, b^n) = [gcd(a, b)]^n, \quad \forall a, b \in \mathbb{Z}$$

Αυτό φαίνεται από τις πρωτογενείς αναλύσεις των a, b:

$$a = p_1^{e_{11}} p_2^{e_{12}} p_3^{e_{13}} \dots$$
$$b = p_1^{e_{21}} p_2^{e_{22}} p_3^{e_{23}} \dots$$

$$\Rightarrow gcd(a, b) = p_1^{min(e_{11}, e_{21})} p_2^{min(e_{12}, e_{22})} p_3^{min(e_{13}, e_{23})} \dots \quad (2)$$

Υψώνοντας σε κοινή δύναμη έχουμε:

$$a^n = (p_1^{e_{11}} p_2^{e_{12}} p_3^{e_{13}} \dots)^n = p_1^{ne_{11}} p_2^{ne_{12}} p_3^{ne_{13}} \dots$$
$$b^n = (p_1^{e_{21}} p_2^{e_{22}} p_3^{e_{23}} \dots)^n = p_1^{ne_{21}} p_2^{ne_{22}} p_3^{ne_{23}} \dots$$

$$\begin{aligned} \Rightarrow gcd(a^n, b^n) &= p_1^{min(ne_{11}, ne_{21})} p_2^{min(ne_{12}, ne_{22})} p_3^{min(ne_{13}, ne_{23})} \dots \\ &= p_1^{n \cdot min(e_{11}, e_{21})} p_2^{n \cdot min(e_{12}, e_{22})} p_3^{n \cdot min(e_{13}, e_{23})} \dots \\ &= (p_1^{min(e_{11}, e_{21})} p_2^{min(e_{12}, e_{22})} p_3^{min(e_{13}, e_{23})} \dots)^n \end{aligned}$$

$$(2) \Rightarrow gcd(a^n, b^n) = [gcd(a, b)]^n \ \checkmark$$

Είναι επίσης τετριμμένο πως

$$gcd(a + mb, b) = gcd(a, b), \quad \forall a, b, m \in \mathbb{Z}$$

Γνωρίζοντας τα παραπάνω, έχουμε:

$$gcd(a, 2c) = d \Rightarrow gcd(a^2, 4c^2) = d^2 \wedge d^2 | a^2 - 4bc^2$$
$$\Rightarrow gcd(\delta, 4c^2) = d^2 \ \square$$

Άσκηση 6:
Επαληθεύστε πειραματικά ότι για όλους τους περιττούς ακεραίους $< 2^{20}$ ισχύει

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

όπου γ η σταθερά Euler.

Αρχικά αναδιαμορφώνουμε την ανισότητα ώστε να μεγιστοποιήσουμε την ταχύτητα υπολογισμού και, κατ΄ επέκταση, επαλήθευσής της:

$$\begin{aligned}
\frac{\sigma(n)}{n} &< \frac{e^{\gamma}}{2} ln(ln(n)) + \frac{0.74}{ln(ln(n))} \\
\xrightarrow{\cdot nln(ln(n))} \sigma(n) ln(ln(n)) &< \frac{e^{\gamma}}{2} n ln^2(ln(n)) + 0.74n \\
\Rightarrow \sigma(n) ln(ln(n)) &< n(\frac{e^{\gamma}}{2} ln^2(ln(n)) + 0.74) \\
\xrightarrow[\mathcal{L}=ln(ln(n))]{\mathcal{C}_e=\frac{e^{\gamma}}{2}} \sigma(n)\mathcal{L} &< n(\mathcal{C}_e \mathcal{L}^2 + 0.74)
\end{aligned}$$

Τώρα μπορούμε να δοκιμάσουμε την ανισότητα στο δεδομένο πεδίο ορισμού. Τρέχοντας το task 6.py αποβραδίς ανακαλύπτουμε πως η ανισότητα όντως ισχύει για κάθε περιττό ακέραιο $< 2^{20}$.

Άσκηση 7:
Αποδείξτε ότι οι αριθμοί

$$9999109081, 655313092675200603148176 1$$

είναι Carmichael. Μπορείτε να βρείτε κάποιον μεγαλύτερο;

Αριθμοί Carmichael λέγονται οι ψευδοπρώτοι αριθμοί, δηλαδή εκείνοι οι σύνθετοι περιττοί που περνούν το τεστ του Fermat.
Μπορούμε να εκφράσουμε το παραπάνω με μαθηματικά ως

$$n: \quad \text{Carmichael}$$
$$\iff$$
$$\forall a \in (1, n) \cap \mathbb{Z} \ni gcd(a, n) = 1 \implies a^{n-1} \equiv 1(\text{mod n})$$

Υλοποιούμε επομένως πρόγραμμα που πιστοποιεί την ιδιότητα Carmichael ελέγχοντας το παραπάνω $\forall a \in [2, n-1]$. Ο έλεγχος του task 7.py επαληθεύει ότι οι δωσμένοι αριθμοί είναι όντως Carmichael.

Παρόλο που δεν έχει αποδειχθεί, οι Carmichael φαίνονται αρκετά σπάνιοι, και δεν υπάρχει κάποιος τύπος ή μέθοδος για την εύρεσή τους. Για να βρούμε κάποιον μεγαλύτερο, επομένως, μπορούμε μόνο να ψάξουμε τυφλά πέρα από τον 655313092675200603148176 1 για περιττούς που ικανοποιούν το παραπάνω κριτήριο.
Η Monte Carlo εκδοχή του προηγουμένου αλγορίθμου δίνει τον

$$655313092675200603148177 7$$

ως τον επόμενο Carmichael, προφανώς με κάποια πιθανότητα.

Άσκηση 8:
Ελέγξτε αν το ζεύγος $835335 \cdot 2^{39014} \pm 1$ **ικανοποιεί το τεστ του Fermat.**

Στο task 8.py υλοποιείται το τεστ του Fermat, και ακολουθεί πιστά την θεωρία. Τρέχοντας το τεστ για το ζεύγος βλέπουμε πως είναι και οι δύο ψευδοπρώτοι.

Μαζί παρέχεται και η Monte Carlo εκδοχή του τεστ του Fermat. Ενώ το αποτέλεσμα αυτής θα είναι πολύ πιο έμπιστο, εφόσον τρέχουμε το απλό τεστ για πολλούς τυχαίους αριθμούς, η μέθοδος είναι κλάσεις πιο αργή.

Άσκηση 9:
Υλοποιείστε τον αλγόριθμο της δοκιμαστικής διαίρεσης και χρησιμοποιείστε τον για να παραγοντοποιείσετε τους $n_1 = 2^{62}-1, n_2 = 2^{102}-1$.

Η δοκιμαστική διαίρεση υλοποιείται στο myfuncs.py, για λόγους επαναχρησιμοποίησης σε επόμενες ασκήσεις. Η υλοποίησή του δεν παρεκκλίνει από τον ψευδοκώδικα των σημειώσεων.

Η αποτελεσματικότητά του γίνεται φανερή στην παραγοντοποίηση των δωσμένων αριθμών, καθώς επιστρέφει

$$[3, 715827883, 2147483647]_{n_1}$$
$$[3, 3, 7, 103, 307, 2143, 2857, 6529, 11119, 43691, 131071]_{n_2}$$

με ικανοποιητική ταχύτητα.

Άσκηση 10:
Υλοποιήστε τον αλγόριθμο του Lehman για παραγοντοποίηση και βρείτε το ποσοστό επιτυχίας για 1000 επαναλήψεις του παρακάτω πειράματος:
Έστω τυχαίος περιττός 100-bit ακέραιος. Αν ο αλγόριθμος του Lehman τον παραγοντοποιήσει μέσα σε 10 δευτερόλεπτα, θεωρούμε επιτυχία.

Υλοποιούμε τον αλγόριθμο του Lehman βασιζόμενοι στον ψευδοκώδικα της θεωρίας. Πρόκειται για υπολογιστικά βαρύ αλγόριθμο, οπότε αξίζει να κάνουμε όσες τροποποιήσεις μπορούμε για να τον επιταχύνουμε.

Από άποψη υλοποίησης, πρέπει να ελαχιστοποιήσουμε τις απαιτούμενες ρίζες και να ανακυκλώσουμε όσες το δυνατόν περισσότερες τιμές γίνεται. Ας δούμε το πεδίο ορισμού του α:

$$\left\lceil \sqrt{4kn} \right\rceil \leq \alpha \leq \left\lfloor \sqrt{4kn} + \frac{n^{1/6}}{4\sqrt{k}} \right\rfloor$$

Έχουμε 3 ρίζες να υπολογίσουμε μόνο γι αυτό το κομμάτι. Όμως, το $4\sqrt{k}$ μοιάζει με το $\sqrt{4kn}$. Αν το φέρουμε στην ίδια μορφή, έχουμε:

$$\left\lceil \sqrt{4kn} \right\rceil \leq \alpha \leq \left\lfloor \sqrt{4kn} + \frac{n^{1/6}n^{1/2}}{4\sqrt{k}\sqrt{n}} \right\rfloor \Rightarrow$$
$$\left\lceil \sqrt{4kn} \right\rceil \leq \alpha \leq \left\lfloor \sqrt{4kn} + \frac{n^{2/3}}{4\sqrt{kn}} \right\rfloor \Rightarrow$$
$$\left\lceil \sqrt{4kn} \right\rceil \leq \alpha \leq \left\lfloor \sqrt{4kn} + \frac{(\sqrt[3]{n})^2}{2\sqrt{4kn}} \right\rfloor$$

και κάνοντας ομώνυμα κλάσματα έχουμε

$$\left\lceil \sqrt{4kn} \right\rceil \leq \alpha \leq \left\lfloor \frac{2 \cdot 4kn + (\sqrt[3]{n})^2}{2\sqrt{4kn}} \right\rfloor \Rightarrow$$
$$\left\lceil \sqrt{4kn} \right\rceil \leq \alpha \leq \left\lfloor \frac{4kn + 0.5(\sqrt[3]{n})^2}{\sqrt{4kn}} \right\rfloor$$

Καταφέραμε, λοιπόν, να διώξουμε την έκτη ρίζα του n, να ανακυκλώσουμε τις τιμές $\sqrt{4kn}$, $\sqrt[3]{n}$ και $4kn$, που επανεμφανίζονται, και να μειώσουμε το πλήθος των απαιτούμενων ριζών προς υπολογισμό σε *όλο τον κώδικα* σε 3.

Όσο αφορά την εκτέλεση, μας νοιάζει μόνο αν ο αλγόριθμος τερματίζει σε 10 δευτερόλεπτα, και όχι το αποτέλεσμά του. Επομένως, αν ξεπεραστεί αυτό το χρονικό περιθώριο, δεν έχει νόημα να περιμένουμε και μπορούμε να το τερματίσουμε νωρίς. Γι αυτό το λόγο τρέχουμε το κάθε τεστ σε ένα νήμα και το 'σκοτώνουμε' αν περάσουν 10 δεύτερα.

Τα παραπάνω υλοποιούνται στο task 10.py. Για τις 1000 επαναλήψεις του πειράματος ο αλγόριθμος σκοράρει $\mathbf{18.9\%}$. Λαμβάνοντας υπόψιν το γεγονός πως 10 δευτερόλεπτα είναι άκρως μη αντιπροσωπευτικό ταβάνι για αξιολόγηση αλγορίθμου παραγοντοποίησης, το αποτέλεσμα αυτό είναι αρκετά καλό.

Διπλασιάζοντας το ταβάνι χρόνου, το ποσοστό επιτυχίας ανεβαίνει στο $\mathbf{20.6\%}$. Βλέπουμε πως, ακόμα και για τις ακραίες μας απαιτήσεις χρόνου, ο αλγόριθμος του lehman καταφέρνει να παραγοντοποιήσει το 1/5 των δειγμάτων.

Άσκηση 11:
Χρησιμοποιείστε τον αλγόριθμο του Pollard σε συνδυασμό με τον αλγόριθμο του Floyd για να παραγοντοποιείσετε τον $N = 2^{257} - 1$. Υποθέστε $F(X) = (X^2 + 1)$ mod N με έλεγχο

$$1 < gcd(|X - Y|, N) < N$$

Ο αλγόριθμος υλοποιείται στο task 11.py. Αποτελεί απλή παραλλαγή του παραδοσιακού αλγορίθμου του Floyd με τις αλλαγές που μας δώθηκαν. Ο αλγόριθμος αυτός χρησιμοποιεί το γεγονός ότι τα collisions σε μεγάλο δακτύλιο δεν είναι και τόσο συχνά, για να 'μαντέψει' παράγοντες του αριθμού που δεν είναι προφανείς. Πράγματι, όσο πλησιάζουμε σε διένεξη, προσεγγίζουμε έναν trivial factor. Αν $x = y$, τότε $gcd(|x - y|, N) = N$, ενώ αν $|x - y| = 1$ τότε $gcd(|x - y|, N) = 1$.

Η μέθοδος είναι αρκετά αποτελεσματική. Έπειτα από ανεκτό χρονικό διάστημα, επιστρέφει τον (μικρό) παράγοντα

$$\approx 5.8688966023081812065445 \cdot 10^{67}$$

Το ακριβές νούμερο μπορεί να βρεθεί στο προαναφερόμενο αρχείο.

Άσκηση 12:
Με RSA public key $(N, e) = (11413, 19)$
αποκρυπτογραφήστε το μήνυμα

$$\mathbf{C = (3203,\ 909,\ 3143,\ 5255,\ 5343,\ 3203,\ 909,\ 9958,\ 5278,\ 5343,\ 9958,\ 5278,\ 4674,\ 909,\ 9958,\ 792,\ 909,\ 4132,\ 3143,\ 9958,\ 3203,\ 5343,\ 792,\ 3143,\ 4443)}$$

Εφόσον γνωρίζουμε το N μπορούμε βρούμε την παραγοντοποίησή του. Το πράττουμε αυτό χρησιμοποιώντας τον αλγόριθμο της δοκιμαστικής διαίρεσης από πριν.

Έχοντας τους παράγοντες του N, υπολογίζουμε το $\phi(N)$, με βάση τον τύπο

$$\phi(N) = N(1 - \frac{1}{p})(1 - \frac{1}{q})$$

όπου p, q οι παράγοντες του N (εξ΄ ορισμού 2 μεγάλοι διαφορετικοί πρώτοι αριθμοί).

Τώρα έχουμε όλα τα εργαλεία για να κάνουμε brute force το μυστικό κλειδί d, δοκιμάζοντας την παρακάτω ισότητα:

$$e \cdot d \equiv 1 \pmod{\phi(N)} \quad \forall d \in [1, \phi(N))$$

Καταλήγουμε στο $d = 1179$. Πλέον μπορούμε να αποκρυπτογραφήσουμε το C, το οποίο κατά τα γνωστά γίνεται υπολογίζοντας το

$$m_i = c_i^d \pmod{N}$$

για κάθε c_i του C. Προφανώς χρησιμοποιούμε τον αλγόριθμο της γρήγορης ύψωσης σε δύναμη για να το καταφέρουμε.

Τελικά καταλήγουμε στο παρακάτω αποκρυπτογραφημένο μήνυμα:

welcowe to the real world

το οποίο είναι (typo-d) quote του Morpheus από το *Matrix*.

Όλη η παραπάνω διαδικασία υλοποιείται στο task 12.py.

Άσκηση 13:
Αποκρυπτογραφήστε το μήνυμα που βρίσκεται στο github, που κρυπτογραφήθηκε με textbook-RSA με public key

$$(N, e) = (194749497518847283, 50736902528669041)$$

και κατόπιν κωδικοποιήθηκε. Για να βρείτε το secret key d εφαρμόστε την επίθεση του Wiener.

Η αποκωδικοποίηση του string που υπάρχει στο github από Base64 δίνει ένα μήνυμα C σαν και αυτό του δεδομένου της άσκησης 12.

Εφαρμόζουμε επίθεση Wiener για να βρούμε το μυστικό κλειδί d. Για την υλοποίησή του, η οποία βρίσκεται στο task 13.py, χρειάστηκε να οριστούν οι συναρτήσεις **continued_fraction** και **convergent_ratio**, που υπολογίζουν συνεχές κλάσμα και τον λόγο συκλίνοντος κλάσματος, αντιστοίχως.

Από εκεί και πέρα η υλοποίηση της επίθεσης ακολουθεί την θεωρία. Έχοντας το d, αποκαλύπτουμε ότι το μήνυμα είναι

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

το οποίο είναι quote του Winston Wolf από την ταινία *Pulp Fiction*.

Άσκηση 14:
Έστω το κλειδί μιας naive ψηφιακής RSA υπογραφής $(N, e) = (899, 839)$. Αν η υπογραφή του μηνύματος $m = 3$ είναι $s = 301$, επαληθεύστε αν η υπογραφή είναι σωστή.

Μια ψηφιακή υπογραφή RSA κρίνεται σωστή/έγκυρη όταν ισχύει

$$m = s^e \pmod{N}$$

Αντικαθιστώντας για τις γνωστές σταθερές βλέπουμε πως

$$3 =^? 301^{839} \pmod{899} \Rightarrow$$
$$3 =^? 675 \text{ ✗}$$

Άρα η υπογραφή δεν είναι σωστή.

Το παραπάνω επαληθεύεται στο task 14.py. Για την πιστοποίηση χρησιμοποιείται ο αλγόριθμος της γρήγορης ύψωσης σε δύναμη.

Άσκηση 15:
Στείλτε ένα κρυπτογραφημένο μήνυμα στον ιδιοκτήτη του δημοσίου κλειδιού με αναγνωριστικό 0xEB1185F82713D6DF.

Έχοντας εγκατεστημένο το GPG στον υπολογιστή μας, παράγουμε κλειδιά μέσω της γραμμής εντολών επικαλώντας το

gpg –full-gen-key

Συμπληρώνουμε κατά τα γνωστά και πλέον έχουμε το δημόσιο κλειδί μας.

Μετέπειτα εισάγουμε το δημόσιο κλειδί του συγγραφέα, που αντιστοιχίζεται στο δεδομένο αναγνωριστικό. Αυτό γίνεται επικαλώντας

gpg –import

όπου επικολλάμε ολόκληρο το κλειδί και πατάμε enter, ctrl + z και enter.

Αφού εισάγουμε το δημόσιο κλειδί του παραλήπτη, προχωράμε στην κρυπτογράφηση. Η διαδικασία ξεκινά επικαλώντας

gpg -a –encrypt –recipient (recipient email)

και δίνοντας το μήνυμα.

Εδώ κρυπτογραφήθηκε το εξής μήνυμα:

A mortal, Frodo, who keeps one of the Great Rings does not die; but he does not grow or obtain more life. He merely continues, until at last every minute is a weariness. And if he often uses the Ring to make himself invisible, he fades; he becomes in the end invisible permanently, and walks in the twilight under the eye of the dark power that rules the Rings.

το οποίο είναι quote του Gandalf από το *Lord of the Rings*.

Στέλνουμε το κρυπτόγραμμα και το δημόσιο κλειδί μας στον παραλήπτη και μας στέλνει αυτός πίσω ένα δικό του κρυπτόγραμμα. Το αποκρυπτογραφούμε επικαλώντας

gpg –decrypt

και δίνοντας το κρυπτόγραμμα με τον προαναφερώμενο τρόπο. Ανακαλύπτουμε, έτσι, το πιστοποιητικό-hash:

23c5877aef9705c4d17b88cd59f81f5d

Άσκηση Bonus:
Υλοποιείστε πρόγραμμα που επιστρέφει ασφαλείς πρώτους μήκους 2048 bits σε περίπου 3 λεπτά.

Επιχειρούμε το ζητούμενο έμμεσα αναζητώντας Sophie Germain πρώτους μήκους 2047 bits. Ο γενικός αλγόριθμος πιστοποίησης Sophie Germain prime και άρα safe prime είναι:

Data: n : desired bit length
Result: $p \ni (p, 2p+1)$ both prime
$p \leftarrow$ random integer of n bits;
while *True* **do**
  if p *and* $2p+1$ *are both prime* **then**
    return p;
  else
    $p \leftarrow p+2$;
  end
end

Algorithm 1: Safe prime generator

Άρα σκοπός μας είναι να κάνουμε τον έλεγχο **is_prime** όσο πιο γρήγορο γίνεται. Το πιο γρήγορο primality test βασίζεται στο τεστ των Miller-Rabin:

Data: n : integer to be tested
Result: $Composite \vee Probable\ Prime$
$a \leftarrow$ random integer in $(2, n-1)$;
Find s,d such that $d \cdot 2^s = n - 1$;
$b \leftarrow a^d \bmod n$;
if $b \equiv \pm 1 (mod\ n)$ **then**
  return $Probable\ Prime$;
end
repeat $s - 1$ **times**
  $b \leftarrow b^2 \bmod n$;
  if $b \equiv 1 (mod\ n)$ **then**
    return $Composite$;
  else if $b = -1 (mod\ n)$ **then**
    return $Probable\ Prime$;
  end
end
return $Composite$

Algorithm 2: Miller-Rabin compositeness test

το οποίο, επειδή είναι πιθανοτικός αλγόριθμος, χρειάζεται να τρέξει πολλαπλές φορές για το ίδιο νούμερο για να γίνει η κρίση $Probable\ Prime$ έμπιστη.

Μπορούμε να βελτιώσουμε την απόδοση αποκλείωντας κάποιους ακεραίους πρωτού τρέξουμε καν Miller-Rabin. Καθώς οι περισσότερες περιπτώσεις έχουν κάποιο μικρό πρώτο ως διαιρέτη, αξίζει να εφαρμόσουμε **trial division** κάθε n για να αποκλείσουμε τις τετριμμένες περιπτώσεις νωρίς.

Βρίσκουμε δυναμικά τους μικρούς πρώτους με το Κόσκινο του Ερατοσθένη. Καθώς ψάχνουμε ασφαλείς πρώτους, μπορούμε να κάνουμε μια περαιτέρω βελτίωση και να εφαρμόσουμε *Combined Sieve* στην trial division:

Data: $n:$ integer to be tested
Result: $Composite \vee Probable\ Prime$
Find small primes using the sieve of Eratosthenes;
forall *small primes* s **do**
  if $n \equiv 0 \vee \frac{s-1}{2}(mod\ s)$ **then**
    return $Composite$;
  end
end
return $Probable\_Prime$

Algorithm 3: Trial Division with combined sieve

Άρα η is_prime γίνεται τελικά:

Data: $n:$ integer to be tested
Result: $True \vee False$
run Trial Division with combined sieve;
if $Composite$ **then**
  return $False$
end
run Miller-Rabin at least 4 times;
if *at least one finds* $Composite$ **then**
  return $False$
else
  return $True$
end

Algorithm 4: Fast primality test

Μπορούμε να κάνουμε μια τελευταία βελτίωση στον αρχικό αλγόριθμο. Ισχύει πως

$$\frac{p-1}{2} \text{ prime} \wedge 2^{p-1} \equiv 1 (\text{mod } p) \rightarrow p \text{ prime}$$

άρα αντί να τρέχουμε την is_prime και στο $2p+1$ μπορούμε να κάνουμε ένα απλό Fermat τεστ σ΄ αυτό, με βάση 2.

Όλα τα παραπάνω υλοποιούνται σε python στο safeprime-gen.py. Μέσα του βρίσκονται και πηγές/αποδείξεις. Το test.py τρέχει την γεννήτρια 100 φορές και επιστρέφει τα παρακάτω αποτελέσματα:

```
C:\Users\User\PycharmProjects\cryptobonusSP\
Results for 100 5-minute tests:
    >>> Hit ratio: 65/100
    >>> Overall success: 65.00%
    >>> Ideal hits: 49/100 (~49.00%)
    >>> False positives: 0/100 (~0.00%)

Process finished with exit code 0
```

Ideal hits είναι τα τεστ που όντως τελείωνουν μέσα σε 3 λεπτά.

Η χρήση της **primitive.primitive.isprime** γίνεται καθαρά για λόγους επαλήθευσης και δεν χρησιμοποιείται πουθενά για την βελτίωση του κώδικα.

(Υ.Γ. Δεν περιμένω πολλές μονάδες καθώς δεν έφτασα τον ζητούμενο μέσο όρο χρόνου. Αν στείλετε πάλι feedback θέλω σας παρακαλώ να μου πείτε τι χάνω από άποψη θεωρητικών βελτιώσεων γιατί έψαξα και δεν βρήκα τίποτα παραπάνω πέρα από παραλληλοποίηση την οποία προσπάθησα και απέτυχα παταγωδώς).