



Πανεπιστήμιο Δυτικής Αττικής  
Σχολή Μηχανικών  
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών  
ΠΜΣ «Κυβερνοασφάλεια»

## Εφαρμοσμένη Κρυπτογραφία

Διδάσκων: Δρ. Παναγιώτης Ριζομυλιώτης

### Εργασία 1

CsCyb23004 Βασιλειάδης Δημήτριος

Αιγάλεω, 11 Δεκεμβρίου 2023



## Περιεχόμενα

Περιεχόμενα.....	i
Μεθοδολογία και παραδοχές .....	1
Εκτέλεση των αρχείων των σεναρίων .....	3
Επίθεση πρώτου σεναρίου .....	5
Περιγραφή της δομής .....	5
Ανάλυση της υλοποίησης .....	6
Στάδιο προετοιμασίας:.....	6
Στάδιο εκτέλεσης της επίθεσης: .....	7
Περιγραφή του αρχείου myfunctions.py .....	8
Περιγραφή του αρχείου binaryfunc.py .....	10
Αποτελέσματα εκτέλεσης πρώτου σεναρίου.....	11
Περιγραφή του αποτελέσματος της εκτέλεσης της επίθεσης: .....	11
Επίθεση δεύτερου σεναρίου.....	13
Περιγραφή της δομής .....	13
Ανάλυση της υλοποίησης .....	14
Στάδιο προετοιμασίας:.....	15
Στάδιο εκτέλεσης της επίθεσης: .....	16
Περιγραφή του αρχείου myfunctions2.py .....	17
Αποτελέσματα εκτέλεσης δεύτερου σεναρίου .....	20
Παρατήρηση από την εκτέλεση της επίθεσης .....	29



## Μεθοδολογία και παραδοχές

Για την υλοποίηση της επίθεσης γενικά στηριζόμαστε στα χαρακτηριστικά της συνάρτησης XOR και τον πίνακα τιμών της

bit A	bit B	bit C = A $\oplus$ B
0	0	0
0	1	1
1	0	0
1	1	0

Οπότε μπορούμε αν έχουμε τρεις τιμές A, B, C που σχετίζονται μεταξύ τους και επιλέξουμε οποιαδήποτε δύο μπορούμε να βρούμε την τρίτη. Αυτό μπορούμε να το εκμεταλλευτούμε στις επιθέσεις προκειμένου είτε να βρούμε το κλειδί (One Time Pad) που επαναχρησιμοποιήθηκε για να κρυπτογραφήσει πολλαπλά μηνύματα.

Με αυτό το χαρακτηριστικό της XOR συνάρτησης, όταν έχουμε γνωστό το κρυπτογραφημένο μήνυμα και άγνωστο το αρχικό μήνυμα, όμως με γνωστές κάποιες προδιαγραφές της δημιουργίας του (π.χ. πως το αλφάριθμο ότι αποτελείται μόνο από τους αριθμητικούς χαρακτήρες), μπορούμε να δοκιμάσουμε διάφορα κλειδιά για να εξετάσουμε εάν θα μπορούσε το κλειδί να μας δώσει κάποιο μήνυμα που ικανοποιεί αυτές τις γνωστές προδιαγραφές.

Στο κώδικα εκτέλεσης των επιθέσεων έχουν γίνει κάποιες παραδοχές που δεν αφορούν τόσο την ίδια την επίθεση, αλλά περισσότερο την δυνατότητα εκτύπωσης και τη μορφοποίηση της εμφάνισης των αποτελεσμάτων. Τα στοιχεία δεν εισάγονται από τον χρήστη αλλά είτε υπάρχουν «καρφωμένα» στον κώδικα, είτε δημιουργούνται με ψευδοτυχαίο τρόπο. Επιπλέον όταν κάποιο στοιχείο εισόδου δημιουργείται με τυχαίο τρόπο προσπαθούμε να επιλέγεται από κάποιο εύρος τιμών του οποίου ο αντίστοιχος ASCII χαρακτήρας να είναι εκτυπώσιμος, για να μπορεί να εμφανιστεί στην οθόνη και να κατανοήσουμε τι εισάγεται και ποιο είναι το αποτέλεσμα.

Τέλος τα δεδομένα που είτε εισάγονται είτε εμφανίζονται σε δυαδική μορφή ομαδοποιούνται ανά οκτάδες και χωρίζονται με έναν κενό διάστημα για να είναι ευκολότερη η κατανόηση κατά την εμφάνιση τους.

Για την απλοποίηση του κώδικα και να μην γίνει προσθήκη ελέγχων που δεν αφορούν τον αλγόριθμο αλλά τον έλεγχο της εισόδου του χρήστη την ορθότητα και την πληρότητα των δεδομένων που εισάγονται, δεν έχουν ενσωματωθεί τέτοιου είδους έλεγχοι. Θεωρούμε πως τα δεδομένα είναι πλήρη, είναι σε σωστή μορφή και καλύπτουν κάθε φορά τις προδιαγραφές που ζητούνται από τις ασκήσεις.

Για παράδειγμα όταν εισάγονται ciphertexts από τον χρήστη (είτε καρφωτά στον κώδικα) δεν γίνεται έλεγχος εάν η είσοδος στα δυαδικά δεδομένα περιλαμβάνει άλλους χαρακτήρες πλην των ‘0’ και ‘1’, ούτε εάν έχουν συμπληρωθεί 5 bits αντί για 8.

Επιπλέον στο δεύτερο σενάριο θεωρούμε πως εάν ο χρήστης δώσει καρφωτά κάποια ciphers για αποκρυπτογράφηση θεωρούμε πως αυτά έχουν προκύψει από την κρυπτογράφηση με κάποιο κλειδί ενός plaintext που αποτελείται από αριθμητικούς χαρακτήρες,

Ενδεικτικά αποτελέσματα της εκτέλεσης υπάρχουν στα αρχεία:

output\_1\_1.txt για το πρώτο σενάριο

και στο αρχείο

output\_1\_2.txt για το δεύτερο σενάριο

Στην άσκηση αναφέρονται οι όροι «χαρακτήρας» και «byte» έχοντας την ίδια έννοια με την αναφορά σε χαρακτήρες ως τα bytes που στην ascii κωδικοποίηση αντιστοιχούν σε εκτυπώσιμους χαρακτήρες, ενώ ως byte αναφερόμαστε γενικότερα στον ascii κωδικό κάποιου χαρακτήρα που μπορεί να είναι και μη εκτυπώσιμος.

## Εκτέλεση των αρχείων των σεναρίων

Η επίθεση του πρώτου σεναρίου μπορεί να γίνει εάν από γραμμή εντολών εκτελέσουμε την εντολή:

`python3 Senario_1_1.py`

Η επίθεση του δεύτερου σεναρίου μπορεί να γίνει εάν από γραμμή εντολών εκτελέσουμε την εντολή:

`python3 Senario_1_1.py [μέγιστο πλήθος αποτελεσμάτων]`

για παράδειγμα εάν εκτελέσουμε την εντολή:

`python3 Senario_1_1.py 8`

Η οποία θα φέρει έως 8 αποτελέσματα συνδυασμών κλειδιών και μηνυμάτων που μπορούν να βρεθούν σε κάθε μία από τις επιθέσεις που εκτελούνται.

Το μέγιστο πλήθος αποτελεσμάτων που μπορούμε να δώσουμε σαν προαιρετική παράμετρο θα περιορίσει τα αποτελέσματα στο πλήθος που δίνουμε, αντί να εμφανίζονται όλα τα αποτελέσματα επειδή το πλήθος είναι μικρό και δεν θα περιοριστεί στο μέγιστο προεπιλεγμένο πλήθος (είτε αυτό που δίνουμε στην προαιρετική παράμετρο είτε στο 2 που έχουμε σαν προεπιλογή).

Στην περίπτωση που έχουμε ciphertexts που τα αποτελέσματα των συνδυασμών των κλειδιών και των αντίστοιχων πιθανών μηνυμάτων είναι έως 16, τότε εμφανίζονται όλα τα αποτελέσματα επειδή το πλήθος είναι μικρό και δεν θα περιοριστεί στο μέγιστο προεπιλεγμένο πλήθος (είτε αυτό που δίνουμε στην προαιρετική παράμετρο είτε στο 2 που έχουμε σαν προεπιλογή).

Αυτή η απόφαση πάρθηκε για το λόγο, ότι επειδή το πλήθος των 16 συνδυασμών είναι μικρό, είναι προτιμότερο να φαίνονται όλοι οι δυνατοί συνδυασμοί για να φανεί πως σε κάποιον από τους δυνατούς συνδυασμούς θα εμφανιστούν και τα αρχικά plaintexts με τα οποία δημιουργήθηκαν τα ciphertexts.

Οι επιθέσεις που εκτελούνται είναι οι εξής:

- i. Επίθεση όταν έχουμε 2 ciphertexts ίσου μήκους 2 χαρακτήρων το καθένα
- ii. Επίθεση όταν έχουμε 3 ciphertexts ίσου μήκους 2 χαρακτήρων το καθένα
- iii. Επίθεση όταν έχουμε 4 ciphertexts (τα τρία πρώτα ίσου μήκους, 2 χαρακτήρων το καθένα, και το τέταρτο μεγαλύτερου μήκους, 6 χαρακτήρων)

Στο αρχείο `Senario_1_2.py` του πηγαίου κώδικα μπορούμε στη γραμμή 32 να αλλάξουμε τα μήκη των plaintexts (και των ciphertexts αντίστοιχα) που θα παραχθούν.

Στη γραμμή 33 του ίδιου αρχείου υπάρχει σε σχόλιο εάν θέλουμε να δοκιμάσουμε κάποιες διαφορετικές τιμές για τα μήκη

```
plain_text_lengths = [2, 3, 2, 6]
#plain_text_lengths = [6, 6, 6, 5]
```

Φυσικά μπορούμε να δοκιμάσουμε με οποιεσδήποτε διαφορετικές τιμές.

Παράδειγμα εκτέλεσης του κώδικα υπάρχει στην ιστοσελίδα στη διεύθυνση

[https://vasileiadis.eu/cscyb103/scenario\\_1\\_1.php](https://vasileiadis.eu/cscyb103/scenario_1_1.php)

(για εκτέλεση του πρώτου σεναρίου)

[https://vasileiadis.eu/cscyb103/scenario\\_1\\_2.php?keys=10](https://vasileiadis.eu/cscyb103/scenario_1_2.php?keys=10)

(για εκτέλεση του δεύτερου σεναρίου, με εμφάνιση 10 αποτελεσμάτων συνδυασμών)

## Επίθεση πρώτου σεναρίου

Όταν τα μηνύματα M1 και M2 έχουν το ίδιο μήκος τότε και τα αντίστοιχα ciphertexts C1 και C2 θα έχουν επίσης το ίδιο μήκος. Αφού γνωρίζοντας πως το ίδιο κλειδί έχει χρησιμοποιηθεί για την κρυπτογράφηση και των δύο μηνυμάτων μπορούμε να χρησιμοποιήσουμε το κλειδί ή το κομμάτι του κλειδιού που χρησιμοποιήθηκε, για να αποκρυπτογράφησουμε πλήρως και το C2.

Όταν τα μηνύματα δεν έχουν το ίδιο μήκος, τότε και τα αντίστοιχα ciphertexts δεν θα έχουν επίσης το ίδιο μήκος. Θα διακρίνουμε τις περιπτώσεις:

- Αν το M1 να έχει μεγαλύτερο μήκος από το M2 τότε και το αντίστοιχο C1 θα έχει μεγαλύτερο μήκος από το C2.

Η αποκωδικοποίηση του C2 θα γίνει χρησιμοποιώντας μόνο όσο τμήμα απαιτείται (όσο είναι το μήκος του C2) από το M1 και C1 αντίστοιχα.

- Αν το M1 να έχει μικρότερο μήκος από το M2 τότε και το αντίστοιχο C1 θα έχει μικρότερο μήκος από το C2.

Η αποκωδικοποίηση θα γίνει μόνο για κάποιο τμήμα του C2, όσο είναι το μήκος του C1. Το υπόλοιπο δεν μπορούμε να το γνωρίζουμε με ασφάλεια. Θα πρέπει να καταφύγουμε σε άλλες μεθόδους, τις οποίες όμως δεν υλοποιούμε στην παρούσα λύση.

Για πληροφοριακούς λόγους στην εμφάνιση των αποτελεσμάτων, θα εμφανίσουμε ποιο ήταν το αρχικό κλειδί της κρυπτογράφησης και ποιο είναι το κλειδί της κρυπτογράφησης που μπορέσαμε να βρούμε. Όταν αποκρυπτογραφούμε το μεγαλύτερο μήνυμα όπως στην ανωτέρω περίπτωση B) θα εμφανίσουμε το μέρος του μηνύματος που αποκρυπτογραφήσαμε καθώς και το υπόλοιπο μέρος του μηνύματος που δεν μπορέσαμε, αντικαθιστώντας όσους χαρακτήρες δεν είναι εκτυπώσιμοι.

### Περιγραφή της δομής

Η συγκεκριμένη επίθεση έχει υλοποιηθεί σε γλώσσα προγραμματισμού python (έκδοση 3), ομαδοποιώντας τις διαδικασίες σε διάφορα αρχεία, τα οποία έχουν την ακόλουθη χρήση.

Στο αρχείο `Senario_1_1.py` βρίσκεται ο κώδικας που υλοποιεί τον βασικό σκελετό του αλγόριθμου της επίθεσης. Αυτό το αρχείο πρέπει να εκτελεστεί προκειμένου να γίνει η επίθεση.

Στο αρχείο `myfunctions.py` βρίσκεται ο κώδικας που υλοποιεί τις λεπτομέρειες του αλγόριθμου της επίθεσης.

Στο αρχείο `binaryfunc.py` βρίσκεται βοηθητικός κώδικας που υλοποιεί βοηθητικές εργασίες σχετικές με τη μετατροπή ενός αριθμού από/σε binary μορφή κειμένου, σπάσιμο του ciphertext σε 8άδες χαρακτήρων, καθώς και τη δημιουργία τυχαίων κλειδιών που θα χρησιμοποιηθεί και στα δύο σενάρια, τροποποίηση ενός string ώστε να αποτελείται από εκτυπώσιμους χαρακτήρες, αλλά και συναρτήσεις που χρησιμοποιούνται και στα δύο σενάρια όπως η `Encrypt`, προκειμένου να μην επαναλαμβάνεται ο κώδικας.

Η επιλογή αυτή έγινε προκειμένου να έχουμε κομμάτια κώδικα επαναχρησιμοποιήσιμα ως είναι σαν βιβλιοθήκη.

Επιπλέον «σπάμε» τον συνολικό κώδικα στα βασικά βήματα του αλγόριθμου και σε «λεπτομέρειες» της υλοποίησης τις οποίες «κρύβουμε» μέσα στο αρχείο `myfunctions.py`.

## Ανάλυση της υλοποίησης

Για την αποκωδικοποίηση των μηνυμάτων θα χρησιμοποιηθεί, για τον κάθε χαρακτήρα του μηνύματος, η συνάρτηση:

$$C_i = M_i \oplus K_i$$

και αν λύσουμε ως προς  $K_i$ , προκειμένου να βρούμε το κλειδί, θα έχουμε:

$$K_i = M_i \oplus C_i$$

Οπότε θα έχουμε από το Known Plaintext Attack

$$C1 \oplus C2 \oplus M1 = M1 \oplus K \oplus M2 \oplus K \oplus M1 = M2$$

Αυτή τη συνάρτηση θα τη χρησιμοποιήσουμε για κάθε χαρακτήρα του γνωστού μηνύματος  $M1$  και του ciphertext  $C1$  αντίστοιχα συνδυάζοντας το με τον αντίστοιχο χαρακτήρα του άγνωστου ciphertext  $C2$  (ίσου μήκους με το  $C1$ ), το  $C3$  (μικρότερου μήκους από το  $C1$ ) και το  $C4$  (μεγαλύτερου μήκους από το  $C1$ ).

Στην αποκρυπτογράφηση του  $C3$  θα επαναλάβουμε μόνο για όσο είναι το μήκος του  $C3$  καθώς το υπόλοιπο τμήμα του  $M1$  και  $C1$  δεν το χρειαζόμαστε, ενώ για την αποκρυπτογράφηση του  $C4$  θα επαναλάβουμε μόνο για όσο είναι το μήκος του  $M1$  καθώς για το υπόλοιπο τμήμα δεν μπορούμε να βρούμε το κλειδί κρυπτογράφησης.

Αναλυτικά σχόλια υπάρχουν και μέσα στα αρχεία του πηγαίου κώδικα για αναφορά.

Στο αρχείο `Senario_1_1.py` ο κώδικας εκτελεί τα εξής βήματα:

Στάδιο προετοιμασίας:

1. Δίνονται κάποιες αρχικές τιμές για τα μηνύματα που τα έχουμε σαν plaintexts
2. Υπολογίζεται πόσο πρέπει να είναι το ελάχιστο μήκος του κλειδιού προκειμένου να μπορέσει να κρυπτογραφήσει όλα τα μηνύματα και αποθηκεύει το μήκος στη μεταβλητή `encryptionKeyLength`.
3. Δημιουργείται ένα τυχαίο κλειδί μήκους `encryptionKeyLength` χρησιμοποιώντας τη συνάρτηση `GenerateRandomKey` και αποθηκεύεται στη μεταβλητή `encryptionKey`.
4. Δημιουργούνται τα ciphertexts και αποθηκεύονται στις μεταβλητές `cipher1`, `cipher2`, `cipher2smaller`, `cipher2bigger` από τα αντίστοιχα plaintexts που έχουμε στις μεταβλητές `plainText1`, `plainText2`, `plainText2smaller`, `plainText2bigger`.
5. Τέλος, αδειάζει τις τιμές που έχουν οι μεταβλητές `plainText1`, `plainText2`, `plainText2smaller`, `plainText2bigger` ώστε να θεωρούμε πως είναι άγνωστα.

Τη μεταβλητή `encryptionsKey`, την κρατάμε ώστε να μπορέσουμε μετά να την εμφανίσουμε για πληροφοριακούς σκοπούς, αλλά ο κώδικας μας τη θεωρεί άγνωστη και δεν χρησιμοποιείται κάπου αλλού.

Στάδιο εκτέλεσης της επίθεσης:

Σε αυτό το σημείο είναι που ξεκινάει η «πραγματική» υλοποίηση της επίθεσης που ζητάει η άσκηση.

Έχουμε σαν γνωστά στοιχεία:

6. Το `plaintext`, στη μεταβλητή `plainText1`
7. Το γνωστό `ciphertext` που έχει προκύψει από το `plaintext` στη μεταβλητή `cipher1`
8. Το άγνωστο `ciphertext` ίδιου μήκους με το `cipher1` στη μεταβλητή `cipher2`
9. Το άγνωστο `ciphertext` μικρότερου μήκους από το `cipher1` στη μεταβλητή `cipher2smaller`
10. Το άγνωστο `ciphertext` μεγαλύτερου μήκους από το `cipher1` στη μεταβλητή `cipher2bigger`

Ο κώδικας, εκτελεί την επίθεση με τα ακόλουθα βήματα:

11. Για πληροφοριακούς λόγους εμφανίζει αυτά τα στοιχεία που τα έχουμε σαν γνωστά.
  12. Εξάγει το κλειδί της κρυπτογράφησης από το γνωστό `plainText1`, `cipher1`, και εμφανίζει ποιο μέρος του κλειδιού μπορεί να ανακτήσει και ποιο παραμένει άγνωστο. Αυτή η ενέργεια είναι καθαρά για πληροφόρηση.
  13. Κάνει την αποκρυπτογράφηση του `cipher2` καλώντας τη συνάρτηση `Decrypt` η οποία δέχεται σαν παράμετρο τα τρία απαραίτητα στοιχεία: το γνωστό `plaintext` (μεταβλητή `plainText1`), το γνωστό `ciphertext` (μεταβλητή `cipher1`) και το άγνωστο `ciphertext` (μεταβλητή `cipher2`). Το αποτέλεσμα της αποκρυπτογράφησης αποθηκεύεται στη μεταβλητή `plainText2`.
  14. Αντίστοιχα (επαναχρησιμοποιώντας τη συνάρτηση `Decrypt`, αλλά δίνοντας σαν 3η παράμετρο τη μεταβλητή `cipher2smaller`), γίνεται η αποκρυπτογράφηση και το αποτέλεσμα αποθηκεύεται στη μεταβλητή `plainText2smaller`.
  15. Επαναλαμβάνεται η διαδικασία και καλείται ξανά η συνάρτηση `Decrypt`, αλλά δίνοντας σαν 3η παράμετρο τη μεταβλητή `cipher2bigger`. Γίνεται η αποκρυπτογράφηση και το αποτέλεσμα αποθηκεύεται στη μεταβλητή `plainText2bigger`.
  16. Για πληροφόρηση εξάγεται το μέρος του μεγαλύτερου `plaintext` που δεν μπόρεσε να αποκρυπτογραφηθεί, μετατρέπεται σε εκτυπώσιμη μορφή και αποθηκεύεται στη μεταβλητή `remainingText2biggerPrintable`.
  17. Αφού έχει ολοκληρωθεί η αποκρυπτογράφηση εμφανίζονται τα αποκρυπτογραφημένα αρχικά μηνύματα, για όσο μέρος των μηνυμάτων μπόρεσε να γίνει η αποκρυπτογράφηση με ασφάλεια.
  18. Τέλος εμφανίζεται το περιεχόμενο της μεταβλητής `remainingText2biggerPrintable` που έχει το κομμάτι του μεγαλύτερου μηνύματος που δεν μπόρεσε να αποκρυπτογραφηθεί.
- Αυτά που μένουν είναι η υλοποίηση των λεπτομερειών της επίθεσης, που υπάρχουν στο αρχείο `myfunctions.py`.

### Περιγραφή του αρχείου myfunctions.py

Η συνάρτηση `ShowKey` είναι βοηθητική και παίρνει σαν παράμετρο το κλειδί που χρησιμοποιήθηκε για τη δημιουργία των ciphertexts. Επιστρέφει ένα κείμενο που αποτελείται από το κάθε byte του κλειδιού να απεικονίζεται σε binary μορφή χωρισμένο από το επόμενο με έναν κενό χαρακτήρα για ευκολότερη ανάγνωση.

Αντίστοιχα παρόμοια εργασία κάνει και η συνάρτηση `ShowKeyAsString` και παίρνει σαν παράμετρο το κλειδί που χρησιμοποιήθηκε για τη δημιουργία των ciphertexts. Όμως αντί να μετατρέπει το κάθε byte του κλειδιού σε binary μορφή το μετατρέπει στον αντίστοιχο ASCII χαρακτήρα, ώστε το συνολικό κλειδί να επιστρέφεται σαν απλό κείμενο. Η συνάρτηση αυτή εξυπηρετεί περισσότερο εάν το κλειδί γνωρίζουμε πως αποτελείται από χαρακτήρες που μπορούν να εμφανιστούν στην οθόνη ή σε κάποιον εκτυπωτή, όπως στην περίπτωση που γνωρίζουμε πως το κλειδί αποτελείται από γράμματα του λατινικού αλφαριθμητικού, αριθμητικά σύμβολα και άλλα εκτυπώσιμα σύμβολα.

Η συνάρτηση `ExportKey` εξάγει το κλειδί, χρησιμοποιώντας τα γνωστά `plaintext1` και `ciphertext1`, που χρησιμοποιήθηκε για την κρυπτογράφηση του `plaintext1`.

Η συνάρτηση `Decrypt` εκτελεί την αποκρυπτογράφηση του `ciphertext` και μας επιστρέφει το μέρος του αρχικού κειμένου που μπόρεσε να αποκρυπτογραφήσει σαν `string`. Δέχεται σαν παράμετρο το γνωστό κείμενο στη μεταβλητή `plaintext1`, το γνωστό `ciphertext` στη μεταβλητή `ciphertext1` και το άγνωστο `ciphertext` στη μεταβλητή `ciphertext2`. Για να μπορέσει να εκτελέσει την αποκρυπτογράφηση εκτελεί τα ακόλουθα βήματα:

19. Αφαιρεί τους περιττούς κενούς χαρακτήρες από το `ciphertext1`, εάν υπάρχουν (που είχαμε προσθέσει για να είναι καλύτερη η εμφάνιση κατά την εκτύπωση) και να μπορούμε να πάρουμε τα «καθαρά δεδομένα» του `ciphertext`.
20. Κάνει το ίδιο ακριβώς και για το `ciphertext2`
21. Σπάει το `ciphertext1` σε πίνακα από 8άδες χαρακτήρων, ώστε η κάθε θέση του πίνακα να περιέχει το binary string του κάθε χαρακτήρα του `ciphertext1` και το αποθηκεύει στη μεταβλητή `ciphertext1List`
22. Προκειμένου να μπορούμε να κάνουμε τις απαιτούμενες πράξεις είναι απαραίτητη η μετατροπή των binary strings σε αριθμό (τον ASCII κωδικό του χαρακτήρα του `ciphertext`). Οπότε χρησιμοποιώντας τη συνάρτηση `BinaryToNumber` γίνεται μετατροπή από binary string σε αριθμό για κάθε χαρακτήρα  $x_i$  του `ciphertext1List` (που έχουμε υπολογίζει στο βήμα 21)
23. Εκτελείται η αντίστοιχη διαδικασία που γίνεται στο βήμα 21, όμως για το `ciphertext2` και ο πίνακας από τα binary strings αποθηκεύεται στη μεταβλητή `ciphertext2List`
24. Εκτελείται η αντίστοιχη διαδικασία που γίνεται στο βήμα 22, όμως για κάθε χαρακτήρα  $x_i$  του `ciphertext2List` (που έχουμε υπολογίσει στο βήμα 23)

Τώρα οι επόμενες ενέργειες της αποκωδικοποίησης, μπορούν να εκτελεστούν με ασφάλεια, μόνο για όσο μήκος είναι κοινό ανάμεσα στο `plaintext` και τα δύο ciphertexts. Οπότε έχουμε

25. Αρχικοποιείται η μεταβλητή `decrypted_text` που θα κρατάει τους αποκρυπτογραφημένους χαρακτήρες με ένα κενό/άδειο string, καθώς ακόμα δεν υπάρχει τίποτα αποκρυπτογραφημένο.
26. Ελέγχει εάν υπάρχει ο i-οστός χαρακτήρας στο `plaintext1` και η i-οστή θέση στον πίνακα `cipherList1` και η i-οστή θέση στον πίνακα `cipherList2`.  
Εάν δεν υπάρχει η i-οστή θέση πηγαίνει στο βήμα 31.  
Εάν υπάρχει η i-οστή θέση εκτελούνται τα εξής:
  27. Αποθηκεύεται ο i-οστός χαρακτήρας του `plaintext1` στη μεταβλητή `ri`, ο i-οστός αριθμός του πίνακα `cipherList1` στη μεταβλητή `ci` και ο i-οστός αριθμός του πίνακα `cipherList2` στη μεταβλητή `ci2`
  28. Βρίσκουμε τον ASCII κωδικό του χαρακτήρα του `ri` και κάνουμε πράξεις XOR με το `ci` και πάλι XOR με το `ci2`. Το αποτέλεσμα αυτής της πράξης αποθηκεύεται στη μεταβλητή `ch`. Δηλαδή εκτελούμε την πράξη:
$$ch = P_i \oplus C_i \oplus C_{i2}$$
  29. Προσθέτουμε τον αποκρυπτογραφημένο χαρακτήρα `ch` που βρήκαμε στους προηγούμενους που έχει η μεταβλητή `decrypted_text` και το αποτέλεσμα το κρατάμε πάλι στη μεταβλητή `decrypted_text`.
  30. Επαναλαμβάνουμε από το βήμα 26 για τον επόμενο χαρακτήρα (i+1)
  31. Επιστρέφουμε το αποκρυπτογραφημένο κείμενο που έχει η μεταβλητή `decrypted_text`.

Η συνάρτηση `EncodedTextAsString` επιστρέφει σαν κείμενο το κομμάτι του `cipher` που δίνουμε σαν παράμετρο από τον χαρακτήρα στη θέση `startPosition` και μετά. Υπάρχει σαν βοηθητική για να εμφανίσουμε το κομμάτι του μεγαλύτερου `ciphertext` που δεν μπορέσαμε να αποκρυπτογραφήσουμε.

### Περιγραφή του αρχείου binaryfunc.py

Στο αρχείο `binaryfunc.py`, υπάρχουν οι εξής βοηθητικές συναρτήσεις που χρησιμοποιούνται και στα δύο σενάρια.

Η συνάρτηση `GenerateRandomKey` είναι βοηθητική και δημιουργεί ένα τυχαίο κλειδί OTP μήκους `keyLength` bytes. Για τη δημιουργία του κλειδιού παρότι μπορούμε να χρησιμοποιήσουμε οποιαδήποτε τιμή χωράει σε ένα byte, προτιμούμε για ευκολία να περιορίσουμε το εύρος τιμών σε αριθμούς και γράμματα του λατινικού αλφαριθμητικού, κεφαλαία και μικρά. Η επιλογή αυτή έγινε προκειμένου να μπορούν να εκτυπωθούν και να είναι κατανοητή η τιμή που έχει το κλειδί.

Η συνάρτηση `GenerateRandomDecimalString` είναι βοηθητική και δημιουργεί ένα τυχαίο κείμενο που αποτελείται αποκλειστικά από αριθμητικούς χαρακτήρες.

Η συνάρτηση `LeftNibble` είναι βοηθητική και επιστρέφει σαν αριθμό την τιμή που δείχνουν τα bits 7-4 ενός binary string.

Η συνάρτηση `RightNibble` είναι βοηθητική και επιστρέφει σαν αριθμό την τιμή που δείχνουν τα bits 3-0 ενός binary string.

Η συνάρτηση `NumberToBinary` είναι βοηθητική και επιστρέφει σε string τη δυαδική αναπαράσταση ενός αριθμού 8 bit

Η συνάρτηση `BinaryToNumber` είναι βοηθητική και μετατρέπει ένα binary string από 8 bits σε αριθμό

Η συνάρτηση `SplitStringByLength` είναι βοηθητική και σπάει ένα κείμενο που δίνεται στην παράμετρο `string` ανά τόσους χαρακτήρες όσο είναι η παράμετρος `length` και επιστρέφει το αποτέλεσμα σαν πίνακα από strings.

Η συνάρτηση `SplitStringByDelimiter` είναι βοηθητική και σπάει ένα κείμενο που δίνεται στην παράμετρο `string` κάθε φορά που εμφανίζεται ο χαρακτήρας που δίνεται στην παράμετρο `delimiter`. Το αποτέλεσμα επιστρέφεται σαν πίνακα από strings.

Η συνάρτηση `Encrypt` εκτελεί την κρυπτογράφηση ενός μηνύματος και επιστρέφει το κρυπτογραφημένο μήνυμα. Δέχεται σαν παράμετρο το μήνυμα και το κλειδί και επιστρέφει το ciphertext σαν binary string με τον κάθε κρυπτογραφημένο χαρακτήρα να χωρίζεται με ένα κενό από το επόμενο. Η συνάρτηση για κάθε χαρακτήρα του καθαρού μηνύματος κάνει την κωδικοποίηση χρησιμοποιώντας την πράξη.

$$C_i = M_i \oplus K_i$$

Το αποτέλεσμα της πράξης το μετατρέπει σε κείμενο binary μορφή χρησιμοποιώντας τη συνάρτηση `NumberToBinary` και προσθέτει έναν κενό χαρακτήρα στο τέλος για να χωρίζεται το ένα byte από το άλλο για ευκολία στην εμφάνιση. Στο τέλος αφαιρεί τον τελευταίο κενό χαρακτήρα αφού δεν χρειάζεται καθώς δεν υπάρχει επόμενος.

Η συνάρτηση `MakeTextPrintable` είναι βοηθητική και δέχεται σαν παράμετρο ένα κείμενο και αντικαθιστά τους μη εκτυπώσιμους χαρακτήρες με κάποιον άλλο συγκεκριμένο εκτυπώσιμο.

## Αποτελέσματα εκτέλεσης πρώτου σεναρίου

Ενδεικτική Εκτέλεση 1

---

```
Plain Text 1      - SIMPLE KNOWN MESSAGE
Cipher Text 1     - 01100001 00000010 00101001 00000011 00101110 01110111 01000111
00111110 00011110 00011101 00011011 00010100 01111001 00100111 00001111 00100110 00110110
00101101 00101001 00000100
Cipher Text 2      - 01100101 00100011 00000101 00100111 01000010 01110011 00101010
01010101 00011001 01110010 00001000 00010101 00010000 00100100 00001101 01010101 00001101
00001001 00011100 00100100
Cipher Text 2 smaller - 01100110 00100011 00001011 00100110 00000101 01011010 00010011
01010101 00111001 00100001 01101100 00111100 00101011 00001111 00101111
Cipher Text 2 bigger - 01100101 00100011 00000101 00100111 01000010 01011011 00010100
01010101 00100000 00110011 00111111 00101110 01111001 00000011 00111001 01010101 00010101
00011110 00000001 00101101 01011010 00110010 00100001 00100100
```

Encryption key used : 2KdSb2guPRLZYjJuelnA**SUTA**

Extracted key from attack : 2KdSb2guPRLZYjJuelnA

```
Decrypted Plain Text 2      - What AM I DOING here
Decrypted Plain Text 2 smaller - Thought is free
Decrypted Plain Text 2 bigger - What is past is prol
    Ramaining 2 bigger - Z2!$
```

Περιγραφή του αποτελέσματος της εκτέλεσης της επίθεσης:

Όπως μπορούμε να δούμε και μέσα στον πηγαίο κώδικα του αρχείου Scenario\_1\_1.py δίνονται plaintexts τα οποία κρυπτογραφούνται τα οποία έχουν τις εξής τιμές:

```
plainText1      = "SIMPLE KNOWN MESSAGE"
plainText2      = "What AM I DOING here"
plainText2smaller = "Thought is free"
plainText2bigger = "What is past is prologue"
```

Το κλειδί για την κρυπτογράφηση δημιουργείται με ψευδοτυχαίο τρόπο, με το μήκος του μεγαλύτερου από αυτά τα μηνύματα γιατί είναι απαραίτητο για να γίνει η κρυπτογράφηση και του plainText2bigger.

Το κλειδί που δημιουργήθηκε έχει τιμή 2KdSb2guPRLZYjJuelnA**SUTA**

αλλά εμείς μπορέσαμε να ανακτήσουμε μόνο ένα μέρος του και το κομμάτι “**SUTA**” δεν μπορέσαμε να το ανακτήσουμε.

Τα ciphers που έχουν μήκος ίσο ή μικρότερο με το μήκος του plaintext 1, αποκρυπτογραφήθηκαν κανονικά

Το cipher του μεγαλύτερου μηνύματος μπόρεσε να αποκρυπτογραφηθεί μόνο για ένα μέρος του, γι αυτό και μας εμφανίζεται το κείμενο “What is past is prol”, ενώ το υπόλοιπο τμήμα που

αντιστοιχεί στους τελευταίους χαρακτήρες “ouge” παρέμεινε κρυπτογραφημένο και εμφανίζεται ως “Z2!\$”

Ενδεικτική Εκτέλεση 2

---

```
Plain Text 1      - SIMPLE KNOWN MESSAGE
Cipher Text 1     - 00100010 00000000 00110101 00111101 00001101 00110011 01111000
00011011 00000001 00011100 00011001 00100111 01000010 01111101 00000100 00000001 00010101
00100101 00101100 00010111
Cipher Text 2     - 00100110 00100001 00011001 00011001 01100001 00110111 00010101
01110000 00000110 01110011 00001010 00100110 00101011 01111110 00000110 01110010 00101110
00000001 00011001 00110111
Cipher Text 2 smaller - 00100101 00100001 00010111 00011000 00100110 00011110 00101100
01110000 00100110 00100000 01101110 00001111 00010000 01010101 00100100
Cipher Text 2 bigger - 00100110 00100001 00011001 00011001 01100001 00011111 00101011
01110000 00111111 00110010 00111101 00011101 01000010 01011001 00110010 01110010 00110110
00010110 00000100 00111110 00101101 01010111 00011101 01010110
```

Encryption key used : qIx<sub>m</sub>AvXPOSNib0ARFd<sub>k</sub>R<sub>B</sub>0h3

Extracted key from attack : qIx<sub>m</sub>AvXPOSNib0ARFd<sub>k</sub>R

```
Decrypted Plain Text 2      - What AM I DOING here
Decrypted Plain Text 2 smaller - Thought is free
Decrypted Plain Text 2 bigger - What is past is prol
Remaining 2 bigger - -WxxV      ( xx represent non printable character)
```

Σε μία δεύτερη ενδεικτική εκτέλεση έχουμε την εξής διαφοροποίηση:

Το κλειδί που χρησιμοποιήθηκε για την κωδικοποίηση με ψευδοτυχαίο τρόπο έχει διαφορετική τιμή, οπότε αλλάζει και το αποτέλεσμα των ciphertexts.

Το κομμάτι του μεγαλύτερου ciphertext που αντιστοιχεί στο τμήμα “ouge” και παρέμεινε κρυπτογραφημένο, δεν αντιστοιχεί ολόκληρο σε εκτυπώσιμους χαρακτήρες του πίνακα ascii, οπότε προκειμένου να εμφανίσουμε κάτι που να μπορεί να εκτυπωθεί αντικαταστήσαμε κάθε μη εκτυπώσιμο χαρακτήρα με τον χαρακτήρα “<sub>xx</sub>”. Η επιλογή αυτή έγινε σε αντίθεση με άλλες εφαρμογές που προτιμούν να χρησιμοποιούν τον χαρακτήρα “.” προκειμένου να μπορούμε να ξεχωρίσουμε εάν αφορά πραγματικά τον χαρακτήρα “.” (τελεία) ή αν πρόκειται για κάποιον μη εκτυπώσιμο χαρακτήρα που αντικαταστάθηκε.

## Επίθεση δεύτερου σεναρίου

Όταν τα ciphertexts C1, C2 έχουν το ίδιο μήκος τότε σημαίνει πως και τα αντίστοιχα μηνύματα M1, M2 από τα οποία προέκυψαν έχουν το ίδιο μήκος, οπότε η επίθεση μπορεί να γίνει χρησιμοποιώντας τα Known Plaintext Statistics. Αν όμως δεν έχουν το ίδιο μήκος τότε δεν μπορούμε να καταφύγουμε σε αυτή την τεχνική καθώς θα έχουμε τμήματα είτε του C1 είτε του C2 που δεν υπάρχει αντίστοιχα το C2 ή το C1.

Οπότε διακρίνουμε τις εξής περιπτώσεις:

- A. Αν τα ciphertexts C1, C2 έχουν το ίδιο μήκος τότε μπορούμε να βρούμε τα κλειδιά και τα μηνύματα που ικανοποιούν τις απαιτήσεις.
- B. Αν τα μηνύματα C1 και C2 έχουν διαφορετικό μήκος, τότε μπορούμε για όσο τμήμα των δύο ciphertexts είναι κοινό να εργαστούμε όπως και στην περίπτωση A. Για το υπόλοιπο τμήμα η επίθεση μπορεί να γίνει εξετάζοντας μόνο το ciphertext που είναι μεγαλύτερο και δεχόμενοι πως το άλλο ciphertext μας είναι αδιάφορο. Μπορούμε να θεωρήσουμε πως μπορεί να έχει οποιαδήποτε τιμή του οποίου το αντίστοιχο plaintext πληροί τις προδιαγραφές.

Για πληροφοριακούς λόγους στην εμφάνιση των αποτελεσμάτων θα εμφανίσουμε τόσο το κλειδί ή τους συνδυασμούς των κλειδιών που θα μπορούσαν να κρυπτογράφησουν τα ciphertexts, αλλά και τους συνδυασμούς των τιμών που θα μπορούσαν να έχουν τα μηνύματα.

Ακολούθως, για το κάθε κλειδί που βρήκαμε πως αποκρυπτογραφεί τα ciphertexts θα εμφανίσουμε και τα αντίστοιχα plaintexts που κρυπτογραφούνται στα ciphertexts.

Τα κλειδιά επειδή ενδέχεται να είναι οποιοσδήποτε αριθμός χωράει σε ένα byte και θέλουμε να το εμφανίσουμε, έχει προτιμηθεί η εμφάνιση θα γίνει σε binary μορφή προκειμένου να είναι εύκολη και η επαλήθευση για την αποκρυπτογράφηση του ciphertext.

### Περιγραφή της δομής

Η συγκεκριμένη επίθεση έχει υλοποιηθεί σε γλώσσα προγραμματισμού python (έκδοση 3), ομαδοποιώντας τις διαδικασίες σε διάφορα αρχεία, τα οποία έχουν την ακόλουθη χρήση.

Στο αρχείο `Senario_1_2.py` βρίσκεται ο κώδικας που υλοποιεί τον βασικό σκελετό του αλγόριθμου της επίθεσης. Αυτό το αρχείο πρέπει να εκτελεστεί προκειμένου να γίνει η επίθεση.

Στο αρχείο `myfunctions2.py` βρίσκεται ο κώδικας που υλοποιεί τις λεπτομέρειες του αλγόριθμου της επίθεσης. Έτσι σπάμε τον κώδικα που υλοποιεί τις λεπτομέρειες ώστε να μπορούμε να τον επαναχρησιμοποιήσουμε είτε κάνοντας δύο ciphertexts, είτε πολλαπλά.

Στο αρχείο `binaryfunc.py` βρίσκεται βοηθητικός κώδικας που έχει περιγραφεί στην επίθεση του πρώτου σεναρίου.

## Ανάλυση της υλοποίησης

Για την αποκωδικοποίηση των μηνυμάτων θα ξεκινήσουμε με την αποκωδικοποίηση του κάθε χαρακτήρα των ciphertexts ξεχωριστά. Θα κάνουμε την αποκωδικοποίηση όλων των χαρακτήρων που βρίσκονται στη θέση 0, μετά όλων των χαρακτήρων που βρίσκονται στη θέση 1 και ούτω καθ' εξής.

Στις διαφάνειες αναφέρεται ο τρόπος επίθεσης Known Plaintext Statistics ο οποίος αναφέρει πως το:

$$C1 \oplus C2 = m1 \oplus k \oplus m2 \oplus k = m1 \oplus m2$$

Αναφέρεται στην ίδια διαφάνεια πως πρέπει να συνδυαστούν οι πιο πιθανές τιμές των  $m1$ ,  $m2$  ώστε να προκύψει το  $C1 \oplus C2$

Αυτό στην περίπτωσή μας θα μπορούσε να αναλυθεί στο

$$C1_i \oplus C2_i = m1_i \oplus k_i \oplus m2_i \oplus k_i = m1_i \oplus m2_i$$

Οπότε πρέπει να ψάχουμε για τα  $m1_i$ ,  $m2_i$  που να δίνουν  $C1_i \oplus C2_i$

Όπου  $i$ , η  $i$ -οστή θέση του ciphertext και του μηνύματος αντίστοιχα.

Αυτή η προσέγγιση θεωρούμε πως είναι ικανοποιητική για δύο ciphertexts ίσου μήκους αλλά δεν διευκολύνει όταν έχουμε πολλαπλά,  $n$ , ciphertexts, καθώς αυξάνει αρκετά το πλήθος των πράξεων.

Για παράδειγμα όταν έχουμε 2 ciphertexts του ενός χαρακτήρα οι πράξεις που πρέπει να εκτελεστούν είναι  $10 \times 10 = 100$  αφού ο κάθε πιθανός χαρακτήρας του plaintext μπορεί να είναι κάποιος από τους 0,1,2,3,4,5,6,7,8,9 (10 χαρακτήρες σύνολο).

Αν έχουμε τρία ciphertexts τότε πρέπει να κάνουμε αυτές τις πράξεις  $\times 2$ , αφού έχουμε πλέον 2 ζεύγη ciphertexts που πρέπει να ελέγχουμε. Οπότε οι πράξεις είναι  $2 \times 10 \times 10 = 200$ .

Στη λύση της άσκησης προτιμήσαμε να ακολουθήσουμε μια διαφορετική αντιμετώπιση. Αντί να ψάχνουμε ποια  $m1 \oplus m2$  δίνουν  $C1 \oplus C2$ , να ψάχουμε ποιο κλειδί  $k \oplus C1$  δίνει κάποιο  $m1$  που να πληροί τις προδιαγραφές, δηλαδή να είναι κάποιο από τα 0,1,2,3,4,5,6,7,8,9.

Στη συνέχεια ελέγχουμε εάν το  $k \oplus C2$  δίνει και αυτό κάποιο αποδεκτό  $m2$  που είναι επίσης αποδεκτό.

Ο έλεγχος γίνεται για όλα τα ciphertexts που έχουμε πριν καταλήξουμε στο συμπέρασμα πως το κλειδί  $k$  είναι αποδεκτό γιατί μπορεί να κρυπτογραφήσει όλα τα ciphertexts. Αν έστω και ένα ciphertext δεν δίνει το αποδεκτό τότε απορρίπτουμε το κλειδί.

Εάν το κλειδί συμπεράνουμε πως είναι αποδεκτό γιατί μπορεί να αποκρυπτογραφήσει τα ciphertexts σε αποδεκτά μηνύματα, τότε μπορούμε να κρατήσουμε και τις τιμές των μηνυμάτων που βρήκαμε ώστε να τα συνδυάσουμε στα πλήρη μηνύματα.

Έτσι έχουμε για δύο ciphertexts να κάνουμε  $2 \times 16 = 32$  φορές τις πράξεις, σε σύγκριση με 100 που ήταν με την προηγούμενη αντιμετώπιση

Αν έχουμε τρία ciphertexts θα γίνουν  $3 \times 16 = 48$  φορές τις πράξεις, σε σύγκριση με 200 ήταν αντίστοιχα με την προηγούμενη αντιμετώπιση.

Επιπλέον εάν κάποιο ciphertext είναι μεγαλύτερο από τα υπόλοιπα μπορούμε να ακολουθήσουμε την ίδια λογική, ελέγχοντας μόνο για τα αυτό το μεγαλύτερο ciphertext.

Δηλαδή δεν εξαρτόμαστε από το πλήθος των ciphertexts και δεν απαιτείται η ύπαρξη του λάχιστον δύο.

Μένει ακόμα να γίνει ο συνδυασμός των μηνυμάτων που βρήκαμε πως αντιστοιχεί σε κάθε θέση για να έχουμε το πλήρες μήνυμα που αντιστοιχεί στο κάθε ciphertext.

Τέλος επιλέχθηκε να γίνει συμπλήρωση στα μηνύματα που είναι μικρότερου μήκους, με κενό χαρακτήρα ‘’ (το κενό string - όχι το διάστημα space), προκειμένου να μπορεί να φανεί ακριβώς ποιος χαρακτήρας βρέθηκε και σε ποιο ciphertext αντιστοιχεί, όπως σε περίπτωση που έχουμε 4 ciphertexts μεγεθών 2, 6, 3 και 6 χαρακτήρων αντίστοιχα.

Σε κάποια τέτοια περίπτωση ο κώδικας μας όταν εκτελεστεί θα εμφανίσει σε κάποιο σημείο πως για το 5o byte των ciphertexts έχουν βρεθεί για παράδειγμα οι χαρακτήρες [ ' ', 6, ' ', 2] το οποίο σημαίνει πως για το 1o και το 3o ciphertext δεν αντιστοιχεί κάποιος χαρακτήρας καθώς είναι μικρότερου μεγέθους.

Προτυπώθηκε αυτή η αντιμετώπιση της εμφάνισης του “ αντί να μην εμφανίζεται τίποτα π.χ. [ , 6, , 2] προκειμένου να είναι καλύτερη η στοίχιση των αποτελεσμάτων όταν εμφανίζονται στην οθόνη και όταν εκτυπώνονται.

Τέλος αναφέρω πως η επιλογή αυτής της εμφάνισης των αποτελεσμάτων έναντι της εμφάνισης μόνο όσων χαρακτήρων βρίσκονται στα ciphertexts μεγαλύτερου μεγέθους π.χ. [6, 2] έγινε για να είναι ορατό και σε ποιο ciphertext αντιστοιχεί ο κάθε χαρακτήρας.

Αναλυτικά σχόλια υπάρχουν και μέσα στα αρχεία του πηγαίου κώδικα για αναφορά.

Στο αρχείο `Senario_1_2.py` ο κώδικας εκτελεί τα εξής βήματα:

Στάδιο προετοιμασίας:

1. Δίνεται ποιοι είναι οι αποδεκτοί χαρακτήρες από τους οποίους αποτελείται κάποιο plaintext
2. Ορίζεται πόσο μήκος θα έχουν τα ciphertexts. Θα χρησιμοποιήσουμε μέχρι τέσσερα, καθώς αυτό το πλήθος μπορεί να δώσει απαντήσεις για όλες τις ζητούμενες περιπτώσεις της άσκησης.  
Τα μήκη των ciphertexts αποθηκεύονται σε έναν πίνακα στη μεταβλητή `plain_text_lengths`
3. Δημιουργούνται 4 plaintexts αποτελούμενα από δεκαδικά ψηφία αντίστοιχων μεγεθών, όσο έχουν δηλωθεί στο `plain_text_lengths`. Η δημιουργία τους γίνεται με ψευδοτυχαίο τρόπο χρησιμοποιώντας τη συνάρτηση `GenerateRandomDecimalString` και αποθηκεύονται στις μεταβλητές `P1`, `P2`, `P3`, `P4` αντίστοιχα.

Η επιλογή του πλήθους των 4 ciphertexts έγινε για τον εξής λόγο:  
Μπορούμε με 2 ciphertexts ίσου μήκους να κάνουμε την αρχική επίθεση. Προσθέτοντας και ένα 3o ciphertext ίσου μήκους, μπορούμε να δούμε αν έχουμε αλλαγή στα αποτελέσματα που βρίσκουμε και αν μπορούμε να είμαστε πιο συγκεκριμένοι στο να βρούμε πιο είναι το κλειδί κρυπτογράφησης και ποια ήταν τα αρχικά μηνύματα.

Προσθέτοντας και ένα 4o ciphertext μεγαλύτερου μεγέθους μπορούμε να δούμε την αλλαγή των αποτελεσμάτων που οφείλεται κυρίως στο επιπλέον τμήμα του μεγαλύτερου ciphertext.

4. Δημιουργείται ένα κλειδί για την κρυπτογράφηση των plaintexts με ψευδοτυχαίο τρόπο χρησιμοποιώντας τη συνάρτηση `GenerateRandomKey`, μήκους όσο το μεγαλύτερο μήνυμα και αποθηκεύεται στη μεταβλητή `encryptionKey`.
5. Δημιουργούνται τα ciphertexts και αποθηκεύονται στις μεταβλητές `Cipher1`, `Cipher2`, `Cipher3`, `Cipher4` αντίστοιχα
6. Για ευκολότερη επεξεργασία σπάει τα ciphertexts που έχουμε σε binary strings στις μεταβλητές `Cipher1`, `Cipher2`, `Cipher3`, `Cipher4` σε πίνακες από strings που το καθένα κρατάει μόνο ένα byte.

Αυτοί οι πίνακες αποθηκεύονται στις μεταβλητές `C1`, `C2`, `C3` και `C4` αντίστοιχα

Στάδιο εκτέλεσης της επίθεσης:

Σε αυτό το σημείο είναι που ξεκινάει η «πραγματική» υλοποίηση της επίθεσης που ζητάει η άσκηση.

Τα plaintexts που είναι αποθηκευμένα στις μεταβλητές `P1`, `P2`, `P3`, `P4` τα θεωρούμε άγνωστα. Δεν χρησιμοποιούνται πουθενά, παρά έχουμε κρατήσεις τις τιμές τους προκειμένου να τα εμφανίσουμε για πληροφόρηση και να κατανοούμε τα αποτελέσματα καλύτερα.

Έχουμε σαν γνωστά στοιχεία:

- Τα ciphertexts που είναι αποθηκευμένα στις μεταβλητές `Cipher1`, `Cipher2`, `Cipher3` και `Cipher4`.

Ο κώδικας, εκτελεί την επίθεση (όταν έχουμε δύο ciphertexts) με τα ακόλουθα βήματα:

7. Για πληροφοριακούς λόγους εμφανίζει τα plaintexts που έδωσαν τα ciphertexts στα οποία θα εκτελέσουμε την επίθεση προσπαθώντας να βρούμε πιθανά plaintexts (κάποιο από τα οποία θα είναι και τα αρχικά που χρησιμοποιήσαμε).
8. Εμφανίζει τα ciphertexts που έχουμε και στα οποία θα εκτελέσουμε την επίθεση.
9. Αποθηκεύει τα ciphertexts που έχουμε στις μεταβλητές σε έναν πίνακα στη μεταβλητή `CiphersToCheck`, προκειμένου να γίνει η επίθεση δυναμικά ανεξαρτήτως του πλήθους των ciphertexts
10. Ψάχνει να βρει ποια είναι τα 4 αριστερά bits του κάθε byte του κλειδιού που έχει κωδικοποιήσει τα μηνύματα. Κάθε byte των ciphertexts έχει κωδικοποιηθεί με το ίδιο byte του κλειδιού, οπότε αφού γνωρίζουμε πως το αρχικό μήνυμα είναι κάποιο δεκαδικό ψηφίο μπορούμε εύκολα να υπολογίσουμε τα most significant nibbles (4 αριστερά bits) των bytes του κλειδιού, ώστε να περιορίσουμε το εύρος αναζήτησης χρησιμοποιώντας τη συνάρτηση `FindAllKeysLeftNibble`. Παρότι τα κλειδιά που θα κάνουν την αποκρυπτογράφηση είναι πολλά, τα most significant nibbles όλων αυτών είναι τα ίδια, οπότε δεν χρειάζεται να τα υπολογίσουμε όλα, αρκεί να υπολογίσουμε ένα.

11. Βρίσκει τα πιθανά κλειδιά και τους πιθανούς συνδυασμούς χαρακτήρων των μηνυμάτων χρησιμοποιώντας τη συνάρτηση `FindAllKeysAndMessageCharacters`.

12. Εμφανίζει τους πιθανούς συνδυασμούς μηνυμάτων και το αντίστοιχους συνδυασμούς κλειδιών που θα μπορούσαν να τα κρυπτογραφήσουν στα ciphertexts που έχουμε.

Πληροφοριακά υπολογίζει και πόσοι είναι αυτοί οι διαφορετικοί συνδυασμοί.

Για την επίθεση όταν έχουμε τρία ciphertexts εκτελούνται πάλι τα βήματα 7 έως 12.

Το ίδιο ισχύει και για την επίθεση όταν έχουμε 4 ciphertexts, που εκτελούνται πάλι τα βήματα 7 έως 12

Περιγραφή του αρχείου `myfunctions2.py`

Στο αρχείο `myfunctions2.py` αρχικά δηλώνονται οι παρακάτω μεταβλητές για τις οποίες έχει χρησιμοποιηθεί η εξής ονοματολογία

Το όνομα των μεταβλητών ξεκινάει με δύο χαρακτήρες `_` υπονοώντας πως προορίζονται για εσωτερική χρήση ενός του αρχείου. Θα χρησιμοποιηθούν προσπαθώντας να μειώσουμε τον αριθμό των παραμέτρων που απαιτούνται να δώσουμε στις συναρτήσεις που χρησιμοποιούνται υλοποιώντας τις λεπτομέρειες του αλγόριθμου.

Οι μεταβλητές αυτές είναι:

`_results_count` η οποία κρατάει το μέγιστο πλήθος των αποτελεσμάτων που θα εμφανιστούν. Αυτή αρχικοποιείται στην τιμή 2 (να έρχονται μέχρι 2 συνδυασμοί αποτελεσμάτων). Εάν δεν θέλουμε να περιορίσουμε τα αποτελέσματα τότε θα πρέπει να αλλάξουμε την τιμή 2 σε `sys.maxsize + 1` όπως αναφέρεται και στα σχόλια του πηγαίου κώδικα

`_keys` η οποία κρατάει όλους τους δυνατούς συνδυασμούς των κλειδιών που θα βρεθούν

`_CiphersToCheck` η οποία κρατάει όλους τους ciphertexts που πρέπει να αποκρυπτογραφηθούν

`_ValidDigits` η οποία κρατάει όλους τους επιτρεπτούς χαρακτήρες από τους οποίους μπορούν να αποτελούνται τα μηνύματα.

`_KeysLeftNibble` η οποία κρατάει σε πίνακα τα most significant nibbles των bytes του κλειδιού που χρησιμοποιήθηκε για την κρυπτογράφηση.

`_checkCiphers` η οποία κρατάει σε πίνακα τους ciphertexts που θα χρησιμοποιηθούν στην αποκρυπτογράφηση.

Η συνάρτηση `FindOneKeyLeftNibble` είναι βοηθητική και παίρνει σαν παράμετρο ένα byte ενός plaintext και ένα byte ενός ciphertext και επιστρέφει την τιμή που θα πρέπει να έχει το most significant nibble (bits 7-4) ενός byte του κλειδιού που μπορεί να χρησιμοποιηθεί για την κρυπτογράφηση του message byte στο cipher byte. Το αποτέλεσμα επιστρέφεται σαν αριθμός.

Η συνάρτηση `FindAllKeysLeftNibble` είναι βοηθητική και παίρνει σαν παράμετρο κάποιο ciphertext. Επιστρέφει σε έναν πίνακα όλα τα most significant nibbles (MSN) των bytes του κλειδιού που μπορούν να κρυπτογραφήσουν κάποιο μήνυμα στο ciphertext. Για την εύρεση των MSN χρειάζεται να γίνει έλεγχος μόνο με τα 4 αριστερά bits κάποιου αριθμητικού ψηφίου. Στην ASCII κωδικοποίηση

όλοι οι αριθμητικοί χαρακτήρες έχουν την binary αναπαράσταση 0011xxxx, όπου το xxxx μπορεί να είναι από 0000 έως 1001. Αρκεί για να βρούμε το MSN να κάνουμε έλεγχο με οποιαδήποτε από αυτές τις τιμές και έχει γίνει η επιλογή του 00110000.

Η συνάρτηση `FindAllKeysAndMessageCharactersAtIndex` παίρνει σαν παράμετρο το `byteIndex` που είναι η θέση του byte που θα γίνει η αποκρυπτογράφηση. Ψάχνει και βρίσκει όλα τα πιθανά κλειδιά που μπορούν να αποκρυπτογραφήσουν το αντίστοιχο byte όλων των ciphers στη θέση `byteIndex`.

Η συνάρτηση `FindAllKeysAndMessageCharacters` παίρνει σαν παράμετρο το `ValidDigits` που είναι οι αποδεκτοί χαρακτήρες που μπορεί να έχουν τα plaintexts, το `KeysLeftNibble` που είναι ο πίνακας με τα most significant nibbles των bytes του κλειδιού και το `checkCiphers` που είναι ο πίνακας με τα Ciphers που θα αποκρυπτογραφηθούν.

Η συνάρτηση `CombineSubKeys`, η οποία καλείται από την `CombineKeys` αλλά και από τον εαυτό της αναδρομικά και δημιουργεί τον συνδυασμό των κλειδιών από όλους τους συνδυασμούς που μπορεί να έχουν τα bytes των κλειδιών. Παίρνει σαν παραμέτρους το πλήθος των κλειδιών στην παράμετρο `countsets`, το τρέχον αριθμό set στην παράμετρο `setNo`, και το προηγούμενο κλειδί που έχει υπολογιστεί στην παράμετρο `keystring`.

Ταυτόχρονα με τους συνδυασμούς των κλειδιών δίνει και τους συνδυασμούς των μηνυμάτων που αποκρυπτογραφούνται με αυτά τα κλειδιά.

Οι ενέργειες που εκτελούνται είναι οι ακόλουθες:

- Γίνεται αρχικά έλεγχος αν έχει εμφανιστεί το πλήθος των συνδυασμών που επιθυμεί ο χρήστης. Αν έχει ολοκληρωθεί δεν συνεχίζει με άλλες ενέργειες.
- Εάν δεν έχει εμφανιστεί το πλήθος των συνδυασμών γίνεται έλεγχος εάν έχουμε και άλλο τμήμα του κλειδιού που πρέπει να συνδυαστεί.
- Προσθέτει το τμήμα του κλειδιού που έχει στο προηγούμενο τμήμα κλειδιού
- Καλεί στον εαυτό της για το επόμενο set κλειδιών, προκειμένου να συμπληρωθούν και τα υπόλοιπα bytes του κλειδιού.
- Εάν δεν υπάρχει άλλο τμήμα που μένει να συνδυαστεί, έχουμε δηλαδή ολόκληρο κάποιον συνδυασμό κλειδιού εμφανίζει ποιο είναι το κλειδί που μπορεί να κάνει την αποκρυπτογράφηση.
- Γίνεται μετατροπή του κλειδιού που έχουμε σε binary string σε αριθμητική μορφή ώστε να μπορούμε να κάνουμε την αποκρυπτογράφηση των ciphertexts
- Για κάθε ciphertext κάνει την αποκρυπτογράφηση ένα-ένα byte του cipher με το αντίστοιχο byte του κλειδιού και το προσθέτει στο αντίστοιχο μήνυμα plaintext
- Εμφανίζει το μήνυμα (που αντιστοιχεί στο ciphertext) και συνεχίζει με το επόμενο ciphertext
- Μειώνει κατά ένα το πλήθος των συνδυασμών που απομένουν να εμφανιστούν

Η συνάρτηση `CombineKeys` παίρνει σαν παράμετρο τα κλειδιά που έχουν υπολογιστεί και δημιουργεί το πρώτο τμήμα των κλειδιών και καλεί τη συνάρτηση `CombineSubKeys` προκειμένου να συμπληρωθούν και οι υπόλοιποι συνδυασμοί.

Οι ενέργειες που εκτελούνται είναι οι ακόλουθες:

- Αποθηκεύει τα κλειδιά εσωτερικά προκειμένου να μην χρειάζεται να τα περνάμε σαν παράμετρο στις άλλες συναρτήσεις που καλούνται
- Υπολογίζει πόσο είναι το μήκος του κλειδιού που θα δημιουργηθεί
- Για κάθε συνδυασμό του κλειδιού, δημιουργεί την binary μορφή του τρέχοντος byte του κλειδιού από το set και καλεί την `CombineSubKeys` με το επόμενο set ώστε να συμπληρωθούν και τα υπόλοιπα bytes.

Η συνάρτηση `PrintKeyCombinations` είναι αυτή που καλείται από τον κεντρικό αλγόριθμο στο αρχείο `Scenario_1_2.py` και παίρνει σαν παράμετρο τα κλειδιά αποκρυπτογράφησης που έχουμε βρει και αποθηκεύει εσωτερικά τους ciphers προκειμένου να μην χρειάζεται να τους περνάει σαν παράμετρο στις συναρτήσεις που χρησιμοποιεί. Καλεί τη συνάρτηση `SetMaximumResultsFromUser` ώστε να περιοριστούν τα αποτελέσματα που θα εμφανίζονται και στη συνέχεια καλεί τη συνάρτηση `CombineKeys` ώστε να εμφανιστούν οι συνδυασμοί των κλειδιών που αποκρυπτογραφούν τα ciphertexts αλλά και τα μηνύματα που αντιστοιχούν στην αποκρυπτογράφηση με το κάθε κλειδί.

Η συνάρτηση `SetMaximumResultsFromUser` παίρνει σαν παράμετρο τα set κλειδιών που έχουν υπολογιστεί. Οι εργασίες που εκτελεί είναι οι εξής:

- Υπολογίζει πόσοι είναι οι δυνατοί συνδυασμοί που μπορούν να παραχθούν και το αποθηκεύει τη μεταβλητή `combinationsNo`.
- Ελέγχει εάν ο χρήστης έχει δώσει κατά την εκτέλεση του κώδικα σαν παράμετρο πόσα αποτελέσματα θέλει να εμφανίζονται και τα αποθηκεύει στη μεταβλητή `_results_count`.
- Ελέγχει εάν το πλήθος που έχει η μεταβλητή `combinationsNo` είναι έως 16 και εάν είναι τότε αυτό το πλήθος, επειδή είναι μικρό, το αποθηκεύει στη μεταβλητή `_results_count`.
- Εάν τα αποτελέσματα δεν είναι λιγότερα από 16 αλλά είναι λιγότερα από αυτά που έχει ζητήσει ο χρήστης να εμφανίζονται τότε στη μεταβλητή `_results_count` αποθηκεύονται όσα έχουν υπολογιστεί, ανεξαρτήτως πόσα περισσότερα ζήτησε ο χρήστης.

Αναλυτικότερα σχόλια για τον τρόπο λειτουργίας των συναρτήσεων υπάρχουν μέσα στα αρχεία του πηγαίου κώδικα.

## Αποτελέσματα εκτέλεσης δεύτερου σεναρίου

Επεξήγηση των αποτελεσμάτων:

Στα αποτελέσματα βλέπουμε πως εμφανίζονται οι γραμμές:

Possible M1, M2 at byte 0 : [6, 9][7, 8][8, 7][9, 6]

Possible M1, M2 at byte 1 : [9, 6][8, 7][7, 8][6, 9]

Αντό σημαίνει πως το byte στη θέση 0 του μηνύματος M1 μπορεί να έχει κάποια από τις τιμές:  
6 ή 7 ή 8 ή 9

και το byte στην θέση 0 του μηνύματος M2 μπορεί να έχει κάποια από τις τιμές 9 ή 8 ή 7 ή 6

Όταν το byte στη θέση 0 του μηνύματος M1 έχει τιμή 6, τότε το αντίστοιχο byte στη θέση 0 του μηνύματος M2 έχει τιμή 9 (σημειώνονται με καφέ χρώμα)

Όταν το byte στη θέση 0 του μηνύματος M1 έχει τιμή 7, τότε το αντίστοιχο byte στη θέση 0 του μηνύματος M2 έχει τιμή 8 (έχουν μαύρο χρώμα)

Αν πάμε τώρα στο byte στη θέση 1 των μηνυμάτων έχουμε αντίστοιχα:

Όταν το byte στη θέση 1 του μηνύματος M1 είναι 9, τότε το byte στη θέση 1 του μηνύματος M2 θα έχει τιμή 6 (έχουν μπλε χρώμα)

Όταν το byte στη θέση 1 του μηνύματος M1 είναι 8, τότε το byte στη θέση 1 του μηνύματος M2 θα έχει τιμή 7 (έχουν πράσινο χρώμα)

Τα αποτελέσματα των συνδυασμών βγαίνουν ως εξής:

Παίρνουμε οποιονδήποτε συνδυασμό τιμών του byte 0 (έστω τον 1<sup>ο</sup>) και έχουμε τα εξής μηνύματα:

M1 = 6

M2 = 9

Συμπληρώνουμε οποιονδήποτε συνδυασμό τιμών του byte 1 (έστω τον 2<sup>ο</sup>) και έχουμε τα εξής μηνύματα:

M1 = 68

M2 = 97

Αν στο byte 0 διαλέξουμε κάποιον άλλο συνδυασμό τιμών (έστω τον 3<sup>ο</sup>) τότε συμπληρώνοντας τον 2<sup>ο</sup> συνδυασμό τιμών του byte 1 θα πάρουμε τα μηνύματα:

M1 = 88

M2 = 77

Στη συνέχεια της επίθεσης που βάζουμε και τρίτο ciphertext βλέπουμε πως τα αποτελέσματα τώρα σε κάθε byte έχουν τριάδες τιμών. Για παράδειγμα:

Possible M1, M2, M3 at byte 0 : [6, 9, 3][7, 8, 2]

Όταν βάλουμε και τέταρτο ciphertext το οποίο έχει μεγαλύτερο μήκος τότε θα δούμε τα αποτελέσματα να εμφανίζονται ως εξής:

Possible M1, M2, M3, M4 at byte 0 : [6, 9, 3, 4][7, 8, 2, 5]

Βλέπουμε πως είναι συμπληρωμένες και οι τέσσερις τιμές για τα bytes 0. Αν όμως φτάσουμε σε κάποια θέση που κάποια από τα ciphertexts είναι μικρότερου μήκους θα δούμε τα αποτελέσματα να είναι στην εξής μορφή: (έχουμε αλλάξει τη στοίχιση για να είναι πιο κατανοητό στο χαρτί που έχει περιορισμένο πλάτος)

Possible M1, M2, M3, M4 at byte 4 :     ['', '', '', 8]['', '', '', 9]  
                                      ['', '', '', 2]['', '', '', 3]  
                                      ['', '', '', 0]['', '', '', 1]  
                                      ['', '', '', 6]['', '', '', 7]  
                                      ['', '', '', 4]['', '', '', 5]

Αντό σημαίνει πως στο byte 4 τιμή μπορεί να γίνει συνδυασμός κάποιας τιμής μόνο για το μήνυμα M4. Τα μηνύματα M1, M2, M3 είναι μικρότερου μήκους και δεν έχουν κάποια τιμή στο συγκεκριμένο byte.

Ενδεικτική Εκτέλεση 1 (με μήκος των plaintexts 2, 2, 2, 6 αντίστοιχα)

---

```
P1 = 76
P2 = 89
C1 = 01000010 00001111
C2 = 01001101 00000000
Possible M1, M2 at byte 0 : [6, 9][7, 8][8, 7][9, 6]
Possible key at byte 0 : 01110100 01110101 01111010 01111011
```

```
Possible M1, M2 at byte 1 : [9, 6][8, 7][7, 8][6, 9]
Possible key at byte 1 : 00110110 00110111 00111000 00111001
```

Will show up to 16/16 possible key and messages combinations.

```
Possible key      = 01110100 00110110
```

```
Possible M1 for C1 = 69
```

```
Possible M2 for C2 = 96
```

```
Possible key      = 01110100 00110111
```

```
Possible M1 for C1 = 68
```

```
Possible M2 for C2 = 97
```

```
Possible key      = 01110100 00111000
```

```
Possible M1 for C1 = 67
```

```
Possible M2 for C2 = 98
```

```
Possible key      = 01110100 00111001
```

```
Possible M1 for C1 = 66
```

```
Possible M2 for C2 = 99
```

Possible key = 01110101 00110110  
Possible M1 for C1 = 79  
Possible M2 for C2 = 86

Possible key = 01110101 00110111  
Possible M1 for C1 = 78  
Possible M2 for C2 = 87

Possible key = 01110101 00111000  
Possible M1 for C1 = 77  
Possible M2 for C2 = 88

Possible key = 01110101 00111001  
Possible M1 for C1 = 76  
Possible M2 for C2 = 89

Possible key = 01111010 00110110  
Possible M1 for C1 = 89  
Possible M2 for C2 = 76

Possible key = 01111010 00110111  
Possible M1 for C1 = 88  
Possible M2 for C2 = 77

Possible key = 01111010 00111000  
Possible M1 for C1 = 87  
Possible M2 for C2 = 78

Possible key = 01111010 00111001  
Possible M1 for C1 = 86  
Possible M2 for C2 = 79

Possible key = 01111011 00110110  
Possible M1 for C1 = 99  
Possible M2 for C2 = 66

Possible key = 01111011 00110111  
Possible M1 for C1 = 98  
Possible M2 for C2 = 67

Possible key = 01111011 00111000  
Possible M1 for C1 = 97  
Possible M2 for C2 = 68

Possible key = 01111011 00111001  
Possible M1 for C1 = 96  
Possible M2 for C2 = 69

---

-----  
-----  
P1 = 76  
P2 = 89  
P3 = 24  
C1 = 01000010 00001111

```
C2 = 01001101 00000000
C3 = 01000111 00001101
Possible M1, M2, M3 at byte 0 : [6, 9, 3][7, 8, 2]
Possible key at byte 0 : 01110100 01110101
```

```
Possible M1, M2, M3 at byte 1 : [7, 8, 5][6, 9, 4]
Possible key at byte 1 : 00111000 00111001
```

Will show up to 4/4 possible key and messages combinations.

```
Possible key      = 01110100 00111000
Possible M1 for C1 = 67
Possible M2 for C2 = 98
Possible M3 for C3 = 35
```

```
Possible key      = 01110100 00111001
Possible M1 for C1 = 66
Possible M2 for C2 = 99
Possible M3 for C3 = 34
```

```
Possible key      = 01110101 00111000
Possible M1 for C1 = 77
Possible M2 for C2 = 88
Possible M3 for C3 = 25
```

```
Possible key      = 01110101 00111001
Possible M1 for C1 = 76
Possible M2 for C2 = 89
Possible M3 for C3 = 24
```

---

```
P1 = 76
P2 = 89
P3 = 24
P4 = 515524
C1 = 01000010 00001111
C2 = 01001101 00000000
C3 = 01000111 00001101
C4 = 01000000 00001000 00000000 01011011 01101010 01000101
Possible M1, M2, M3, M4 at byte 0 : [6, 9, 3, 4][7, 8, 2, 5]
Possible key at byte 0 : 01110100 01110101
```

```
Possible M1, M2, M3, M4 at byte 1 : [7, 8, 5, 0][6, 9, 4, 1]
Possible key at byte 1 : 00111000 00111001
```

```
Possible M1, M2, M3, M4 at byte 2 : [', ', ', ', 0][', ', ', ', 1][', ', ', ', 2][', ', ',
', ', 3][', ', ', ', 4][', ', ', ', 5][', ', ', ', 6][', ', ', ', 7][', ', ', ', 8][', ', ',
', ', 9]
Possible key at byte 2 : 00110000 00110001 00110010 00110011 00110100 00110101 00110110
00110111 00111000 00111001
```

Possible M1, M2, M3, M4 at byte 3 : [', ', ', ', 9][', ', ', ', 8][', ', ', ', 3][', ',  
' , 2][', ', ', 1][', ', ', 0][', ', ', 7][', ', ', ', 6][', ', ', ', 5][', ',  
' , 4]

Possible key at byte 3 : 01100010 01100011 01101000 01101001 01101010 01101011 01101100  
01101101 01101110 01101111

Possible M1, M2, M3, M4 at byte 4 : [', ', ', ', 8][', ', ', ', 9][', ', ', ', 2][', ',  
' , 3][', ', ', 0][', ', ', 1][', ', ', 6][', ', ', ', 7][', ', ', ', 4][', ',  
' , 5]

Possible key at byte 4 : 01010010 01010011 01011000 01011001 01011010 01011011 01011100  
01011101 01011110 01011111

Possible M1, M2, M3, M4 at byte 5 : [', ', ', ', 5][', ', ', ', 4][', ', ', ', 7][', ',  
' , 6][', ', ', 1][', ', ', 0][', ', ', 3][', ', ', ', 2][', ', ', ', 9][', ',  
' , 8]

Possible key at byte 5 : 01110000 01110001 01110010 01110011 01110100 01110101 01110110  
01110111 01111100 01111101

Will show up to 5/40000 possible key and messages combinations.

Possible key = 01110100 00111000 00110000 01100010 01010010 01110000

Possible M1 for C1 = 67

Possible M2 for C2 = 98

Possible M3 for C3 = 35

Possible M4 for C4 = 400985

Possible key = 01110100 00111000 00110000 01100010 01010010 01110001

Possible M1 for C1 = 67

Possible M2 for C2 = 98

Possible M3 for C3 = 35

Possible M4 for C4 = 400984

Possible key = 01110100 00111000 00110000 01100010 01010010 01110010

Possible M1 for C1 = 67

Possible M2 for C2 = 98

Possible M3 for C3 = 35

Possible M4 for C4 = 400987

Possible key = 01110100 00111000 00110000 01100010 01010010 01110011

Possible M1 for C1 = 67

Possible M2 for C2 = 98

Possible M3 for C3 = 35

Possible M4 for C4 = 400986

Possible key = 01110100 00111000 00110000 01100010 01010010 01110100

Possible M1 for C1 = 67

Possible M2 for C2 = 98

Possible M3 for C3 = 35

Possible M4 for C4 = 400981

Ενδεικτική Εκτέλεση 2 (με μήκος των plaintexts 6, 6, 6, 5 αντίστοιχα)

-----  
P1 = 521663

P2 = 741292

```
C1 = 01010000 01110100 01000001 01110010 01111011 01011000
C2 = 01010010 01110010 01000001 01110110 01110100 01011001
Possible M1, M2 at byte 0 : [0, 2][1, 3][2, 0][3, 1][4, 6][5, 7][6, 4][7, 5]
Possible key at byte 0 : 01100000 01100001 01100010 01100011 01100100 01100101 01100110
01100111

Possible M1, M2 at byte 1 : [4, 2][5, 3][6, 0][7, 1][0, 6][1, 7][2, 4][3, 5]
Possible key at byte 1 : 01000000 01000001 01000010 01000011 01000100 01000101 01000110
01000111

Possible M1, M2 at byte 2 : [1, 1][0, 0][3, 3][2, 2][5, 5][4, 4][7, 7][6, 6][9, 9][8, 8]
Possible key at byte 2 : 01110000 01110001 01110010 01110011 01110100 01110101 01110110
01110111 01111000 01111001

Possible M1, M2 at byte 3 : [2, 6][3, 7][0, 4][1, 5][6, 2][7, 3][4, 0][5, 1]
Possible key at byte 3 : 01000000 01000001 01000010 01000011 01000100 01000101 01000110
01000111

Possible M1, M2 at byte 4 : [9, 6][8, 7][7, 8][6, 9]
Possible key at byte 4 : 01000010 01000011 01001100 01001101

Possible M1, M2 at byte 5 : [8, 9][9, 8][0, 1][1, 0][2, 3][3, 2][4, 5][5, 4][6, 7][7, 6]
Possible key at byte 5 : 01100000 01100001 01101000 01101001 01101010 01101011 01101100
01101101 01101110 01101111

Will show up to 5/204800 possible key and messages combinations.

Possible key      = 01100000 01000000 01110000 01000000 01000010 01100000
Possible M1 for C1 = 041298
Possible M2 for C2 = 221669

Possible key      = 01100000 01000000 01110000 01000000 01000010 01100001
Possible M1 for C1 = 041299
Possible M2 for C2 = 221668

Possible key      = 01100000 01000000 01110000 01000000 01000010 01101000
Possible M1 for C1 = 041290
Possible M2 for C2 = 221661

Possible key      = 01100000 01000000 01110000 01000000 01000010 01101001
Possible M1 for C1 = 041291
Possible M2 for C2 = 221660

Possible key      = 01100000 01000000 01110000 01000000 01000010 01101010
Possible M1 for C1 = 041292
Possible M2 for C2 = 221663
```

-----  
-----

```
P1 = 521663
P2 = 741292
P3 = 397579
C1 = 01010000 01110100 01000001 01110010 01111011 01011000
C2 = 01010010 01110010 01000001 01110110 01110100 01011001
```

```
C3 = 01010110 01111111 01000111 01110001 01111010 01010010
Possible M1, M2, M3 at byte 0 : [0, 2, 6][1, 3, 7][2, 0, 4][3, 1, 5][4, 6, 2][5, 7, 3][6,
4, 0][7, 5, 1]
Possible key at byte 0 : 01100000 01100001 01100010 01100011 01100100 01100101 01100110
01100111

Possible M1, M2, M3 at byte 1 : [2, 4, 9][3, 5, 8]
Possible key at byte 1 : 01000110 01000111

Possible M1, M2, M3 at byte 2 : [1, 1, 7][0, 0, 6][3, 3, 5][2, 2, 4][5, 5, 3][4, 4, 2][7,
7, 1][6, 6, 0]
Possible key at byte 2 : 01110000 01110001 01110010 01110011 01110100 01110101 01110110
01110111

Possible M1, M2, M3 at byte 3 : [2, 6, 1][3, 7, 0][0, 4, 3][1, 5, 2][6, 2, 5][7, 3, 4][4,
0, 7][5, 1, 6]
Possible key at byte 3 : 01000000 01000001 01000010 01000011 01000100 01000101 01000110
01000111

Possible M1, M2, M3 at byte 4 : [9, 6, 8][8, 7, 9][7, 8, 6][6, 9, 7]
Possible key at byte 4 : 01000010 01000011 01001100 01001101

Possible M1, M2, M3 at byte 5 : [8, 9, 2][9, 8, 3][2, 3, 8][3, 2, 9]
Possible key at byte 5 : 01100000 01100001 01101010 01101011

Will show up to 5/16384 possible key and messages combinations.

Possible key      = 01100000 01000110 01110000 01000000 01000010 01100000
Possible M1 for C1 = 021298
Possible M2 for C2 = 241669
Possible M3 for C3 = 697182

Possible key      = 01100000 01000110 01110000 01000000 01000010 01100001
Possible M1 for C1 = 021299
Possible M2 for C2 = 241668
Possible M3 for C3 = 697183

Possible key      = 01100000 01000110 01110000 01000000 01000010 01101010
Possible M1 for C1 = 021292
Possible M2 for C2 = 241663
Possible M3 for C3 = 697188

Possible key      = 01100000 01000110 01110000 01000000 01000010 01101011
Possible M1 for C1 = 021293
Possible M2 for C2 = 241662
Possible M3 for C3 = 697189

Possible key      = 01100000 01000110 01110000 01000000 01000011 01100000
Possible M1 for C1 = 021288
Possible M2 for C2 = 241679
Possible M3 for C3 = 697192
```

-----  
-----

```
P1 = 521663
P2 = 741292
P3 = 397579
P4 = 49904
C1 = 01010000 01110100 01000001 01110010 01111011 01011000
C2 = 01010010 01110010 01000001 01110110 01110100 01011001
C3 = 01010110 01111111 01000111 01110001 01111010 01010010
C4 = 01010001 01111111 01001001 01110100 01111001
Possible M1, M2, M3, M4 at byte 0 : [0, 2, 6, 1][1, 3, 7, 0][2, 0, 4, 3][3, 1, 5, 2][4, 6,
2, 5][5, 7, 3, 4][6, 4, 0, 7][7, 5, 1, 6]
Possible key at byte 0 : 01100000 01100001 01100010 01100011 01100100 01100101 01100110
01100111

Possible M1, M2, M3, M4 at byte 1 : [2, 4, 9, 9][3, 5, 8, 8]
Possible key at byte 1 : 01000110 01000111

Possible M1, M2, M3, M4 at byte 2 : [1, 1, 7, 9][0, 0, 6, 8]
Possible key at byte 2 : 01110000 01110001

Possible M1, M2, M3, M4 at byte 3 : [2, 6, 1, 4][3, 7, 0, 5][0, 4, 3, 6][1, 5, 2, 7][6, 2,
5, 0][7, 3, 4, 1][4, 0, 7, 2][5, 1, 6, 3]
Possible key at byte 3 : 01000000 01000001 01000010 01000011 01000100 01000101 01000110
01000111

Possible M1, M2, M3, M4 at byte 4 : [7, 8, 6, 5][6, 9, 7, 4]
Possible key at byte 4 : 01001100 01001101

Possible M1, M2, M3, M4 at byte 5 : [8, 9, 2, ''][9, 8, 3, ''][2, 3, 8, ''][3, 2, 9, '']
Possible key at byte 5 : 01100000 01100001 01101010 01101011

Will show up to 5/2048 possible key and messages combinations.

Possible key      = 01100000 01000110 01110000 01000000 01001100 01100000
Possible M1 for C1 = 021278
Possible M2 for C2 = 241689
Possible M3 for C3 = 697162
Possible M4 for C4 = 19945

Possible key      = 01100000 01000110 01110000 01000000 01001100 01100001
Possible M1 for C1 = 021279
Possible M2 for C2 = 241688
Possible M3 for C3 = 697163
Possible M4 for C4 = 19945

Possible key      = 01100000 01000110 01110000 01000000 01001100 01101010
Possible M1 for C1 = 021272
Possible M2 for C2 = 241683
Possible M3 for C3 = 697168
Possible M4 for C4 = 19945

Possible key      = 01100000 01000110 01110000 01000000 01001100 01101011
Possible M1 for C1 = 021273
Possible M2 for C2 = 241682
Possible M3 for C3 = 697169
Possible M4 for C4 = 19945
```

Possible key = 01100000 01000110 01110000 01000000 01001101 01100000  
Possible M1 for C1 = 021268  
Possible M2 for C2 = 241699  
Possible M3 for C3 = 697172  
Possible M4 for C4 = 19944

### Παρατήρηση από την εκτέλεση της επίθεσης

Όταν έχουμε εκτελέσει την επίθεση με 2 ciphertexts ίσου μεγέθους παίρνουμε κάποια αποτελέσματα.

Όταν επαναλάβουμε την επίθεση με επιπλέον το 3ο ciphertext ίσου μεγέθους μπορούμε να συγκρίνουμε τα αποτελέσματα και να δούμε τη μείωση των πιθανών συνδυασμών και τη καλύτερη σύγκλιση προς τα αρχικά plaintexts.

Συμπεραίνουμε ότι πως όσο προσθέτουμε στην επίθεση και άλλα ciphertexts ίσου μεγέθους τότε η πιθανότητα να βρούμε το συγκεκριμένο κλειδί με το οποίο γίνεται η κρυπτογράφηση και ποια είναι τα συγκεκριμένα μηνύματα αυξάνεται.

Όταν προσθέσουμε επιπλέον ciphertexts μεγαλύτερου μηνύματος τότε αυξάνονται κατά πολύ το πλήθος των δυνατών συνδυασμών αφού υπάρχει κάποιο τμήμα των ciphertexts, του 4<sup>ο</sup> ciphertext στον συγκεκριμένο κώδικα που πρακτικά ελέγχεται μόνο του και πρακτικά για κάθε byte του ciphertext μπορούν να υπάρχουν και 10 πιθανοί χαρακτήρες με αντίστοιχο byte στο κλειδί που να το κρυπτογραφούν.

Αυτή η παρατήρηση μπορεί εύκολα να επιβεβαιωθεί εάν αντί να διαλέξουμε σαν 4<sup>ο</sup> ciphertext κάποιο τυχαίο, πάρουμε ως πρώτο τμήμα του κάποιο από τα υπόλοιπα τρία ciphertexts και προσθέσουμε επιπλέον ένα μικρό τμήμα που προέρχεται από κρυπτογράφηση κάποιου μικρού μήκους plaintext.