# 1ο Project στο μάθημα της Κρυπτογραφίας

Δήμητρα Παζούλη, 3902

May 2023

## Άσκηση 1

Το επόμενο κρυπτόγραμμα έχει ληφθεί:

**οκηθμφδζθγοθχυκχσφθμφμχγ**

Ο αλγόριθμος κρυπτογράφησης είναι ο εξής: Κάθε γράμμα του αρχικού μας μηνύματος αντικαθίσταται από την αριθμητική του τιμή (α → 1, ..., ω → 24). Ας είναι $x_0$ μία ρίζα του τριωνύμου $g(x) = x^2 + 3x + 1$. Σε κάθε αριθμό του μηνύματός μου προσθέτω την τιμή του πολυωνύμου $f(x) = x^5 + 3x^3 + 7x^2 + 3x^4 + 5x + 4$, στο $x_0$. Αντικαθιστώ κάθε αριθμό με το αντίστοιχο γράμμα. Βρείτε το αρχικό μήνυμα.

Ο συγκεκριμένος αλγόριθμος κρυπτογράφησης είναι ιδιαίτερα απλός στην κατανόηση, καθώς δεν χρειάστηκε πάνω από 10-15 γραμμές στην γλώσσα προγραμματισμού Python για να υλοποιηθεί.
Οι συναρτήσεις **greek_to_num()** και **num_to_greek()**, που αντιστοιχίζουν τους χαρακτήρες του ελληνικού αλφαβήτου στις αριθμητικές τιμές τους και αντίστροφα, αξιοποιούν τις συναρτήσεις ord() και char() της Python. Συγκεκριμένα, η θέση του χαρακτήρα άλφα (α) βρίσκεται στην θέση 945, άρα αρκεί να αφαιρέσουμε το 944 για να φέρουμε το τον χαρακτήρα άλφα στην επιθυμητή τιμή 1. Ομοίως πράττουμε και για τις υπόλοιπες τιμές. Βέβαια, επειδή στον πίνακα ASCII υπάρχει και το τελικό σίγμα (ς) στην θέση 962 δημιουργείται μια μικρή επιπλοκή που πρέπει να προσέξουμε, αφού το αλφάβητο του δικού μας αλγόριθμου αποτελείται μόνο από 24 χαρακτήρες, χωρίς να συμπεριλαμβάνεται αυτός.
Ο αλγόριθμος λοιπόν επιλέγει κάθε χαρακτήρα του ciphertext ξεχωριστά, τον μετατρέπει στην αριθμητική του τιμή μέσω την συνάρτησης που προαναφέραμε, **αφαιρεί** από τον αριθμό αυτό την τιμή του πολυωνύμου της εκφώνησης και ύστερα πραγματοποίεται η αντίστροφη μετατροπή από νούμερο σε αλφαβητικό χαρακτήρα. Στην 2η μετατροπή είναι σημαντικό να προσέξουμε ο αριθμός να μην βγει έξω από τα όρια 1 έως 24. Επομένως, εκτελούμε πρώτα την πράξη ( $x \bmod 24$ ) + 1.
Εισάγοντας στο πρόγραμμα μας το κρυπτόγραμμα που δόθηκε στην εκφώνηση, λαμβάνουμε την φράση **'μηδεισαγεωμετρητοσεισιτω'**. Προσθέτοντας τα απαραίτητα κενά παρατηρούμε πως πρόκειται για την επιγραφή στην είσοδο της Ακαδημίας του Πλάτωνα.

# Άσκηση 2

Αποκρυπτογραφήστε το κείμενο που σας δόθηκε, το οποίο κρυπτογραφήθηκε με τον αλγόριθμο του Vigenère. Συστήνουμε να χρησιμοποιήσετε python. Για το μήκος του κλειδιού μπορείτε να χρησιμοποιήσετε είτε test Kasiski ή τη μέθοδο του Friedman.

Για την αποκρυπτογράφηση εκτελούμε τα παρακάτω βήματα:

1. Με την μέθοδο Kasiski βρίσκουμε το μήκος και ύστερα το περιεχόμενο του κλειδιού το οποίο χρησιμοποιήθηκε για την κρυπτογράφηση του αρχικού μηνύματος.

2. Χρησιμοποιόντας την συνάρτηση **longKey(key, length))**, επαναλαμβάνουμε το κλειδί στον εαυτό του μέχρι να φτάσει στο επιθυμητό μήκος, δηλαδή να έχει ίσο μήκος με το κρυπτόγραμμα. Παραδείγματος χάριν, για length = 10 έχουμε DOG $\rightarrow$ DOGDOGDOGD.

3. Αντιστιχίζουμε κάθε γράμμα του key και ciphertext στις αριθμητικές του τιμές (A $\rightarrow$ 0, ..., Z $\rightarrow$ 25). Αφαιρούμε τις τιμές του κλειδιού από τις τιμές του ciphertext , ώστε να λάβουμε τα plaintext numbers. Προσέχουμε ο αριθμός να μην βγει έξω από τα όρια 0 έως 25, άρα εκτελούμε την πράξη ( $x \bmod 26$ ) Το βήμα (3) που μόλις περιγράψαμε αναλαμβάνει η **decipher_vigenere(c, key)**.

4. Μετατρέπουμε τα plaintext numbers πίσω σε χαρακτήρες του αλφάβητου χρησιμοποιώντας την συνάρτηση της Python chr(x + 65). Ο λόγος που προσθέτουμε την τιμή 65 είναι επειδή στο Unicode ASCII table το κεφαλαίο λατινικό A βρισκεται στην θεση 65. Το αρχικό μήνυμα είναι πλέον έτοιμο προς εκτύπωση στον χρήστη.

Εφαρμόζουμε τα παραπάνω για τα παραδείγματα που δόθηκαν στην εκφώνηση και βρίσκουμε πως τα κλειδιά των κειμένων είναι 'EMPEROR' και 'SHANNON' αντίστοιχα. Εκτελώντας τον κώδικα λαμβάνουμε την περίφημη ομιλία από την ταινία **The Great Dictator (1940)** και το επιδραστικό άρθρο του Claude Shannon με τίτλο **'Creative Thinking'(1952)**.

## Άσκηση 3

> Έστω ένα μήνυμα $m$, 16-bits. Θεωρούμε την κυκλική κύλιση προς τα αριστερά $<<$ a κατά a bits. Έστω ότι το $m$ κωδικοποιείται στο $c$ σύμφωνα με τον τύπο,
>
> $$c = m \oplus (m << 6) \oplus (m << 10)$$
>
> Βρείτε τον τύπο αποκωδικοποίησης. Δηλαδή, γράψτε το $m$ ως συνάρτηση του $c$. Υλοποιήστε κατάλληλο κώδικα, για να δείξετε ότι ο τύπος που φτιάξατε είναι σωστός.

Έστω $c = c_0 c_1 c_2 c_3 ... c_{15}$ και $m = m_0 m_1 m_2 m_3 ... m_{15}$
Απο τον τύπο κωδικοποίησης προκύπτει ότι:
$c_k = m_k \oplus m_{k+6} \oplus m_{k+10} \forall k = 0, 1, 2..15$ Παρατηρουμε ότι

$$c_{k+6} \oplus c_{k+12} = m_{k+6} \oplus m_{k+12} \oplus m_{k+16} \oplus m_{k+12} \oplus m_{k+18} \oplus m_{k+22} = m_k \oplus m_{k+2} \tag{1}$$

επειδή $c$, $m$ 16 bits ισχύει ότι $c_j = c_{j+16x} \forall j = 0, 1, 2, 3..15$ και $\forall x \in \mathbb{Z}$
Επειδή η επιλογή του $k$ είναι τυχαία, ισχύει επίσης ότι:

$$c_{k+8} \oplus c_{k+14} = m_{k+2} \oplus m_{k+4} \tag{2}$$

$$c_{k+10} \oplus c_k = m_{k+4} \oplus m_{k+6} \tag{3}$$

$$c_k \oplus c_{k+6} = m_{k+10} \oplus m_{k+12} \tag{4}$$

Παρατηρούμε ότι:
$(1)\oplus(2)\oplus(3)\oplus(4) = m_k \oplus m_{k+2} \oplus m_{k+2} \oplus m_{k+4} \oplus m_{k+4} \oplus m_{k+6} \oplus m_{k+10} \oplus m_{k+12} = m_k \oplus m_{k+6} \oplus m_{k+10} \oplus m_{k+12} = c_k \oplus m_{k+12} =>$
$m_{k+12} = (1) \oplus (2) \oplus (3) \oplus (4) \oplus c_k = c_{k+6} \oplus c_{k+12} \oplus c_{k+8} \oplus c_{k+14} \oplus c_{k+10} \oplus c_k \oplus c_k \oplus c_{k+6} \oplus c_k =>$
$m_{k+12} = c \oplus c_{k+8} \oplus c_{k+10} \oplus c_{k+12} \oplus c_{k+14}$
Συνεπώς $m_k = c_{k-12} \oplus c_{k-4} \oplus c_{k-2} \oplus c_k \oplus c_{k+2}$
Επομένως, ο τύπος αποκωδικοποίησης είναι ο εξής:

$$m = c \oplus (c << 2) \oplus (c >> 2 \oplus (c << 4) \oplus (c << 12))$$

Για να επαληθεύσουμε την ορθότητα του αλγορίθμου αποκωδικοποίησης, αρκεί να πάρουμε όλα τα δυνατά μηνύματα $m$ μήκους 16-bits και να εφαρμόσουμε στο καθένα. Πράγματι, αν τρέξετε τον επισυναπτόμενο κώδικα Python παρατηρούμε πως $m = decipher(encode(m))$ $\forall$ δυαδικό αριθμό $m$, όπου $len(m) = 16$.

## Άσκηση 4

> Να αποδείξετε ότι, αν στο σύστημα μετατόπισης διαλέγουμε τυχαία τα κλειδιά από το σύνολο 0, 1, ..., 23, τότε το σύστημα έχει τέλεια ασφάλεια.

Σύμφωνα με τον **Shannon**, τέλεια ασφάλεια σημαίνει ότι αν κάποιος έχει το κρυπτομήνυμα $c$, αυτό δεν του παρέχει καμιά πληροφορία για το αρχικό μήνυμα $m$, και αυτό συμβαίνει για κάθε $m, c$. Ικανή και αναγκαία συνθήκη για να έχει ένα κρυπτοσύστημα τέλεια ασφάλεια είναι :

$$Pr = (C = c|P = m) = Pr(C = c)$$ ,

όπου $C$ το κρυπτογραφημένο μήνυμα, $P$ το αρχικό μήνυμα, $c$ ένα πιθανό κρυπτομήνυμα και $m$ ένα πιθανό αρχικό μήνυμα. Από την εκφώνηση ξέρουμε πως τα κλειδιά επιλέγονται εντελώς τυχαία, δηλαδή **οι πιθανότητες να επιλεχθεί ένα απο τα 24 κλειδιά είναι ίσες**$(= 1/24)$. Βάση των παραπάνω, μόνο με την απλή γνώση του $c$ η EVE δεν μπόρει να αποσπάσει περισσότερες πληροφορίες για το plaintext $m$, οπότε καταλήγουμε στο συμπέρασμα ότι το σύστημα έχει τέλεια ασφάλεια.

## Άσκηση 5

> Υλοποιήστε τον OTP, αφού αρχικά μετατρέψετε το μήνυμά σας σε bits με χρήση του πίνακα. Θα πρέπει να δουλεύουν η κρυπτογράφηση και η αποκρυπτογράφηση. Το μήνυμα δίνεται κανονικά και εσωτερικά μετατρέπεται σε bits. Το κλειδί είναι διαλεγμένο τυχαία και έχει μήκος όσο το μήκος του μηνύματός σας. Το αποτέλεσμα δίνεται όχι σε bits, αλλά με λατινικούς χαρακτήρες.

Για την κρυπτογράφηση ενός μηνύματος με την χρήση One Time Pad εκτελούμε τα παρακάτω βήματα:

1. Δημιουργούμε ένα κλειδί το οποίο είναι (μεγαλύτερο ή) ίσο με το αρχικό μήνυμα. Το κλειδί αποτελείται από μια σειρά τυχαίων αριθμών ή γραμμάτων. Επομένως κατασκευάζουμε την συνάρτηση **random_binary(length)** , η οποία χρησιμοποιεί μια γεννήτρια ψευδοτυχαίων δυαδικών αριθμών random. randint(0, 1). Να σημειωθεί πως εδώ πρέπει να γίνει import η βιβλιοθήκη random της Python.

2. Ορίζουμε την λίστα **symbols = ['A', 'B', ..., 'Z', '.', '!', '?', ...]**, βάση του πίνακα που μας δόθηκε στο textbook, ο οποίος θα αποτελεί το αλφάβητο του συστήματός μας. Θα φανεί χρήσιμο αργότερα στην μετατροπή ενός αριθμού στο αντίστιχο λατινικό χαρακτήρα/σύμβολό του και αντίστροφα.

3. Μετατρέπουμε το μήνυμα σε δεκαδική μορφή και ύστερα σε δυαδική μορφή μέσω της συνάρτησης **text_to_binary(a)** . Αυτό μπορεί να γίνει χρησι-

μοποιώντας το σχήμα κωδικοποίησης ASCII. Κάθε χαρακτήρας/σύμβολο αντιστοιχίζεται σε ένα μοναδικό 5-bit δυαδικό αριθμό.

4. Προσθέτουμε κάθε ψηφίο του μηνύματος στο αντίστοιχο ψηφίο του κλειδιού μαζί (mod 2), χρησιμοποιώντας την συνάρτηση **XOR(a, b)**. Το αποτέλεσμα θα είναι ένα κρυπτογραφημένο μήνυμα ως σειρά από 0 και 1.

5. Μετατρέπουμε τους δυαδικούς αριθμούς πίσω σε γράμματα/σύμβολα. Η συνάρτηση **binary_to_text(a)** επιλέγει ανά 5αδα τα ψηφία του δυαδικού αριθμού (str) και τα μετατρεπει σε δεκαδικους αριθμους και υστερα σε γραμματα μέσω του index από την λίστα symbols.

6. Στέλνουμε το κρυπτογραφημένο μήνυμα και το κλειδί στον αποδέκτη.

Για την αποκρυπτογράφηση ακολουθούμε την ακριβώς αντίστροφη διαδικασία, η οποία θα είναι πλέον πολύ ευκολότερη στην υλοποίηση αφού έχουμε όλες τις συναρτήσεις διαθέσιμες. Μόλις τελειώσουμε την αποκρυπτογράφηση **καταστρέφουμε το κλειδί**, καθώς η επαναχρησιμοποίηση του κλειδιού μπορεί να θέσει σε κίνδυνο την ασφάλεια του συστήματος.

# Άσκηση 6

Αποδεικνύεται ότι το πλήθος των ανάγωγων πολυωνύμων βαθμού $n$ στο σώμα $\mathbb{F}_2$ είναι

$$N_2(n) = \frac{1}{n}\sum_{d|n} \mu(d) 2^{\frac{n}{d}}$$

όπου

$$m(d) = \begin{cases} 1 & d = 1 \\ (-1)^k & d = p_1 p_2 \cdots p_k (p_i : \text{πρώτοι}) \\ 0 & \text{αλλού} \end{cases}$$

Με το σύμβολο $d|n$ εννοούμε όλους τους θετικούς διαιρέτες του ν. Π.χ. αν $n = 30$, τότε

$$\{d|n : 1 \leq d \leq n\} = \{1, 2, 3, 5, 6, 10, 15, 30\}.$$

Με χρήση του συστήματος sagemath υπολογίστε το $N_2(10)$.

Παρακάτω βλέπουμε μία απλή υλοποίηση των δύο συναρτήσεων **m(d)** και **N(n)**.
Η συνάρτηση **contains_square(num)** επίσης παίζει βασικό ρόλο στην δομή του προγράμματός μας. Συγκεκριμένα, εξετάζει εάν ένας αριθμός ανήκει στην 2η ή στην 3η κατηγορία της συνάρτησης $\mu(d)$. Αν το γινόμενο της παραγοντοποίησης του αριθμού είναι γινόμενων γνήσιων πρώτων αριθμών. Π.χ. το 20 ανήκει στην 3η κατηγορία, αφού ισχύει $2 * 2 * 5 = 20$. Αυτό είναι εφικτό μέσω της συνάρτησης του sagemath factor().

Τέλος, υπολογίζουμε το πλήθος των ανάγωγων πολυωνύμων βαθμού $n$ στο σώμα $\mathbb{F}_2$ $N_2(n)$ της εκφώνησης, δηλαδή αντικαθιστούμε όλους τους διαιρέτες του $n$ στον τύπο, τα προσθέτουμε και πολλαπλασιάζουμε το άθροισμα με $\frac{1}{n}$. Για τιμή $n = 10$, παίρνουμε αποτέλεσμα $N_2(10) = 99$.

```python
def contains_square(n):
    factors = list(factor(n))
    for i in range(len(factors)):
        if factors[i][1] > 1:
            return True
    return False
def m(d):
    if d==1:
        return 1
    if not contains_square(d):
        k = len(prime_divisors(d))
        return (-1)**k
    return 0
def N(n):
    N = 0
    for d in divisors(n):
        N += m(d) * (2**(n/d))
    return (1/n)*N
N(10)
```

# Άσκηση 7

Υλοποιήστε τον RC4. Χρησιμοποιώντας το κλειδί HOUSE κρυπτογραφήστε το μήνυμα (ξαναγράψτε το κείμενο χωρίς κενά):
MISTAKES ARE AS SERIOUS AS THE RESULTS THEY CAUSE
Η υλοποίησή σας πρέπει και να αποκρυπτογραφεί σωστά.

Ο αλγόριθμος RC4 που υλοποίησα λειτουργεί ως εξής:

1. Το κλειδί RC4 παράγεται με μια διαδικασία που ονομάζεται Key Scheduling Algorithm. Δημιουργούμε έναν πίνακα $S$ 256 byte που αρχικοποιείται με τις τιμές 0 έως 255 με τη σειρά. Το κλειδί χρησιμοποιείται για τη δημιουργία μεταθέσεων των τιμών στον πίνακα καταστάσεων με βάση το κλειδί.

2. Μετά τη δημιουργία του κλειδιού RC4, χρησιμοποιείται το stream generation για τη δημιουργία μιας ροής ψευδοτυχαίων byte.Το stream generation χρησιμοποιεί τον πίνακα καταστάσεων $S$, ο οποίος αρχικοποιήθηκε στο βήμα 1. Πρόκειται για έναν βρόγχο που επαναλαμβάνεται σε κάθε byte στο απλό κείμενο. Για κάθε byte στο απλό κείμενο, δημιουργείται ένα νέο byte της ροής κλειδιών. Ο πίνακας $S$ ενημερώνεται με εναλλαγή τιμών με βάση τον τρέχοντα δείκτη και την τιμή του τρέχοντος byte της ροής κλειδιών.

3. Εφαρμόζω την πράξη XOR στην ροή κλειδιών που δημιουργείται και στο αρχικό κείμενο για την παραγωγή του κρυπτογραφημένου κειμένου.

4. Όσο αφορά την αποκρυπτογράφηση του κειμένου, χρησιμοποιώ το ίδιο κλειδί για να αρχικοποιήσω τον πίνακα $S$ και να δημιουργήσω την ίδια ροή κλειδιών που χρησιμοποιήθηκε για την κρυπτογράφηση του απλού κειμένου. Ύστερα εφαρμόζω πάλι την πράξη XOR .

Παραδείγματος χάριν, αν τρέξω τον αλγόριθμο παίρνω τα εξής αποτελέσματα:
c = 0X150XDD0XA60X670XFE0X6A0XDD0X1A0X280X50X6D0...
m = MISTAKESAREASSERIOUSASTHERESULTSTHEYCAUSE
PRESS ENTER TO EXIT
Είναι σημαντικό να σημειωθεί ότι το RC4 δεν θεωρείται πλέον ασφαλές και συνιστάται η χρήση άλλων αλγορίθμων κρυπτογράφησης όπως ο AES.

# Άσκηση 8

Αν Σ ένα σύνολο με |Σ| συμβολίζουμε το πλήθος των στοιχείων του, υπολογίστε τη διαφορική ομοιομορφία (diffential uniformity) του S-box (4.2.3).

$$\text{Diff}(S) = max|\{z \in \{0,1\}^6\} : S(z \oplus x) \oplus S(z) = y|$$

Όσο μικρότερη είναι αυτή η ποσότητα, τόσο πιο ανθεκτικό είναι το S-Box στη διαφορική κρυπτανάλυση.

$$\textbf{S-Box=} \begin{bmatrix} 0 & 2 & 3 & 7 & 9 & 12 & 15 & 7 & 6 & 15 & 15 & 1 & 7 & 3 & 1 & 0 \\ 1 & 5 & 6 & 13 & 4 & 1 & 5 & 11 & 7 & 8 & 7 & 1 & 1 & 3 & 2 & 13 \\ 5 & 3 & 5 & 12 & 11 & 1 & 1 & 5 & 13 & 0 & 15 & 7 & 2 & 2 & 13 & 0 \\ 3 & 12 & 3 & 11 & 2 & 2 & 2 & 4 & 6 & 5 & 5 & 0 & 4 & 3 & 1 & 0 \end{bmatrix}$$

Αρχικά μεταφέρουμε το S-box από το textbook σε ενα δισδιάστατο πίνακα 4x16 με δυαδικούς αριθμούς αποθηκεύμενους ως string. Τώρα με την συνάρτηση **S(num, sbox)** μπορούμε να μετατρέπουμε ένα δυαδικό αριθμό 6-bit σε 4-bit σύμφωνα με τον πίνακα S-box. Με δεδομένη μια είσοδο 6 bit, η έξοδος 4 bit βρίσκεται επιλέγοντας τη σειρά χρησιμοποιώντας τα εξωτερικά δύο bit (το πρώτο και το τελευταίο bit)και τη στήλη χρησιμοποιώντας τα εσωτερικά τέσσερα bit. Για παράδειγμα, μια είσοδος ΄011011΄ έχει εξωτερικά βιτς ΄01΄ και εσωτερικά βιτς ΄1101΄, οπότε η αντίστοιχη έξοδος θα ήταν στην 2η σειρά και 14η στήλη.
Τώρα καλούμε την συνάρτηση **diff(sbox, n, max_diff = 0)**. Ως ορίσματα δίνουμε το S-box της εκφώνησης και τον χώρο εισόδου μεγέθους, δηλαδή $16 * 4 = 64$. Υπολογίζει τη διαφορική ομοιομορφία βάση του τύπου που μας δόθηκε στην εκφώνηση.
Οι πράξεις XOR ανάμεσα στους 6-bit δυαδικούς αριθμούς εκτελούνται με την συνάρτηση **XOR(a, b)**, τις οποίας τα ορίσματα είναι string. Ο λόγος που έκανα αυτήν την επιλογή είναι επειδή στην Python δεν είναι δυνατό να ξεκινάει ένας ακέραιος με το 0, οπότε ορισμένοι δυαδικοί αριθμοί πχ ′011010′ δεν θα ήτανε δυνατόν να αναπαρισταθούν με άλλο τρόπο πέρα της συμβολοσειράς.

Η συνάρτηση **max_count_occurences(my_list)** χρησιμοποιεί την βιβλιοθήκη Collections ώστε να υπολογίσει το πλήθος του πιο συχνού στοιχείου σε μία λίστα. Αν και θα ήτανε εφικτό να υλοποίηθει και με απλά for loops κλπ., προτίμησα να χρησιμοποιήσω αυτή την βιβλιοθήκη ώστε να είναι ο κώδικας πιο αποδοτικός και εύκολα συντηρίσιμος.
Εκτελώντας τον κώδικα που μόλις περιγράψαμε, η διαφορική ομοιομορφία υπολογίζεται ίση με 14.

## Άσκηση 9

> Εξετάστε αν ισχύει το avalanche effect στον AES-128. Αναλυτικότερα, φτιάξτε αρκετά ζευγάρια ($> 30$) μηνυμάτων ($m1, m2$) που να διαφέρουν σε ένα bit. Εξετάστε σε πόσα bits διαφέρουν τα αντίστοιχα κρυπτομηνύματα. Δοκιμάστε με δύο καταστάσεις λειτουργίας : ECB και CBC (η δεύτερη θέλει και IV block ). Τα μήκη των μηνυμάτων που θα χρησιμοποιήσετε να έχουν μήκος διπλάσιο του μήκους ενός block. Δηλαδή για τον AES-128, να είναι μήκους 256 bits.

## Άσκηση 10

> (CTF-like) Μπορείτε να ανοίξετε το secure.zip? Hint. Ότι χρειάζεστε είναι στην διαφάνεια course-1-Introduction.pdf στο elearning .

Με third-party programs όπως John The Ripper προσπάθησα να εφαρμόσω brute force. Δυστυχώς λόγω περιορισμένου χρόνου (3 μέρες) δεν κατάφερα να καταλήξω σε κάποιο στοιχείο για τον κωδικό του αρχείου που μας δόθηκε.