Project I
Συμμετρική Κρυπτογραφία

Υφαντίδης Δημήτριος (AEM: 3938)

10 Μαΐου 2023

Εισαγωγή

Το παρόν αποτελεί την αναφορά του πρώτου μέρους της εργασίας στο μάθημα "Θεμελιώσεις Κρυπτογραφίας".

Η αναφορά είναι γραμμένη σε LATEX και μεταγλωττίστηκε από τον **MiKTeX** compiler (ver: One MiKTeX Utility 1.6 - MiKTeX 23.1).

Οι υλοποιήσεις των ασκήσεων έγιναν σε γλώσσα **Python (v 3.10.6)**. Για κάθε άσκηση που ζητάει υλοποίηση σε κώδικα, υπάρχει ο αντίστοιχος φάκελος:

proj_1_crypto_3938/κώδικας/ex*

Όπου βρίσκεται ένα μοναδικό python script, **ex*.py** που περιέχει την υλοποίηση και ενδεχομένως κάποια ".txt" αρχεία ή άλλους πόρους για το πρόγραμμα.

Σχεδόν όλοι οι μαθηματικοί συμβολισμοί που εμφανίζονται στο κείμενο είναι κοινώς αποδεκτοί. Εξαίρεση αποτελούν οι ακόλουθοι, που διευκρινίζονται:

- **x mod y** : Το υπόλοιπο της ακέραιας διαίρεσης του x με το y (όχι κλάση ισοδυναμίας).
- $\mathbb{N}$: Το σύνολο των φυσικών αριθμών **συμπεριλαμβάνοντας** το 0, δηλ. [0, 1, 2, ...].
- $\mathbb{N}^*$: Το σύνολο των φυσικών αριθμών **χωρίς** το 0, δηλ. [1, 2, 3, ...].

Άσκηση 1 (2.2)

Έχουμε:

- $g(x) = x^2 + 3x + 1,\ x \in R$
- $f(x) = x^5 + 3x^3 + 7x^2 + 3x^4 + 5x + 4,\ x \in R$
- x_0 κάποια ρίζα της $g \quad \Rightarrow \quad g(x_0) = 0$

Υπολογίζουμε την $f(x_0)$:

$$\begin{aligned}
f(x_0) &= x_0^5 + 3x_0^3 + 7x_0^2 + 3x_0^4 + 5x_0 + 4 \\
&= x_0^5 + 3x_0^4 + 3x_0^3 + 7x_0^2 + 5x_0 + 4 \\
&= x_0^5 + 3x_0^4 + x_0^3 + 2x_0^3 + 7x_0^2 + 5x_0 + 4 \\
&= x_0^5 + 3x_0^4 + x_0^3 + 2x_0^3 + 6x_0^2 + 2x_0 + x_0^2 + 3x_0 + 1 + 3 \\
&= (x_0^5 + 3x_0^4 + x_0^3) + (2x_0^3 + 6x_0^2 + 2x_0) + (x_0^2 + 3x_0 + 1) + 3 \\
&= x_0^3(x_0^2 + 3x_0 + 1) + 2x_0(x_0^2 + 3x_0 + 1) + (x_0^2 + 3x_0 + 1) + 3 \\
&= x_0^3 \cdot \cancel{g(x_0)} + 2x_0 \cdot \cancel{g(x_0)} + \cancel{g(x_0)} + 3 \Rightarrow \\
f(x_0) &= 3
\end{aligned}$$

Άρα, αν $\mathcal{E} = \{E, D, \mathcal{K}, \mathcal{M}, \mathcal{C}\}$ το κρυπτοσύστημά μας τότε:

- $\mathcal{M} = \mathcal{C} = \{\alpha : 1, \beta : 2, ..., \omega\}$
- $\mathcal{K} = \{f(x_0)\} = \{3\}$
- $E(m,k)|_{k=3} \ = \ (m + k - 1)\ mod\ 24 + 1\ |_{k=3} \ = \ (m + 2)\ mod\ 24 + 1 = c$
- $D(c,k)|_{k=3} \ = \ (m - k - 1)\ mod\ 24 + 1\ |_{k=3} \ = \ (m - 4)\ mod\ 24 + 1 = m$

Είναι δηλαδή ένα σύστημα μετατόπισης, όπου η αρίθμηση δεν ξεκινάει από το 0, αλλά από το 1 (από εκεί προκύπτουν και οι επιπλέον άσσοι στις συναρτήσεις E και D). Προκύπτει ότι, αφού εισάγουμε το κρυπτομήνυμα **"οκηθμφδζθγοθχυκχσφθμφμχγ"**, δίνεται ως έξοδος το αποκρυπτογραφημένο μήνυμα **"μηδεισαγεωμετρητοσεισιτω"**.

(Υλοποίηση: **κώδικας/ex1/ex1.py**)

Άσκηση 2 (2.3)

Χρησιμοποιώντας τη μέθοδο του Friedman προκύπτει ότι το μήκος κλειδιών και των δύο μηνυμάτων είναι 7. Μια brute force υλοποίηση εύρεσης κλειδιού βρίσκεται μέσα στο script ex2.py όπου ψάχνει όλες τις δυνατές συμβολοσειρές μήκους 7 με τους 26 λατινικούς χαρακτήρες (26^7 συνδυασμοί).

Εκτυπώνονται μόνο οι αποκρυπτογραφήσεις των οποίων το κείμενο είναι σχετικά αναγνώσιμο, δηλαδή η συχνότητα εμφάνισης των γραμμάτων είναι κοντά με την πραγματική.

(Υλοποίηση: **κώδικας/ex1/ex1.py**)

Άσκηση 3 (2.4)

Έχουμε, $E : \{0,1\}^{16} \to \{0,1\}^{16}$ με $E(m) = m \oplus (m << 6) \oplus (m << 10) = c$. Γράφουμε τις δυαδικές ακολουθίες με τη μορφή διανυσμάτων:

$$\begin{bmatrix} c_{15} \\ c_{14} \\ c_{13} \\ c_{12} \\ c_{11} \\ c_{10} \\ c_9 \\ c_8 \\ c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} m_{15} \\ m_{14} \\ m_{13} \\ m_{12} \\ m_{11} \\ m_{10} \\ m_9 \\ m_8 \\ m_7 \\ m_6 \\ m_5 \\ m_4 \\ m_3 \\ m_2 \\ m_1 \\ m_0 \end{bmatrix} \oplus \begin{bmatrix} m_9 \\ m_8 \\ m_7 \\ m_6 \\ m_5 \\ m_4 \\ m_3 \\ m_2 \\ m_1 \\ m_0 \\ m_{15} \\ m_{14} \\ m_{13} \\ m_{12} \\ m_{11} \\ m_{10} \end{bmatrix} \oplus \begin{bmatrix} m_5 \\ m_4 \\ m_3 \\ m_2 \\ m_1 \\ m_0 \\ m_{15} \\ m_{14} \\ m_{13} \\ m_{12} \\ m_{11} \\ m_{10} \\ m_9 \\ m_8 \\ m_7 \\ m_6 \end{bmatrix} = \begin{bmatrix} m_{15} \oplus m_9 \oplus m_5 \\ m_{14} \oplus m_8 \oplus m_4 \\ m_{13} \oplus m_7 \oplus m_3 \\ m_{12} \oplus m_6 \oplus m_2 \\ m_{11} \oplus m_5 \oplus m_1 \\ m_{10} \oplus m_4 \oplus m_0 \\ m_9 \oplus m_3 \oplus m_{15} \\ m_8 \oplus m_2 \oplus m_{14} \\ m_7 \oplus m_1 \oplus m_{13} \\ m_6 \oplus m_0 \oplus m_{12} \\ m_5 \oplus m_{15} \oplus m_{11} \\ m_4 \oplus m_{14} \oplus m_{10} \\ m_3 \oplus m_{13} \oplus m_9 \\ m_2 \oplus m_{12} \oplus m_8 \\ m_1 \oplus m_{11} \oplus m_7 \\ m_0 \oplus m_{10} \oplus m_6 \end{bmatrix}$$

Μπορούμε οπτικά να συμπεράνουμε ότι

$$c_i \;=\; m_i \;\oplus\; m_{(i+10)\,mod\,16} \;\oplus\; m_{(i+6)\,mod\,16}, \quad i = 0, 1, ..., 15$$

ή αλλιώς

$$c_{i\,mod\,16} \;=\; m_{i\,mod\,16} \;\oplus\; m_{(i+10)\,mod\,16} \;\oplus\; m_{(i+6)\,mod\,16}, \quad \forall i \in \mathbb{N}$$

Προκύπτει (μετά από πειραματισμούς) ότι αν κάνουμε **XOR** κατά μέλη τα c_2, c_4, c_6, c_8, c_{10} παίρνουμε το έκτο ψηφίο του του αρχικού μηνύματος (m_6). Παρατίθεται η απόδειξη:

$$\begin{array}{rrcccccc} & c_{(i+2)\,mod\,16} & = & m_{(i+2)\,mod\,16} & \oplus & m_{(i+12)\,mod\,16} & \oplus & m_{(i+8)\,mod\,16} \\ & c_{(i+4)\,mod\,16} & = & m_{(i+4)\,mod\,16} & \oplus & m_{(i+14)\,mod\,16} & \oplus & m_{(i+10)\,mod\,16} \\ & c_{(i+6)\,mod\,16} & = & \mathbf{m_{(i+6)\,mod\,16}} & \oplus & m_{(i+\cancel{16})\,mod\,16} & \oplus & m_{(i+12)\,mod\,16} \\ & c_{(i+8)\,mod\,16} & = & m_{(i+8)\,mod\,16} & \oplus & m_{(i+2)\,mod\,16} & \oplus & m_{(i+14)\,mod\,16} \\ \oplus & c_{(i+10)\,mod\,16} & = & m_{(i+10)\,mod\,16} & \oplus & m_{(i+4)\,mod\,16} & \oplus & m_{(i+\cancel{16})\,mod\,16} \\ \hline & & & & & & & m_{(i+6)\,mod\,16} \end{array}$$

Δηλαδή,

$$c_{(i+2)\,mod\,16} \oplus c_{(i+4)\,mod\,16} \oplus c_{(i+6)\,mod\,16} \oplus c_{(i+8)\,mod\,16} \oplus c_{(i+10)\,mod\,16} = m_{(i+6)\,mod\,16}$$

Θέτοντας $u = i - 10 \Rightarrow i = u + 10$ η παραπάνω σχέση γράφεται ως:

$$m_{u \, mod \, 16} = c_{(u+12) \, mod \, 16} \oplus c_{(u+14) \, mod \, 16} \oplus c_{u \, mod \, 16} \oplus c_{(u+2) \, mod \, 16} \oplus c_{(u+4) \, mod \, 16}$$

Όπως είδαμε στην αρχή, η αριστερή κυκλική κύλιση κατά 6 αντιστοιχεί στη μαθηματική έκφραση $\mathbf{m_{(i+10) \, mod \, 16}}$ ενώ η αριστερή κυκλική κύλιση κατά 10 αντιστοιχεί στη μαθηματική έκφραση $\mathbf{m_{(i+6) \, mod \, 16}}$.

Γενικότερα η κυκλική κύλιση κατά α, $0 < \alpha < N$ σε έναν αριθμό των N bits μετατοπίζει τα πρώτα α MSBs στη θέση των πρώτων α LSBs. Άρα το LSB (θέση 0) τώρα θα είναι το ψηφίο που πρηγουμένως ήταν στη θέση $(N-1) - \alpha$. Άρα έκφραση που πήραμε ως αποτέλεσμα για τα ψηφία του m ως συνάρτηση των ψηφίων του c μπορεί να μετασχηματιστεί στην έκφραση:

$$m \; = \; (c << 4) \; \oplus \; (c << 2) \; \oplus \; c \; \oplus \; (c << 14) \; \oplus \; (c << 12) \; = \; D(c)$$

Το παραπάνω συμπέρασμα επιβεβαιώνεται πειραματικά από το script "ex3.py" καθώς τερματίζει με **error_flag = False**.

(Υλοποίηση: **κώδικας/ex3/ex3.py**)

Άσκηση 4 (2.5)

Ένα σύστημα μετατόπισης θα μπορούσε να έχει τέλεια ασφάλεια αν το κλειδί άλλαζε για κάθε χαρακτήρα με τυχαίο, ομοιόμορφο τρόπο.
Συγκεκριμένα, έστω το σύστημα $\mathcal{E} = \{E, D, \mathcal{K}, \mathcal{M}, \mathcal{C}\}$ με

- $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0, 1, ..., 23\}$
- $E(m, k) = (m + k) \, mod \, 2 = c$
- $D(c, k) = (c - k) \, mod \, 2 = m$

Έστω ένα κείμενο t μήκους ℓ χαρακτήρων. Αν m_0, $m_1 \in \mathcal{M}$ δύο οποιοιδήποτε χαρακτήρες του κειμένου $t \in \mathcal{M}^\ell$ θα ισχύει η εξίσωση τέλειας ασφάλειας:

$$Pr(k \xleftarrow{\$} \mathcal{K} : E(k, m_0) = c) \; = \; Pr(k \xleftarrow{\$} \mathcal{K} : E(k, m_1) = c)$$

αφού κάθε κρυπτογραφημένος χαρακτήρας c μπορεί να είναι το αποτέλεσμα μετατόπισης κατά οποιοδήποτε αριθμό k θέσεων.

Συμπερασματικά προκύπτει ότι κάθε συγκεκριμένο γράμμα δε θα μετατρέπεται σε ένα άλλο συγκεκριμένο γράμμα, άλλα σε οποιοδήποτε άλλο γράμμα για κάθε ξεχωριστή του εμφάνιση (άρα και δεν υπάρχει κίνδυνος όσον αφορά την εύρεση του κλειδιού με στατιστικές μεθόδους κλπ).

Αυτή η υλοποίηση ενός συστήματος μετατόπισης μοιάζει με το One Time Pad.

Άσκηση 5 (2.6)

Ο πίνακας για την μετατροπή των συμβόλων σε δυαδικές ακολουθίες των 5 bit υλοποιείται ως ένα λεξικό (συγκεκριμένα, δύο λεξικά για τη μετατροπή από σύμβολο σε bits και από bits σε σύμβολο).
Η συνάρτηση κρυπτογράφησης είναι ίδια με τη συνάρτηση αποκρυπτογράφησης

$$D(x, y) = E(x, y) = x \oplus y$$

όπως είναι φανερό και στον κώδικα.
Το κείμενο που θα χρησιμοποιηθεί ως μήνυμα βρίσκεται στο αρχείο **"sample_text.txt"**.

Να σημειωθεί ότι το κλειδί επιλέγεται ψευδοτυχαία με τη χρήση της βιβλιοθήκης random. Στην πραγματικότητα στο OTP το κλειδί τυχαίο, όμως επειδή το κλειδί πρέπει να έχει το ίδιο μήκος με το κείμενο, η επιλογή ενός κλειδιού με το χέρι είναι υπερβολή στην συγκεκριμένη περίπτωση.

Το πρόγραμμα εμφανίζει (με τη σειρά) το τυχαίο κλειδί, το κρυπτογραφημένο κείμενο και το αποκρυπτογραφημένο κείμενο, που σε κάθε περίπτωση είναι ίδιο με το περιεχόμενο του αρχείου.

(Υλοποίηση: **κώδικας/ex5/ex5.py**)

Άσκηση 6 (3.6)

Η άσκηση υλοποιήθηκε σε Python.

- Συνάρτηση **is_prime(n: int)**, επιστρέφει σε χρόνο $\mathcal{O}(\sqrt{n})$ αν ο n είναι πρώτος.
- Συνάρτηση **prime_factors(n: int)**, επιστρέφει μια λίστα $[p_1, p_2, ..., p_k]$ όπου

$$\prod_{i=1}^{k} p_i \;=\; n, \quad p_i \text{ πρώτοι αριθμοί}$$

- Συνάρτηση **divisors(n: int)**, επιστρέφει λίστα με όλους τους θετικούς διαιρέτες του n.
- Συνάρτηση **mu(n: int)**, υλοποίηση της συνάρτησης Möbius, $\mu(n)$.
- Συνάρτηση **N_2(n: int)**, το ζητούμενο της άσκησης.

Το πρόγραμμα διαλέγει 10 τυχαίους ακεραίους, $\mathbf{n_i}$, στο διάστημα [1, 100]. Εκτυπώνει, με την ακόλουθη σειρά, τους διαιρέτες του, τους πρώτους παράγοντές του και το πλήθος των ανάγωγων πολυωνύμων βαθμού n_i στο σώμα $\mathbb{F}_2$. Τα ακόλουθα αποτελέσματα εκτυπώνονται για $n = 30$:

```
[30] info:
-- divisors:  [1, 2, 3, 5, 6, 10, 15, 30]
-- prime factorization:  [2, 3, 5]
-- N_2(30) = 35790267
```

(Υλοποίηση: **κώδικας/ex6/ex6.py**)

Άσκηση 7 (3.8)

Όπως και σε προηγούμενες ασκήσεις, το script χρημοποιεί λεξικά για τις κωδικοποιήσεις και αποκωδικοποιήσεις των 32 δοσμένων χαρακτήρων σε δυαδικές ακολουθίες των 5-bit (όπως επιβάλλει ο πίνακας της άσκησης 3.5) και, φυσικά, την υλοποίηση της XOR.

Ο αλγόριθμος RC4 απαιτεί 8-bit κωδικοποίηση των χαρακτήρων, όμως οι χαρακτήρες μας κωδικοποιούνται σε 5 bits ο καθένας. Μια πρώτη σκέψη θα ήταν να κάνουμε padding τρία μηδενικά μπροστά από τις δυαδικές αναπαραστάσεις των χαρακτήρων. Αυτό όμως δεν αποτελεί λύση του προβλήματος καθώς αν `S[i] > 31` για κάποιο i, τότε το αποτέλεσμα της XOR ενός χαρακτήρα με το `S[i]` θα προκαλούσε υπερχείλιση.

Έτσι υλοποιήθηκε, απλά, ο RC4 με όριο το 32 αντί για το 256. Άρα ο S είναι $S = [0, 1, ..., 31]$ αντί για $S = [0, 1, ..., 255]$, οι δείκτες i, j μηδενίζονται κάθε 32 επαναλήψεις αντί για 256 κλπ.

Για την κρυπτογράφηση και την αποκρυπτογράφηση χρησιμοποιείται η ίδια συνάρτηση `rc4()` που υλοποιεί τον αλγόριθμο αρχικοποίησης, την κατσκευή μετάθεσης και τον αλγόριθμο κρυπτοροής ταυτόχρονα.

Για μήνυμα `"MISTAKESAREASSERIOUSASTHERESULTSTHEYCAUSE"` και κλειδί `"HOUSE"` παίρνουμε ως κρυπτοκείμενο το `"IGD!APO-TJUQPDSPMAOZUIAZ(VF(VFGQ.IIWMB(WX"`. Βάζοντας το κρυπτοκείμενο στην ίδια συνάρτηση και το ίδιο κλειδί παίρνουμε ξανά το αρχικό μήνυμα, πράγμα που επιβεβαιώνει ότι οι συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης είναι ίδιες.

(Υλοποίηση: **κώδικας/ex7/ex7.py**)

Άσκηση 8 (4.3)

Το script υλοποιεί τον τύπο υπολογισμού της διαφορικής ομοιομορφίας:

$$Diff(S) \ = \ \max_{x \in F_2^n - \{0\},\, y \in F_2^m} |\{z \in F_2^n : \ S(x \oplus z) \oplus S(z) \ = \ y\}|$$

Για $n = 6$, $m = 4$ και $S : \ \{0,1\}^6 \to \{0,1\}^4$ με τύπο του S-box (4.2.3), το πρόγραμμα τερματίζει εμφανίζοντας ως έξοδο `Diff(S) = 14`.

Μερικές πληροφορίες για τον κώδικα:

- Λίστα `S_i:` υλοποίηση του εσωτερικού lookup table της συνάρτησης `S`
- Συνάρτηση `S(x: str) -> str:` η συνάρτηση $S : \ \{0,1\}^6 \to \{0,1\}^4$
- Συνάρτηση `xor(x: str, y: str) -> str:` επιστρέφει $x \oplus y$
- Συνάρτηση `differential_uniformity(...):` δέχεται τη συνάρτηση `S` και το μήκος των συμβολοσειρών εισόδων της. Επιστρέφει το $Diff(S)$

(Υλοποίηση: **κώδικας/ex8/ex8.py**, αποτελεί αντιγραφή του κώδικα για το δεύτερο ερώτημα του homework 4 με μικρο-αλλαγές)

Άσκηση 9 (4.7)

Η άσκηση 4.7 προϋποθέτει την ύπαρξη υλοποίησης του κρυπταλγορίθμου AES. Δεδομένου ότι δεν υπάρχει κάποιο προεγκατεστημένο πακέτο της Python με την εν λόγω υλοποίηση, χρησιμοποιήθηκε το πακέτο **PyCryptodome (v 3.17)** - (documentation link). Η εγκατάσταση έγινε στη γραμμή εντολών με την εντολή: `pip install pycryptodome` (Αντίστοιχα, πληροφορίες για το περιβάλλον conda παρέχονται εδώ).

Όλα τα εργαλεία της βιβλιοθήκης βρίσκονται στο module **Crypto**. Αναφέρται στο documentation ότι η PyCryptodome παρουσιάζει προβλήματα αν είναι ταυτόχρονα εγκατεστημένη η PyCrypto (παλιά PyCryptodome). Σε αυτή την περίπτωση πρέπει να εγκατασταθεί η PyCryptodomeX με την εντολή `pip install pycryptodomex`. Στην περίπτωση εγκατάστασης αυτής της ανεξάρτητης εκδοχής τα εργαλεία βρίσκονται στο module **Cryptodome**.

Σχετικά με τον AES, υπάρχουν οδηγίες στο documentation που χρησιμοποιήθηκαν για την υλοποίηση της άσκησης (καθώς και από μια άλλη πηγή που θα αναφερθεί αργότερα).

Το αρχείο "messages.txt" περιέχει 78 μηνύματα που θα χρησιμοποιηθούν για το στατιστικό τεστ που θα διαπιστώσει το βαθμό διάχυσης του αλγορίθμου AES. Παρατίθενται τα βήματα του προγράμματος:

1. Διαβάζονται τα μηνύματα από το αρχείο και αποθηκεύονται σε έναν πίνακα.
2. Αποθηκεύονται με την ίδια σειρά τα μηνύματα οι επεξεργασμένες εκδοχές τους (ένα bit-flip) σε άλλον πίνακα.
3. Δημιουργείται ένα κλειδί των 16 bytes (Άρα AES-128).
4. Δημιουργούμε δύο AES αντικείμενα με το κλειδί, ένα σε ECB λειτουργία και ένα σε CBC λειτουργία.
5. Δημιουργείται μια δομή (zip object) με ζεύγη $(c_i,\ c_i')$ που προκύπτουν από την κρυπτογράφηση των $(m_i,\ m_i')$ με AES-128 ECB mode, όπου $m_i,\ m_i'$ μηνύματα των 256 bit που διαφέρουν κατά ένα bit. Κάνουμε το ίδιο και για το CBC mode. Να σημειωθεί ότι τα μηνύματα είναι λιγότερα από 32 bytes οπότε γίνεται padding με τη χρήση του `Padding.pad` (Για αυτό το θέμα χρησιμοποιήθηκε κώδικας από αυτό το post στο Stack Overflow).
6. Δημιουργείται ένα αρχείο στο οποίο καταγράφεται το ποσοστό διαφορετικών bits μεταξύ c_i και c_i' (κάνοντας XOR και μετρώντας το πλήθος των άσσων του αποτελέσματος) για κάθε $i = 1, 2, ..., 78$. Επίσης καταγράφεται και ο μέσος όρος αλλαγής ψηφίων τόσο στο αρχείο, όσο και στην οθόνη. Αυτή η διαδικασία γίνεται δύο φορές, μία για κάθε AES λειτουργία.

Παρατηρείται ότι το ο AES-128 δε χαρακτηρίζεται από υψηλή διάχυση στη λειτουργία ECB καθώς το μέσο ποσοστό ψηφίων που επηρεάστηκαν στο κρυπτομήνυμα είναι περίπου 25%, δηλαδή $< 50\%$.

Κατά τ΄ άλλα αν εκτελεστεί ο κρυπταλγόριθμος στη CBC λειτουργία τότε το αποτέλεσμα αλλάζει δραστικά, με περίπου $50.5\% - 51\%$ αλλαγή ψηφιων. Επομένως το φαινόμενο της χιονοστιβάδας παρατηρείται στον αλγόριθμο Rijndael, όχι όμως σε όλες τις λειτουργίες του.

(Υλοποίηση: **κώδικας/ex9/ex9.py**)

Άσκηση 10

1. Εισάγουμε το **course-1-introduction.pdf** σε αυτό το PDF Metadata Viewer.

2. Αντιγράφουμε το Base64 string που βρίσκεται στη θέση XMP → TIFF:ARTIST

   aHR0cHM6Ly9jcnlwdG9sb2d5LmNzZC5hdXRoLmdyOjgwODAvaG9tZS9wdWIvMTUv

   και το εισάγουμε σε έναν αποκωδικοποιητή. Έτσι, προκύπτει ο σύνδεσμος:

   https://cryptology.csd.auth.gr:8080/home/pub/15/

3. Η ιστοσελίδα γράφει «Ο χημικός Walter White βρήκε μέσα στο εργαστήριο του το εξής μήνυμα: #2-75-22-6!». Το δοσμένο string δεν είναι ο κωδικός, όμως, αν γίνει κάποια συσχέτιση με τη χημεία, συμπεραίνουμε ότι οι αριθμοί αυτοί είναι ατομικοί αριθμοί στοιχείων, άρα $(2, 75, 22, 6) \rightarrow (He, Re, Ti, C)$. Επομένως, εισάγοντας στο secure.zip τον κωδικό **#heretic!**, τότε αυτό ανοίγει.

4. Μέσα στο secret.zip βρίσκεται το secret.txt. Αποκωδικοποιώντας το μεγάλο Base64 string προκύπτει μια φωτογραφία του Bobby Fischer να παίζει σκάκι. Στα μεταδεδομένα της φωτογραφίας υπάρχει ένα tinyurl.

5. Η ιστοσελίδα γράφει (σε αλγεβραϊκή μορφή) την τρέχουσα κατάσταση μιας παρτίδας σκάκι. Κάνοντας prompt το ChatGPT με εντολή:

   ```
   Calculate the next best move:
   1.e4 e5 2.f4 exf4 3.Bc4 g5 4.Nf3 g4 5.O-O gxf3 6.Qxf3 Qf6 7.e5
   Qxe5 8.d3 Bh6 9.Nc3 Ne7 10.Bd2 Nbc6 11.Rae1 Qf5 12.Nd5 Kd8
   13.Bc3 Rg8 14.Bf6 Bg5 15.Bxg5 Qxg5 16.Nxf4 Ne5 17.Qe4 d6 18.h4
   Qg4 19.Bxf7 Rf8 20.Bh5 Qg7 21.d4 N5c6 22.c3 a5 23.Ne6+ Bxe6
   24.Rxf8+ Qxf8 25.Qxe6 Ra6 26.Rf1 Qg7 27.Bg4 Nb8
   ```

   αυτό απαντάει:

   ```
   In this position, White has a strong initiative, but Black is still defending well.
   28.Rf7
   ...
   ```

6. Υπολογίζουμε το `hashlib.md5(b'Rf7').hexdigest()` που είναι ίσο με:

   f1f5e44313a1b684f1f7e8eddec4fcb0 που είναι όντως το κλειδί του secure2.zip. Το mastersecret.txt περιέχει ως απάντηση το hash:

   be121740bf988b2225a313fa1f107ca1