2015

1η προαιρετική εργασία

ΘΕΜΕΛΙΩΣΕΙΣ ΚΡΥΠΤΟΓΡΑΦΙΑΣ

ΑΝΑΓΝΩΣΤΟΥ ΑΝΤΩΝΙΟΣ – 2268 – anagnoad@csd.auth.gr
ΛΑΣΚΑΡΙΔΗΣ ΣΤΕΦΑΝΟΣ – 2315 – laskstef@csd.auth.gr

Περιεχόμενα

Εισαγωγή 2
Θέμα 1ο 3
Υποερωτήμα i) 3
Υποερώτημα ii) 3
Υποερώτημα iii) 4
Υποερώτημα iv) 4
Θέμα 2ο 4
Θέμα 3ο 5
Θέμα 4ο 6
Θέμα 5ο 6
Θέμα 6ο 6
Θέμα 7ο 7
Θέμα 8ο 7
Θέμα 9ο 7
Υποερώτημα i) 7
Υποερώτημα ii) 9
Βιβλιογραφία 13
Παράρτημα 14
Οδηγίες εκτέλεσης 14
Ex2_code.py 14
Ex3_code.py 15
Ex4_code.py 19
Ex5_code.py 20
Ex7_code.py 24
Ex8_code.py 25

Εισαγωγή

Η παρούσα εργασία αποσκοπεί στην εξοικείωση με τις βασικές έννοιες της κρυπτογραφίας, εξάσκηση με τους αλγόριθμους κρυπτογράφησης καθώς και στην εφαρμογή μεθόδων κρυπτάναλυσης για την παραβίαση κρυπτοσυστημάτων.

Για όλες τις προγραμματιστικές ασκήσεις που ακολουθούν, χρησιμοποιήθηκαν τα ακόλουθα:

- Python 3
- pycrypt library
- Sage math software
- OS: Ubuntu 14.04, 64-bit
- Github repository: https://github.com/stevelaskaridis/cryptography_project1

Επισυναπτόμενα αρχεία κατά την παράδοση της εργασίας θα βρείτε:

- Αρχεία python:
    - ex2_code.py
    - ex3_code.py
    - ex4_code.py
    - ex5_code.py
    - ex7_code.py
    - ex8_code.py
- Αρχείο excel (vigenere.xlsx)
- english.txt
- password.txt
- test_zip.zip
- test_zip
- Report_2315_2268.pdf

Θέμα 1ο

Υποερωτήμα i)

Αρχή Kerchoff:

Σύμφωνα με την αρχή του Kerkhoff, η ασφάλεια ενός κρυπτοσυστήματος θα πρέπει να βασίζεται εξ'ολοκλήρου στη μυστικότητα του κλειδιού και στην τυχαιότητα των γεννητριών τυχαίων bitstrings. Αυτό σημαίνει πως η μυστικότητα του κώδικα και της υλοποίησης του κρυπτοσυστήματος δεν θα πρέπει να προσδίδει το στοιχείο της ασφάλειας στο σύστημά μας.

Ακόμη και αν ο επιτιθέμενος γνωρίζει τους αλγορίθμους υλοποίησης του συστήματος, δε θα πρέπει να μπορεί να "σπάσει" το κρυπτοσύστημα λόγω αυτής του της γνώσης.

Πλεονεκτήματα αυτού του κανόνα αποτελούν:

1. Ο κώδικας υλοποίησης του κρυπτοσυστήματος μπορεί να είναι open-source, με συνέπεια την παρακολούθηση και εξέτασή του από μεγαλύτερη μερίδα προγραμματιστών. ("Every bug is shallow, given enough eyes")

2. Ακόμη και αν οι γνώστες της υλοποίησης του κρυπτοσυστήματος αποτελούν τμήμα της επίθεσης προς το σύστημα, η γνώση δεν προσδίδει καμία πληροφορία σε σχέση με το κλειδί.

3. Στα πλαίσια ανοχής παραβιάσεων, αν η λειτουργία και ασφάλεια του συστήματος βασίζεται στη μυστικότητα του αλγορίθμου υλοποίησης, αποτελεί ενδεχόμενο failure point.

Υποερώτημα ii)

Έστω (E, D) ένα κρυπτοσύστημα επί του συνόλου (K, M, I). Λέμε ότι το σύστημα (E, D) έχει τέλεια ασφάλεια, αν :

$$\forall\, m_0, m_1 \in m\ (\text{ίδιου μήκους})\ \text{και}\ c \in l\ \text{έχουμε}:$$
$$P(E(k, m_0) = c) = P(E(k, m_1) = c),$$
$$\text{όπου}\ k \overset{R}{\leftarrow} K$$

Δηλαδή αν η Εύα κλέψει το c, τότε δε γνωρίζει αν αυτό προήλθε από το m_0 ή το m_1.

http://pileas.csd.auth.gr/file.php/141/cource-1-2-3.pdf [94]

Κρυπτοσυστήματα με τέλεια ασφάλεια αποτελούν:

- Vigénere
- OTP (One Time Pad)

http://crypto.stackexchange.com/questions/3896/simply-put-what-does-perfect-secrecy-means

Υποερώτημα iii)

Όχι. Στην περίπτωση που κρυπτογραφηθούν δύο μηνύματα m1, m2 με το ίδιο κλειδί k, και τα αντίστοιχα ciphertexts είναι c1 και c2, τότε ισχύει το παρακάτω:

```
c1 xor c2 = (m1 xor k) xor (m2 xor k) = m1 xor m2
```

Αν τα μηνύματα είναι σε φυσική γλώσσα και ακολουθείται κάποια κατανομή χρήσης των γραμμάτων, τότε μπορούν να βρεθούν τα μηνύματα. Τα μηνύματα, ωστόσο, πρέπει να έχουν πλεονασμό.

Υποερώτημα iv)

Η κατάσταση λειτουργίας ενός block cipher αναφέρεται στον τρόπο με τον οποίο ο αλγόριθμος κρυπτογράφησης χρησιμοποιεί και συνδυάζει τα τμήματα στα οποία έχει χωρίσει το μήνυμα προς κρυπτογράφηση για την παραγωγή του ciphertext.

Κατάσταση λειτουργίας ECB (Electronic Codebook)

Η απλούστερη κατάσταση λειτουργίας. Το μήνυμα διαιρείται σε blocks, καθένα από τα οποία κρυπτογραφείται αυτόνομα.

Το μειονέκτημα αυτής της μεθόδου είναι ότι επιτρέπει την ανίχνευση patterns στο κρυπτογραφημένο ciphertext, καθώς δεδομένου ότι παρόμοια blocks του μηνύματος κρυπτογραφούνται με το ίδιο κλειδί, έχουν και παρόμοιο παραγόμενο ciphertext.

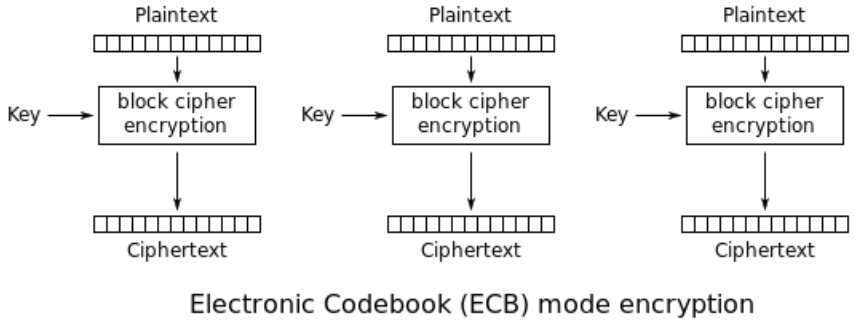

FIGURE 1 ECB DIAGRAM

Θέμα 2ο

Το τμήμα κρυπτογράφησης του αλγορίθμου RC4 υλοποιήθηκε σε python3, ο κώδικας του οποίου επισυνάπτεται στο παράρτημα της παρούσας εργασίας. Χρησιμοποιώντας τον αλγόριθμο αυτό, και κρυπτογραφώντας το μήνυμα "Never send a human to do a machine s job" με κλειδί την λέξη "MATRIX" λαμβάνουμε την ακόλουθη έξοδο, σε δεκαεξαδική μορφή:

"ed b1 10 24 57 11 6d af d1 d8 33 7f 7f 32 ba 8c ae 01 d3 75 a8 fb db 6c 89 dd 43 88 92 20 4e 1b aa 1f e6 11 5f 48 5f c7"

Το ακόλουθο αποτέλεσμα επαληθεύτηκε και από εξωτερική υλοποίηση του αλγόριθμου, του οποίου ο σύνδεσμος βρίσκεται στην βιβλιογραφία.

Θέμα 3ο

Η αποκρυπτογράφηση του μηνύματος έγινε με τα εξής βήματα:

- Εύρεση του μήκους του κλειδιού (14), με βάση τον αλγόριθμο που προτείνεται στις διαφάνειες του μαθήματος.
- Διαίρεση του μηνύματος σε 14 ομάδες, οι χαρακτήρες των οποίων θα έχουν κρυπτογραφηθεί με την ίδια μετατόπιση.
- Πραγματοποίηση λεξικογραφικής ανάλυσης, ώστε να διαπιστωθεί ποιο γράμμα βρίσκεται στην συγκεκριμένη θέση του κλειδιού.

Για την εκτέλεση του αλγορίθμου απαιτείται python3, ενώ τα αποτελέσματα είναι τα ακόλουθα:

Κλειδί: ΤΣΑΠΛΙΝΤΣΑΠΛΙΝ

Μήνυμα: ΛΥΠΑΜΑΙ ΑΛΛΑ ΔΕΝ ΘΕΛΩ ΝΑ ΓΙΝΩ ΑΥΤΟΚΡΑΤΟΡΑΣ. ΔΕΝ ΕΙΝΑΙ ΔΙΚΗ ΜΟΥ ΥΠΟΘΕΣΗ. ΔΕΝ ΘΕΛΩ ΟΥΤΕ ΝΑ ΒΑΣΙΛΕΨΩ ΟΥΤΕ ΝΑ ΚΑΤΑΚΤΗΣΩ ΚΑΝΕΝΑΝ. ΘΑ ΗΘΕΛΑ ΝΑ ΒΟΗΘΗΣΩ ΟΛΟ ΤΟΝ ΚΟΣΜΟ ΑΝ ΜΠΟΡΟΥΣΑ, ΕΒΡΑΙΟΥΣ ΧΡΙΣΤΙΑΝΟΥΣ ΜΑΥΡΟΥΣ ΛΕΥΚΟΥΣ. ΟΛΟΙ ΕΠΙΘΥΜΟΥΜΕ ΤΗΝ ΑΛΛΗΛΕΓΓΥΗ ΑΥΤΗ. ΕΙΝΑΙ Η ΦΥΣΗ ΤΩΝ ΑΝΘΡΩΠΩΝ ΝΑ ΖΟΥΜΕ ΜΕ ΤΗΝ ΕΥΤΥΧΙΑ ΤΩΝ ΑΛΛΩΝ ΚΑΙ ΟΧΙ ΜΕ ΤΗ ΔΥΣΤΥΧΙΑ ΤΟΥΣ. ΔΕΝ ΘΕΛΟΥΜΕ ΟΥΤΕ ΝΑ ΜΙΣΟΥΜΕ, ΟΥΤΕ ΝΑ ΠΕΡΙΦΡΟΝΟΥΜΕ. ΣΤΟΝ ΚΟΣΜΟ ΑΥΤΟΝ ΥΠΑΡΧΕΙ ΧΩΡΟΣ ΓΙΑ ΤΟΝ ΚΑΘΕΝΑ. Η ΚΑΛΗ ΓΗ ΕΙΝΑΙ ΠΛΟΥΣΙΑ ΚΑΙ ΜΠΟΡΕΙ ΝΑ ΠΑΡΕΧΕΙ ΓΙΑ ΟΛΟΥΣ. Η ΖΩΗ ΜΠΟΡΕΙ ΝΑ ΕΙΝΑΙ ΕΛΕΥΘΕΡΗ ΚΑΙ ΩΡΑΙΑ, ΑΛΛΑ ΧΑΣΑΜΕ ΑΥΤΟ ΤΟ ΜΟΝΟΠΑΤΙ. Η ΠΛΕΟΝΕΞΙΑ ΔΗΛΗΤΗΡΙΑΣΕ ΤΙΣ ΨΥΧΕΣ ΤΩΝ ΑΝΘΡΩΠΩΝ, ΑΝΥΨΩΣΕ ΤΟΥΣ ΦΡΑΓΜΟΥΣ ΤΟΥ ΜΙΣΟΥΣ, ΜΑΣ ΚΑΤΑΔΙΚΑΣΕ ΣΤΗ ΔΥΣΤΥΧΙΑ ΚΑΙ ΣΤΗ ΣΦΑΓΗ. ΟΡΙΣΑΜΕ ΤΗΝ ΤΑΧΥΤΗΤΑ ΑΛΛΑ ΚΛΕΙΣΤΗΚΑΜΕ ΣΤΟΝ ΕΑΥΤΟ ΜΑΣ. Η ΕΚΒΙΟΜΗΧΑΝΙΣΗ ΠΡΟΣΦΕΡΕΙ ΑΦΘΟΝΙΑ, ΑΛΛΑ ΜΑΣ ΕΧΕΙ ΑΦΗΣΕΙ ΣΕ ΕΝΔΕΙΑ. Η ΕΠΙΣΤΗΜΗ ΜΑΣ ΕΚΑΝΕ ΚΥΝΙΚΟΥΣ, Η ΕΥΦΥΙΑ ΜΑΣ, ΣΚΛΗΡΟΥΣ ΚΑΙ ΑΞΕΣΤΟΥΣ. ΣΚΕΦΤΟΜΑΣΤΕ ΠΟΛΥ ΚΑΙ ΑΙΣΘΑΝΟΜΑΣΤΕ ΕΛΑΧΙΣΤΑ. ΠΕΡΙΣΣΟΤΕΡΟ ΚΑΙ ΑΠΟ ΤΙΣ ΜΗΧΑΝΕΣ. ΧΡΕΙΑΖΟΜΑΣΤΕ ΤΗΝ ΑΝΘΡΩΠΙΑ ΠΙΟ ΠΟΛΥ ΑΠΟ ΤΗΝ ΕΠΙΔΕΞΙΟΤΗΤΑ. ΧΡΕΙΑΖΟΜΑΣΤΕ ΤΗΝ ΚΑΛΟΣΥΝΗ ΚΑΙ ΤΗΝ ΕΥΓΕΝΕΙΑ. ΧΩΡΙΣ ΑΥΤΕΣ ΤΙΣ ΑΡΕΤΕΣ Η ΒΙΑ ΘΑ ΚΥΡΙΑΡΧΗΣΕΙ ΣΤΗ ΖΩΗ ΚΑΙ ΟΛΑ ΘΑ ΧΑΘΟΥΝ. ΤΟ ΑΕΡΟΠΛΑΝΟ ΚΑΙ ΤΟ ΡΑΔΙΟΦΩΝΟ ΜΑΣ ΕΦΕΡΑΝ ΠΙΟ ΚΟΝΤΑ. Η ΙΔΙΑ Η ΦΥΣΗ ΑΥΤΩΝ ΤΩΝ ΕΦΕΥΡΕΣΕΩΝ ΔΙΑΛΑΛΕΙ ΤΗΝ ΚΑΛΟΣΥΝΗ ΤΩΝ ΑΝΘΡΩΠΩΝ. ΔΙΑΛΑΛΕΙ ΤΗΝ ΠΑΓΚΟΣΜΙΑ ΑΔΕΛΦΟΣΥΝΗ, ΤΗΝ ΕΝΟΤΗΤΑ ΟΛΩΝ ΜΑΣ. ΑΚΟΜΑ ΚΙ ΑΥΤΗ ΤΗ ΣΤΙΓΜΗ, Η ΦΩΝΗ ΜΟΥ ΦΤΑΝΕΙ ΣΤΑ ΑΥΤΙΑ ΕΚΑΤΟΜΜΥΡΙΩΝ ΑΝΘΡΩΠΩΝ, ΑΠΕΛΠΙΣΜΕΝΩΝ ΓΥΝΑΙΚΩΝ, ΠΑΙΔΙΩΝ ΠΟΥ ΕΙΝΑΙ ΘΥΜΑΤΑ ΕΝΟΣ ΣΥΣΤΗΜΑΤΟΣ ΠΟΥ ΞΕΡΕΙ ΜΟΝΟ ΝΑ ΒΑΣΑΝΙΖΕΙ ΚΑΙ ΝΑ ΦΥΛΑΚΙΖΕΙ ΑΘΩΟΥΣ ΑΝΘΡΩΠΟΥΣ. ΣΕ ΑΥΤΟΥΣ ΠΟΥ ΜΕ ΑΚΟΥΝΕ ΛΕΩ: "ΜΗΝ ΑΠΕΛΠΙΖΕΣΤΕ! Η ΤΩΡΙΝΗ ΜΑΣ ΔΥΣΤΥΧΙΑ ΔΕΝ ΕΙΝΑΙ ΠΑΡΑ ΤΟ ΠΕΡΑΣΜΑ ΤΗΣ ΠΛΕΟΝΕΞΙΑΣ ΚΑΙ ΤΗΣ ΣΚΛΗΡΟΤ(ΗΤΑΣ)

Τα γράμματα που βρίσκονται εντός τον παρενθέσεων δεν αποτελούν μέρος του αποκρυπτογραφημένου μηνύματος αλλά παρατίθενται για την πληρότητα του κειμένου.

Θέμα 4ο

Για την αποκρυπτογράφηση του μηνύματος εφαρμόσαμε έναν brute-force αλγόριθμο που δοκιμάζει μετατόπιση για n = 1...23 χαρακτήρες. Το μόνο από τα μηνύματα που είχαν κάποιο νόημα ήταν το ακόλουθο:

«ΜΗΔΕΙΣ ΑΓΕΩΜΕΤΡΗΤΟΣ ΕΙΣΙΤΩ ΜΟΥ ΤΗΝ ΣΤΕΓΗΝ»

Θέμα 5ο

Το avalanche effect αναφέρεται στο γεγόνος ότι μικρές αλλαγές σε bits ενός plaintext θα πρέπει να οδηγούν σε μεγάλες αλλαγές bits στα αντίστοιχα ciphertexts. Με τον τρόπο αυτό, διασφαλίζεται η ασφάλεια ενός κρυπτοσυστήματος αφού δεν μπορεί να πραγματοποιηθεί με εύκολο τρόπο κρυπτανάλυση βάσει απόστασης από το πρωτότυπο του κείμενου.

Στον κώδικα της άσκησης αυτής, που επισυνάπτεται στο παράρτημα, δημιουργούμε τυχαία ζευγάρια κειμένων (128 bits το καθένα) και αντίγραφα αυτών μεταλλαγμένα κατά 1 bit. Χρησιμοποιώντας τον αλγόριθμο κρυπτογράφησης AES από την βιβλιοθήκη pycrypto κρυπτογραφούμε τα ζεύγη αυτά και μετράμε την απόσταση [1]σε bits.

Κατά μέσο όρο τα μηνύματα αυτά διαφέρουν κατά τουλάχιστον 62 bits από το αρχικά μηνύματα.

Ένα πρόβλημα το οποίο αντιμετωπίσαμε στον κώδικα που παρατίθεται στην εργασία είναι ότι η κρυπτογράφηση των πρωτότυπων κειμένων δίνει στην έξοδο διαφορετικό μήκος bitstrings (<128). Αυτό είχε σαν αποτέλεσμα να μην καθίσταται δυνατή η σύγκριση με τα ciphertexts των μεταλλαγμένων μηνυμάτων. Για να το αντιμετωπίσουμε, πραγματοποιούμε left padding με μηδενικά στην αρχή των ciphertexts.

Θέμα 6ο

Το μήνυμα αυτό εξαιτίας του μικρού του μεγέθους δεν επιδέχεται κάποιας λεξικογραφικής ανάλυσης, ώστε να εξάγουμε κάποια πληροφορία για το κλειδί. Ακόμα και αν το κλειδί περιέχει μόνο τους χαρακτήρες K, E, Y, ένας brute force αλγόριθμος θα απαιτούσε 3^21 επαναλήψεις για όλους τους πιθανούς συνδυασμούς του κλειδιού.

Στην προσέγγιση μας, κάναμε χρήση ενός ευριστικού αλγόριθμου (heuristic), στον οποίο ενεργούμε ως εξής:

1. Δημιουργούμε έναν βοηθητικό πίνακα με τις μετατοπίσεις του κάθε γράμματος του μηνύματος δεδομένου του κάθε χαρακτήρα του κλειδιού (K, E, Y)
2. Ξεκινάμε με ένα από τα τρία πιθανά γράμματα για την πρώτη θέση του μηνύματος.
3. Συνεχίζουμε στις επόμενες θέσεις, και κοιτάμε, αν ο συνδυασμός του γράμματος αυτού με όλα τα προηγούμενα, αποτελεί κάποια έγκυρη λέξη ή φράση στην αγγλική γλώσσα.

[1] **απόσταση bits**: το πλήθος των διαφορετικών bits μεταξύ των δύο ciphertexts.

4. Αν δεν υπάρχει κάποια τέτοια πιθανή φράση, τότε γίνεται κλάδεμα (pruning) όλου του "κλαδιού", και πραγματοποιούμε backtracking, δοκιμάζοντας κάποιον άλλο χαρακτήρα.

Με τον τρόπο αυτό, καταφέραμε να βρούμε με σχετική ευκολία το μήνυμα, το οποιό ήταν το:

"Peace begins with a smile"

Επισυνάπτεται στα προς παράδοση αρχεία και ένα βοηθητικό αρχείο Excel που χρησιμοποιήσαμε για την εξαγωγή του μηνύματος.

Θέμα 7ο

Η υλοποίηση του αλγόριθμου για την άσκηση αυτή κάνει χρήση ενός dictionary με αγγλικές λέξεις, και δοκιμάζει διαδοχικά όλες τις λέξεις μέχρι να πραγματοποιηθεί σωστά η εξαγωγή του αρχείου.

Αξίζει να σημειωθεί όμως ότι υπάρχει "bug" στη βιβλιοθήκη του zipfile όπου με διαφορετικό κωδικό του σωστού, μπορεί να γίνει extract του αρχείου με μηδενικό μέγεθος. Στην προτεινόμενη λύση, αγνοούνται όλα τα exceptions και γίνονται οι απαραίτητοι έλεγχοι για το πότε το extraction έχει πραγματοποιηθεί επιτυχώς.

Το κλειδί είναι η λέξη "**secret**" και τα περιεχόμενα του zip είναι

"In the Beginning...Was the Command Line

by Neal Stephenson ".

Θέμα 8ο

Για την άσκηση αυτή, παράξαμε όλες τις δυνατές εξάδες ακεραίων μεταξύ 0 και 9: δηλαδή τα πιθανά κλειδιά 000000 - 999999. Έπειτα, δοθέντων του salt και του κρυπτογραφημένου κωδικού (από το αρχείο /etc/shadow που δόθηκε), μπορέσαμε να συγκρίνουμε το hashing του καθένος απ'αυτά, μέχρι να βρούμε το σωστό κλειδί, το οποίο είναι το 961616.

Για την πιο γρήγορη εκτέλεση του προγράμματος, το iteration γίνεται με φορά από το τέλος προς στην αρχή.

Έχοντας το κλειδί, συνδεθήκαμε μέσω ssh και εισάγαμε τα ονόματα μας στο τέλος του αρχείου my_name.

```
ssh test@sage.csd.auth.gr

echo “Antonios Anagnostou 2268, Stefanos Laskaridis 2315 OMADA” >> my_name
```

Θέμα 9ο

Υποερώτημα i)

Ο αλγόριθμος Blowfish είναι ένας συμμετρικός αλγόριθμος κρυπτογράφησης τμήματος που προτάθηκε το 1993 από τον Bruce Scneier και έχει αρκετά μεγάλη διάδοση στη σημερινή εποχή. Παρέχει αποδεκτό ρυθμό κρυπτογράφησης και δεν έχει υπάρξει κάποια μέθοδος κρυπτανάλυσης που να έχει δημοσιευτεί μέχρι σήμερα. Αποτελεί εναλλακτική στο δημοφιλέστατο AES (Advanced Encryption Standard), με τον δεύτερο να έχει μεγαλύτερο μερίδιο χρήσης.

Γενικά, ο Blowfish θεωρείται πως είναι ένας αρκετά γρήγορος αλγόριθμος κρυπτογράφησης, εκτός από την περίπτωση αλλαγής κλειδιών. Κάθε νέο κλειδί χρειάζεται μία προεπεξεργασία αντίστοιχη με την κρυπτογράφηση κειμένου 4KB, το οποίο θεωρείται εξαιρετικά αργό σε σύγκριση με άλλους αλγορίθμους. Ωστόσο, το γεγονός αυτό θα μπορούσε να λειτουργήσει και θετικά: για παράδειγμα, το OpenBSD χρησιμοποιεί έναν αλγόριθμο εμπνευσμένο από το Blowfish που αξιοποιεί αυτή την καθυστέρηση, για να αμυνθεί απέναντι σε dictionary attacks. Άλλες μετεξελίξεις του BlowFish αποτελούν οι αλγόριμοι Twofish και Threefish.

Περιγραφή λειτουργίας:

Ο αλγόριθμος, όπως ισχύει και στον DES και τον AES, κάνει χρήση S-boxes. Πιο συγκεκριμένα, ο Blowfish έχει block μεγέθους 64-bit και χρησιμοποιεί το σχήμα Feistel 16 φάσεων. Το μήκος του κλειδιού μεταβάλλεται από 32 μέχρι 448 bits, ενώ χρησιμοποιεί μεγάλα S-boxes που εξαρτώνται από το κλειδί.

Αναλυτικότερα, ο αλγόριθμος χρησιμοποιεί δύο subkey arrays: ένα array P 18 θέσεων, καθώς και 4 S-boxes των 256-bit. Τα S-boxes έχουν εισόδους των 8 bit ενώ το παραγόμενο αποτέλεσμα είναι 32 bit. Σε κάθε φάση χρησιμοποιείται μία μόνο θέση από το P και μετά τον τελευταίο γύρο, κάθε data block χωρίζεται στη μέση και πραγματοποιείται η πράξη XOR με μία από τις δύο εναπομείναντες θέσεις του πίνακα P.

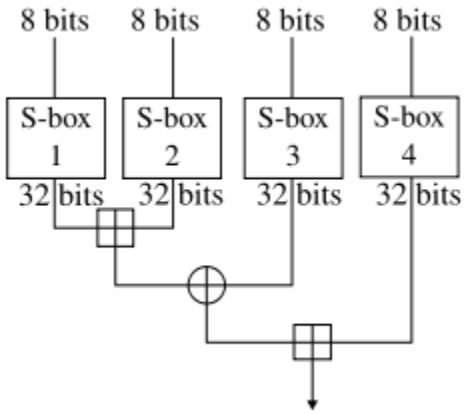

FIGURE 2 BLOWFISH DIAGRAM

Έχοντας πει τα παραπάνω, μπορούμε να περιγράψουμε και την συνάρτηση F του Blowfish. Η συνάρτηση αυτή χωρίζει μία είσοδο 32-bit σε 4 τμήματα των 8 bits και τοποθετεί τα bits αυτά σαν είσοδο στα S-boxes. Οι έξοδοι προστίθενται με αριθμητική modulo 2^32 και αφού πραγματοποιηθεί λογική αποκλειστική διάζευξη (XOR) παράγεται το τελικό output των 32-bit.

Η αποκρυπτογράφηση λειτουργεί με αντίστοιχο τρόπο, με την μόνη διαφορά πως οι 18 θέσεις του P (P1, P2, ..., P18) βρίσκονται σε αντίστροφη σειρά.

Όσον αφορά στην διαδικασία παραγωγής των subkeys, ο Blowfish αρχικοποιεί το P και τα S-boxes με τις δεκαεξαδικές τιμές των ψηφίων του άρρητου αριθμού π, αφού δεν υπάρχει κάποια αλληλουχία στους αριθμούς αυτούς. Έπειτα πραγματοποιείται XOR του κάθε byte του κλειδιού (επαναλαμβάνοντας αυτό αν χρειαστεί), με τις θέσεις του P. Στη συνέχεια, κρυπτογραφείται ένα μηδενικό block 64 bit και το παραγόμενο κρυπτοκείμενο αντικαθιστά τα P1 και P2. Το ίδιο κρυπτοκείμενο κρυπτογραφείται ξανά, με νέα subkeys, και το νέο output αντικαθιστά τα P3 και P4. Η διαδικασία αυτή συνεχίζει, αντικαθιστώντας ολόκληρο το array P και τα S-boxes. Συνολικά δηλαδή, ο αλγόριθμος Blowfish χρειάζεται 521 περάσματα για να παράξει όλα τα subkeys.

Τρωτά σημεία:

Ο Blowfish θεωρείται ευάλωτος σε επιθέσεις όπου χρησιμοποιούνται "αδύναμα" κλειδιά. Οι χρήστες θα πρέπει να αποφεύγουν ορισμένες κλάσεις κλειδιών που είναι γνωστές για την αδυναμίες τους στη χρήση τους με τον Blowfish ή να στραφούν προς εναλλακτικές όπως ο AES, ο ChaCha (που χρησιμοποιείται από τη Google) ή ο Salsa20.

Υποερώτημα ii)

Το bitcoin αποτελεί ένα είδος κατανεμημένου ψηφιακού συστήματος πληρωμών ανοιχτού κώδικα το οποίο εφευρέθηκε από τον/την/τους Satoshi Nakamoto. Στη βάση του, το bitcoin είναι μία δομή η οποία συνδέει λογαριασμούς με υπόλοιπο[2] χρημάτων, ένα αντίγραφο του οποίου βρίσκεται αποθηκευμένο σε κάθε κόμβο του δικτύου.

Σημαντικό οικονομικό στοιχείο στο υπόλοιπο που συνδέεται με καθένα από τους λογαριασμούς είναι το γεγονός πως αποκτούν αξία επειδή οι ίδιοι οι χρήστες του bitcoin είναι διατεθειμένοι να ανταλλάξουν προϊόντα και υπηρεσίες έναντι bitcoin.

Η διαφορά με μία συμβατική τράπεζα είναι πως η δομή είναι απολύτως αποκεντρωμένη, και ο καθένας μέσα στο δίκτυο του bitcoin γνωρίζει για κάθε άλλη συναλλαγή, όχι μόνο για τις προσωπικές του.

Βασικά σημεία ασφαλείας που τηρούνται είναι:

- η ακεραιότητα της συναλλαγής μέσω ψηφιακών υπογραφών, referenced transactions και block chain
- η προστασία από το φαινόμενου του double spending[3]
- η ανωνυμία των χρηστών

Συναλλαγές

[2] Στην ουσία το υπόλοιπο βρίσκεται με την μορφή links σε προηγούμενες συναλλαγές (transactions). Η εγκυρότητα αυτών των συναλλαγών ελέγχεται από κάθε κόμβο του δικτύου. (Transaction chain)

[3] Αναφέρεται στην επαναχρησιμοποίηση bitcoins που έχουν ξοδευτεί.

1. Οι νέες συναλλαγές γίνονται broadcast σε όλους τους κόμβους του δικτύου
2. Κάθε κόμβος μαζεύει τις συναλλαγές σε ένα block.
3. Κάθε κόμβος δουλεύει στο να βρει τη λύση σε ένα πρόβλημα hashing[4].
4. Όταν ο κόμβος βρει τη λύση, την κάνει broadcast σε όλους τους υπόλοιπους κόμβους.
5. Οι κόμβοι δέχονται τη συναλλαγή, μόνο αν όλες οι προηγούμενες συναλλαγές είναι έγκυρες και δεν υπάρχει double-spending.
6. Οι κόμβοι εκφράζουν την αποδοχή του block δουλεύοντας πάνω στο επόμενο block, χρησιμοποιώντας το hash του προηγούμενου block, αυτού δηλαδή που μόλις έγινε αποδεκτό.

Κάθε συναλλαγή αποτελεί μέρος του transaction chain, μίας αλυσίδας συναλλαγών που ανατρέχει μέχρι την πρώτη συναλλαγή.

Όταν κάποιος κόμβος πραγματοποιεί μία συναλλαγή, ουσιαστικά διαδίδει (broadcasts) ένα μήνυμα με τη συναλλαγή που πραγματοποιήθηκε, υπογράφοντάς με μία μοναδική ψηφιακή υπογραφή[5] την προηγούμενη συναλλαγή και το public key του επόμενου κατόχου του bitcoin.

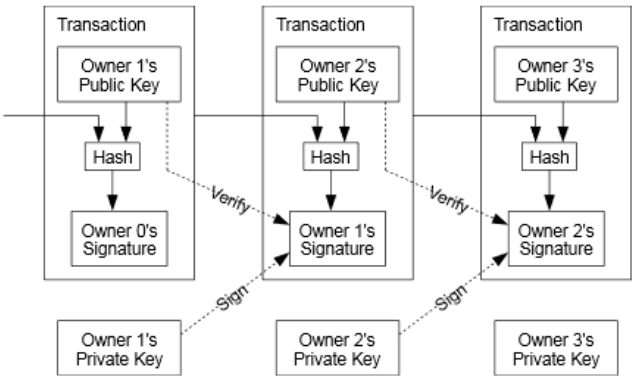

FIGURE 3 TRANSACTION CHAIN

Η ψηφιακή αυτή υπογραφή είναι μοναδική για κάθε συναλλαγή, αφού εξαρτάται από το ίδιο το μήνυμα. Επιπλέον, με την ψηφιακή υπογραφή εγγυάται και η ακεραιότητα και η αυθεντικότητα των δεδομένων του μηνύματος, καθώς οποιαδήποτε αλλαγή σε αυτό, θα ακύρωνε την ψηφιακή υπογραφή.

Εάν χαθεί το ιδιωτικό κλειδί ενός bitcoin wallet, χάνονται και τα bitcoins που κατέχει, όχι μόνο από τον κάτοχο, αλλά από την οικονομία του bitcoin.

Η σειρά των συναλλαγών

Ιδιαίτερα σημαντικό στοιχείο ασφαλείας στο bitcoin αποτελεί η σειρά με την οποία λαμβάνονται τα μηνύματα περί συναλλαγών στο δίκτυο.

[4] Βιβλιογραφικά καλείται proof of work, καθώς η λύση αντανακλά τον υπολογιστικό χρόνο που έχει αφιερωθεί για την εύρεση της λύσης.

[5] Μία ψηφιακή υπογραφή βασίζεται στις αρχές της ασύμμετρης κρυπτογραφίας, με βασικό της στοιχείο να αποτελεί πως μπορεί να επαληθευτεί η αυθεντικότητα ενός μηνύματος.

Έστω ότι ο η κακόβουλη Alice στέλνει ένα ποσό σε bitcoins στον Bob. Αφού ο Bob αποστείλει το προϊόν το οποίο η Alice αγόρασε, η Alice δημιουργεί μία δεύτερη συναλλαγή, στέλνοντας τα ίδια inputs στον εαυτό της. Λόγω της δομής του δικτύου, σε κάποιους κόμβους η δεύτερη συναλλαγή θα ερχόταν πρώτη, με αποτέλεσμα να θεωρείται η συναλλαγή με τον Bob άκυρη. Ουσιαστικά θα υπήρχε διαφωνία μεταξύ των διαφορετικών κόμβων για το εάν τα bitcoins της συναλλαγής ανήκουν στο Bob ή στην Alice.

Η λύση για την κοινή γνώση περί σειράς των συναλλαγών είναι το block chaining όπου κάθε block περιέχει το hash του προηγούμενου.

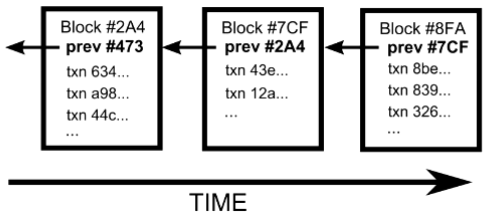

FIGURE 4 BLOCK CHAIN

Κάθε συναλλαγή η οποία θεωρείται έγκυρη, βρίσκεται μέσα στο block chain. Οι συναλλαγές οι οποίες δε βρίσκονται ακόμη εκεί, θεωρούνται μη επαληθευμένες.

Κάθε κόμβος μπορεί να ομαδοποιήσει πολλές μη επαληθευμένες συναλλαγές σε ένα block και να κάνει broadcast στο υπόλοιπο δίκτυο για το επόμενο block στο block chain.

Σε κάθε block πρέπει να περιέχεται η λύση σε ένα μαθηματικό πρόβλημα (proof of work): να βρει το input ενός hash που δημιουργήθηκε από μία cryptographic hash function (SHA256) μέσω τυχαίων guesses τα οποία συνδέονται με έναν αριθμό nonce. Σε όλο το δίκτυο του Bitcoin, παίρνει περίπου 10 λεπτά μέχρι να βρεθεί η λύση στο παραπάνω πρόβλημα. Ο πρώτος που λύσει το πρόβλημα είναι ο επόμενος του οποίου το block θα είναι η επόμενη στο transactions block chain.

Επειδή υπάρχει η περίπτωση ταυτόχρονης λύσης προβλημάτων, δημιουργούνται branches στο block chain, από τα οποία επιλέγεται το μακρύτερο ως το πιο αξιόπιστο.

Καθώς η υπολογιστική δυνατότητα του δικτύου αυξάνεται, γίνεται αναδιαμόρφωση του μαθηματικού προβλήματος ώστε τα 10 λεπτά κατά μέσο όρο για την επίλυσή του από το δίκτυο να παραμένουν σταθερά.

Πιθανή επίθεση

Πιθανή επίθεση double spending αποτελεί η δημιουργία ενός μακρύτερου branch στο οποίο θα χανόταν η σειρά μίας συναλλαγής. Ωστόσο, αυτό είναι ιδιαίτερα δύσκολο καθώς η κακόβουλη Alice που ηγείται της επίθεσης πρέπει να ανταγωνιστεί την υπολογιστική ισχύ όλου του υπόλοιπου δικτύου.

Ανωνυμία χρηστών

Η διευθυνσιοδότηση στο bitcoin network βασίζεται ουσιαστικά στα public keys του κάθε bitcoin wallet.

Αν οι συναλλαγές στο δίκτυο του bitcoin πραγματοποιηθούν από ένα δίκτυο ανωνυμίας (βλ. Tor), τότε το μόνο που συνδέει το λογαριασμό με τον κάτοχό του είναι το public key. Το σύνολο των public keys που μπορούν να χρησιμοποιηθούν στο bitcoin είναι $1.46 * 2^{160}$.

Επιπλέον, για να μην μπορούν να ομαδοποιηθούν συναλλαγές bitcoin μεταξύ τους, δίνεται η δυνατότητα στο χρήστη παραγωγής νέου public key ανά συναλλαγή.

Εξόρυξη bitcoin (Bitcoin mining)

Ως τρόπος διανομής των bitcoins, όταν βρίσκεται η λύση σε πρόβλημα για πρόταση επόμενου block στο block chain, δίνεται κάποιο ποσό ως ανταμοιβή.

Το ποσό αυτό μειώνεται στο μισό, κάθε τέσσερα έτη. Όταν, λοιπόν δεν υπάρχουν υπόλοιπα bitcoins προς εξόρυξη, τότε η ανταμοιβή στο block θα αποτελεί κάποιο transaction fee. Συναλλαγές που δε θα έχουν transaction fee, δε θα επιλέγονται έναντι αυτών που έχουν.

Πλεονεκτήματα bitcoin

- Καμία κυβέρνηση δεν μπορεί να εκτυπώσει ή να υποτιμήσει το νόμισμα.
- Ανωνυμία συναλλασσόμενων.
- Χαμηλότερα κόστη συναλλαγών σε σχέση με τις τράπεζες.

Προβλήματα bitcoin

- Δυσκολία συναλλαγής.
- Χρήση σε παράνομες συναλλαγές με αδυναμία παρακολούθησης.
- Η εξόρυξη απαιτεί σημαντικό ποσό ενέργειας.
- Χρόνος επαλήθευσης συναλλαγής.

Βιβλιογραφία

bitcoin.org. (n.d.). *bitcoin.org*. Ανάκτηση από wiki.bitcoin.it: https://en.bitcoin.it/wiki/Main_Page

Nakamoto, S. (n.d.). *Bitcoin: A peer-to-peer Electronic Cash System.* Ανάκτηση από bitcoin.org: https://bitcoin.org/bitcoin.pdf

Nguyen, M. V. (2009, 12). *Exploring Cryptography Using the Sage Computer Algebra System.* Ανάκτηση από sagemath.org: http://www.sagemath.org/files/thesis/nguyen-thesis-2009.pdf

online-domain-tools.com. (n.d.). *RC4 – Symmetric Ciphers Online*. Ανάκτηση από online-domain-tools.com: http://rc4.online-domain-tools.com/

rosettacode.org. (2014, November). *Vigenère cipher/Cryptanalysis*. Ανάκτηση από rosettacode.org: http://rosettacode.org/wiki/Vigen%C3%A8re_cipher/Cryptanalysis

stackoverflow.com. (2012, September). *Simply put, what does perfect secrecy means?* Ανάκτηση από stackoverflow.com: http://crypto.stackexchange.com/questions/3896/simply-put-what-does-perfect-secrecy-means

stackoverflow.com. (2013, 10). *Determining the key of a Vigenere Cipher if key length is known*. Ανάκτηση από stackoverflow.com: http://stackoverflow.com/questions/19478566/determining-the-key-of-a-vigenere-cipher-if-key-length-is-known

Wikipedia. (2015, 04 10). *Bitcoin*. Ανάκτηση από Wikipedia: http://en.wikipedia.org/wiki/Bitcoin

Wikipedia. (n.d.). *Block Cipher Mode of Operation*. Ανάκτηση από wikipedia.org: http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Electronic_Codebook_.28ECB.29

Wikipedia. (n.d.). *Blowfish (cipher)*. Ανάκτηση από wikipedia.org: http://en.wikipedia.org/wiki/Blowfish_%28cipher%29

Wikipedia. (n.d.). *Ανάλυση Συχνότητας Γλώσσας*. Ανάκτηση από wikipedia.org: http://el.wikipedia.org/wiki/%CE%91%CE%BD%CE%AC%CE%BB%CF%85%CF%83%CE%B7_%CF%83%CF%85%CF%87%CE%BD%CF%8C%CF%84%CE%B7%CF%84%CE%B1%CF%82_%CE%B3%CE%BB%CF%8E%CF%83%CF%83%CE%B1%CF%82

Κωνσταντίνος, Δ. (2015, March). *Πανεπιστημιακές Σημειώσεις*. Ανάκτηση από pileas.csd.auth.gr: http://pileas.csd.auth.gr/file.php/141/cource-1-2-3.pdf

Παράρτημα

Οδηγίες εκτέλεσης

Για την εκτέλεση των παρακάτω python scripts, τρέχετε την παρακάτω εντολή:

```
python3 <path_to_script.py>
```

Ex2_code.py

```
#/usr/bin/env python3

# RC4 Encryption implementation

# Key Scheduling Algorithm
def KSA(key):
        S = []
        for i in range(0, 256):
                S.append(i)
        j = 0
        for i in range(0, 256):
                j = (j + S[i] + ord(key[i%len(key)]))%256
                swap(S, i, j)
        return S

# Pseudo-random Generation Algorithm
def PRGA(plaintext, S):
        i = 0
        j = 0
        k = []
        plaintext_length = len(plaintext)
        while plaintext_length > 0:
                i = (i+1)%256
                j = (j+S[i])%256
                swap(S, i, j)
```

```
                k.append(S[(S[i]+S[j])%256])
                plaintext_length -= 1
        return k

# Encryption function
def encrypt(K, plaintext):
        plaintext_bin = []
        counter = 0
        output = []
        for character in plaintext:
                plaintext_bin.append(ord(plaintext[counter]))
                output.append(plaintext_bin[counter] ^ K[counter])
                counter += 1
        return output

# Swapping letter function
def swap(S, i, j):
        temp = S[i]
        S[i] = S[j]
        S[j] = temp

if __name__ == "__main__":
        key = "MATRIX"
        text = "Never send a human to do a machine s job"
        S = KSA(key)
        k = PRGA(text, S)
        encrypted = encrypt(k, text)
        for number in encrypted:
                print(format(number, '02x'), end=" ")
        print("")
```

Ex3_code.py

```
from operator import itemgetter

# Dictionary on Greek letters-ints
let_to_int = {}
```

```
int_to_let = {}

for i in range (25):
    if 913+i > 930:
        let_to_int[chr(913+i)] = i-1
        int_to_let[i-1] = chr(913+i)
    elif 913+i == 930:
        continue
    else:
        let_to_int[chr(913+i)] = i
        int_to_let[i] = chr(913+i)



# Construct Greek Uppercase letters list
greek_uppercase = []

for key in let_to_int:
        greek_uppercase.append(key)

greek_uppercase = sorted(greek_uppercase)

# Computation of Index of Coincidence for given array of letters
def compute_greek_IC(letter_array, letters_length):
    sum = 0
    for i in range(24):
        sum += letter_array[i]*(letter_array[i]-1)/(letters_length*(letters_length-1))
    return sum

# Greek_IC as given in exercise input
greek_ic = 0.069

# Decryption function
def vigenere_decrypt(target_freqs, input):
    nchars = 24 # number of uppercase characters in Greek alphabet
    ordA = let_to_int['A']
    sorted_targets = sorted(target_freqs)
```

```
# Calculates frequences for each letter inside the text
def frequency(input):
    result = [[c, 0.0] for c in greek_uppercase]
    for c in input:
        result[let_to_int[c] - ordA][1] += 1
    return result

# Calculates correlation of characters in text and statistical distribution
def correlation(input):
    result = 0.0
    freq = frequency(input)
    freq.sort(key=itemgetter(1))

    for i, f in enumerate(freq):
        result += f[1] * sorted_targets[i]
    return result

cleaned = input
best_len = 0

# Find the key length that was used to Vigenere Cipher
found = False
while not found:
    for key_length in range(2, len(cleaned)):
        subtexts = []
        for i in range(key_length):
            column = []
            j = 0
            while i+j < len(cleaned):
                column.append(cleaned[i+j])
                j += key_length
            if len(column) == 1 :
                break;
            subtexts.append(column)
        for column in subtexts:
            # print(column)
            frequencies = [0] * 24
```

```
            for i in range(len(column)):
               frequencies[let_to_int[column[i]] - let_to_int['A']] += 1
            for i in range(24):
                frequencies[i] *= 100/len(column)
            # print(frequencies)
            # print(sum(frequencies))
            column_ic = compute_greek_IC(frequencies,100)
            # print(column_ic)
            if abs(column_ic - greek_ic) < 0.001 :
                found = True;
                best_len = key_length
                print("Found k: " + str(key_length))
                break;
        if found:
            break

# Exit if no length is found to satisfy the constraint
if best_len == 0:
    return ("Text is too short to analyze", "")

# Re-create the frequencies table
pieces = [[] for _ in range(best_len)]
for i, c in enumerate(cleaned):
    pieces[i % best_len].append(c)

freqs = [frequency(p) for p in pieces]

# Find best match
key = ""
for fr in freqs:
    fr.sort(key=itemgetter(1), reverse=True)

    m = 0
    max_corr = 0.0
    for j in range(nchars):
        corr = 0.0
        c = ordA + j
```

```
            for frc in fr:
                d = (let_to_int[frc[0]] - c + nchars) % nchars
                corr += frc[1] * target_freqs[d]

            if corr > max_corr:
                m = j
                max_corr = corr

        key += int_to_let[m + ordA]

    r = (int_to_let[(let_to_int[c] - let_to_int[key[i % best_len]] + nchars) % nchars + ordA] for i, c in enumerate(cleaned))
    return (key, "".join(r))



def main():
    encoded = """ΕΝΠΠΧΙΦΤΔΛΠΞΝΑΒΧΛΟΨΙΟΓΖΩΠΖΓΓΔΚΑΚΑΑΝΜΦΕΔΟΡΑΤΒΔΩΥΟΩΙΝΥΗΑΠΡΜΩΔΥΨΠΡΕΡΟΛΕΝΑΤΤΑΙΤΤΡΡΡΟΛΕΝΑΤΓΑΚΛΣΗΑΛΩΑΛΦΡΗΣΝΨΛΟΥΨΔΑΔΛΚΓΑΑΗΙΚΨΨΙΜΟΔΥΨΖΖΘΑΔΧΩΓΛΘΥΙΛΝΞΛΣΙΖΖΒΚΛΒΣΚΤΙΑΙΝΣΓΛΔΕΙΝΣΒΟΔΧΙΝΣΖΦΨΦΨΙΙΨΖΥΓΞΕΕΚΡΦΝΕΔΗΒΟΛΟΞΩΑΛΕΟΡΓΖΑΩΡΕΘΜΩΤΟΨΙΑΒΚΩΗΚΦΑΤΨΟΛΧΝΩΨΜΗΔΟΔΗΞΟΙΠΕΘΑΤΔΛΟΨΣΝΓΘΧΩΧΝΗΑΦΥΙΕΔΚΓΣΤΖΖΒΠΨΖΘΥΦΨΘΖΧΟΛΕΝΑΤΕΙΙΑΔΩΨΘΥΚΟΦΝΚΧΡΩΗΑΓΗΘΥΓΟΒΗΙΖΚΖΔΥΓΤΝΤΖΨΔΔΤΚΧΥΤΖΜΛΘΣΣΤΙΗΙΖΚΠΣΝΑΤΩΚΠΦΟΟΑΧΙΔΛΡΔΕΘΥΙΤΙΧΤΒΜΗΑΑΡΓΖΑΗΛΑΡΠΧΙΣΤΙΓΕΘΥΙΡΞΜΑΕΠΖΓΝΦΗΣΕΩΨΙΦΨΔΕΛΣΝΕΑΓΑΩΚΑΝΓΣΑΒΦΙΚΤΛΑΓΟΙΘΝΘΤΖΧΨΑΙΙΑΚΤΟΔΕΧΟΔΟΧΦΤΦΗΒΡΓΤΛΒΑΙΟΓΦΜΠΥΝΟΒΗΣΖΑΔΣΑΜΚΡΝΠΨΔΛΣΛΕΚΑΔΖΟΚΑΣΧΨΘΜΜΟΛΧΡΖΙΝΣΓΛΒΧΤΜΑΤΤΣΝΜΧΣΚΡΜΘΜΜΥΝΤΙΧΤΒΣΚΡΒΙΤΥΗΖΓΡΖΤΕΕΚΡΦΗΤΟΥΚΡΓΝΤΔΛΠΥΤΡΓΛΤΧΥΙΩΨΛΤΖΨΝΝΞΜΟΓΛΒΤΨΓΒΩΑΥΤΠΣΝΩΔΟΔΛΘΣΜΟΑΡΓΣΦΨΑΦΦΤΣΛΒΛΥΝΜΧΧΥΤΙΙΑΛΕΩΔΝΡΗΦΕΩΛΟΡΚΒΣΚΡΥΤΖΣΣΥΥΙΑΨΓΥΔΤΣΓΞΛΗΥΖΕΘΓΣΜΠΔΒΧΕΩΡΖΖΒΧΤΒΑΕΟΒΗΙΝΣΙΥΝΙΝΘΜΠΔΓΡΚΘΛΛΥΙΦΤΒΣΨΛΦΓΖΣΣΚΟΝΨΤΟΙΙΕΙΔΨΚΙΙΔΨΗΨΚΟΑΛΡΝΚΘΤΩΔΥΤΠΣΝΥΔΖΕΨΒΑΦΑΥΝΜΜΕΚΡΦΝΗΑΡΟΒΡΝΚΒΟΗΑΤΘΤΙΟΚΡΦΡΚΒΔΥΩΡΓΝΩΤΠΘΑΡΓΣΖΖΧΙΖΝΧΤΧΨΣΝΕΘΣΛΨΟΧΤΒΤΧΨΝΘΦΧΝΥΤΙΚΣΚΙΙΛΔΗΨΛΤΩΔΙΕΨΜΕΙΡΚΦΤΑΑΑΖΑΦΤΚΧΧΔΝΦΜΜΗΦΚΟΧΤΒΟΒΛΠΝΠΣΘΖΖΦΗΙΣΕΘΑΩΨΤΖΟΑΛΡΗΙΚΑΤΤΨΙΣΖΟΓΛΒΡΟΧΡΠΨΩΦΙΓΟΔΕΙΤΓΦΙΠΡΕΘΜΩΑΛΕΘΑΝΡΝΥΗΝΘΛΧΣΥΚΦΠΓΣΛΠΦΝΦΝΩΝΑΛΤΓΜΝΝΧΕΘΑΤΖΘΘΚΩΜΗΦΙΠΦΙΨΨΒΤΧΨΩΝΦΓΟΙΧΡΝΤΦΕΒΗΨΖΞΖΗΚΡΦΡΗΘΤΧΕΙΓΕΡΝΓΛΒΝΔΘΜΠΥΡΝΞΜΗΚΡΒΗΓΥΜΧΡΕΜΗΩΜΖΖΕΗΤΖΕΩΔΓΝΤΝΤΩΛΝΧΤΜΟΓΧΔΕΓΡΝΠΨΠΕΣΙΩΔΛΩΡΕΙΙΙΧΝΑΣΖΓΛΨΙΦΔΡΝΗΛΡΠΓΡΝΗΑΔΡΓΖΑΩΣΔΩΤΜΑΥΨΨΖΜΝΣΚΡΥΝΝΘΣΗΑΔΒΨΚΕΩΧΨΑΙΖΑΡΛΒΝΗΒΖΥΤΣΝΓΖΑΜΖΤΝΔΒΖΥΤΙΥΣΘΥΙΛΦΥΛΡΠΖΖΒΖΨΣΥΚΑΔΖΚΘΥΓΟΙΧΙΝΝΥΦΝΜΖΩΝΠΒΝΨΚΒΖΥΔΓΡΑΜΩΘΤΦΤΖΣΣΤΖΒΗΞΟΙΠΞΝΑΨΒΝΠΤΩΝΛΣΤΖΒΝΕΤΛΜΠΕΟΖΚΔΕΖΨΝΒΓΣΣΑΛΡΗΑΛΣΑΦΟΕΙΜ"""
    greek_ab_freq = [
                12, 0.8, 2, 1.7, 8, 0.5, 2.9, 1.3, 7.8, 4.2, 3.3, 4.4, 7.9, 0.6, 9.8, 5.024, 5.009, 4.9, 9.1, 4.3, 1.2, 1.4, 0.2, 1.6
        ]

    (key, decoded) = vigenere_decrypt(greek_ab_freq, encoded) # maybe divide by 100
    print("Key:", key)
    print("\nText:", decoded)

main()
```

Ex4_code.py

```
#! /usr/bin/env python3
```

```
# Shift cipher code



text = 'ΟΚΗΘΜΦΔΖΘΓΟΘΧΥΚΧΣΦΘΜΦΜΧΓΟΣΨΧΚΠΦΧΘΖΚΠ'

text2 = list(text)

text_int = list(text)


# Create two invert with each other hashes for Greek uppercase letter-int dictionary (due to inconsistence
of sigm inside the ASCII table)

let_to_int = {}

int_to_let = {}


for i in range (25):

    if 913+i > 930:

        let_to_int[chr(913+i)] = i-1

        int_to_let[i-1] = chr(913+i)

    elif 913+i == 930:

        continue

    else:

        let_to_int[chr(913+i)] = i

        int_to_let[i] = chr(913+i)




for i in range(len(text)):

    text_int[i] = let_to_int[text[i]]




if __name__=="__main__":

    for i in range(25):

        for j in range(len(text)):

            text2[j] = int_to_let[(text_int[j]+i)%24]

        print(''.join(text2))
```

Ex5_code.py

```
#/usr/bin/env python3


from Crypto.Cipher import AES

import binascii

import copy
```

```
import string
import random



# Avalanche effect in AES


NUMBER_OF_INPUTS = 80 # Number of inputs from which we gather results


# Encrypt message with AES
def encrypt(key, bin_message):
        obj = AES.new(key, AES.MODE_ECB)
        ciphertext1 = binascii.hexlify(obj.encrypt(bin_message))
        ciphertext = bin(int(ciphertext1, 16))[2:]
        ciphertext = '0000000000000000'[:128-len(ciphertext)] + ciphertext # padding to the front for having
the same length in ciphertexts
        return ciphertext



# Measure the number of different bits between two encrypted messages
def bin_distance(key, message1, message2):
        # message 1 and message 2 MUST have the same length and differ by one bit
        ciphertext1 = encrypt(key, message1)
        ciphertext2 = encrypt(key, message2)
        counter = 0
        counter_step = 0
        for character in ciphertext1:
                if ciphertext1[counter_step] != ciphertext2[counter_step]:
                        counter += 1
                counter_step += 1
        return counter


# Generate a random string
def string_gen(size=16, chars=string.ascii_lowercase + string.digits):
        return ''.join(random.choice(chars) for _ in range(size))


# Convert string to bitstring
def tobits(s):
```

```
    result = []
    for c in s:
        bits = bin(ord(c))[2:]
        bits = '00000000'[len(bits):] + bits
        result.extend([int(b) for b in bits])
    return result

# Convert bistring to string
def frombits(bits):
    chars = []
    for b in range(int(len(bits) / 8)):
        byte = bits[b*8:(b+1)*8]
        chars.append(chr(int(''.join([str(bit) for bit in byte]), 2)))
    return ''.join(chars)

if __name__ == "__main__":
        samples = []
        samples_diff = []
        for i in range(NUMBER_OF_INPUTS):
                samples.append(string_gen())

        samples_diff = []
        for i in range(NUMBER_OF_INPUTS):
                bitstring = tobits(samples[i])
                position_to_change = random.randint(0, len(bitstring)-1)
                bitstring_list = list(bitstring)
                if bitstring_list[position_to_change] == 0:
                        bitstring_list[position_to_change] = 1
                else:
                        bitstring_list[position_to_change] = 0
                samples_diff.append(frombits(bitstring_list))

        results = []
        for i in range(NUMBER_OF_INPUTS):
                try:
                        results.append(bin_distance(b"somethinsomethin", samples[i], samples_diff[i]))
                except: # We opt for not catching the errors due to different lengths of strings in the
Greek language
```

```
                pass

print("The average number of different bits measured by "+str(len(results))+" messages is:")
print(sum(results)/len(results))
```

Ex7_code.py

```
from zipfile import *
import os

# Relevant files
input_zip_file = 'test_zip.zip'
output_file = 'test_zip'
dictionary_file = 'english.txt'

if __name__ == "__main__":
    z_file = ZipFile(input_zip_file)

    # Cleanup
    if os.path.isfile(output_file):
        os.remove(output_file)

    # Read dictionary
    with open(dictionary_file) as f:
        content = f.readlines()

    # bruteforce from dictionary
    secret_key = ''
    for password in content:
        try:
            # print(password)
            z_file.extractall(pwd=str.encode(password.strip()))
        except:
            pass
        if os.path.isfile(output_file) and os.path.getsize(output_file)!=0: # bug of zipfile library outputting files with zero size with wrong password
            secret_key = password;
            break # key found
        else:
            continue

    # Output
```

```
    print('Secret key is: '+secret_key)
    print('Contents of file:')
    with open(output_file, 'r') as file:
        print(file.read())
```

Ex8_code.py

```
#!/usr/bin/env python3

import crypt

# Helper function used in padding a number with zeros
def fillWithZeros(number):
        stringnumber = str(number)
        length = len(stringnumber)
        for i in range(6 - length):
                stringnumber = str(0) + stringnumber
        return stringnumber

enc_password = 'ALDlgQkIyB3/I7Zcqohd2t147EBHagFE2.GHFy.zP5eAHxHbujjnCMLJvrWFqMo6LZ5g5.5eMu61tebZ/djLM.'
salt = 'ANrWqWm8'
enc_password = "$6$"+ salt + "$" + enc_password

# Start guessing from the end (ends sooner!)
for i in reversed(range(999999)):
        password = fillWithZeros(i)
        enc = crypt.crypt(password,"$6$"+salt+"$");
        # print('Now testing: '+ str(i))
        if enc == enc_password:
                print('Found! Password is ' + password)
                exit()
```