

- Konstantinos Filippou
- ics23044

Exercise 1

Ο κρυπταλγόριθμος ροής που περιγράψαμε στη διάλεξη μπορεί εύκολα να γενικευτεί και να λειτουργεί και σε άλλα αλφάβητα εκτός από το δυαδικό. Για χειροκίνητη κρυπτογράφηση, θα ήταν χρήσιμος ένας κρυπταλγόριθμος ροής που λειτουργεί με είσοδο κεφαλαία γράμματα του αγγλικού αλφάβητου.

- Αναπτύξτε ένα σχήμα που να λειτουργεί με τα γράμματα A, B,.., Z, για την κωδικοποίηση των οποίων μπορούμε να χρησιμοποιήσουμε τους αριθμούς 0,1,..,25. Πώς μοιάζει το κλειδί (stream); Ποιες είναι οι συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης;
- Αποκρυπτογραφήστε το ακόλουθο κρυπτογραφημένο κείμενο: BSASPP KKUOSR το οποίο κρυπτογραφήθηκε με τη χρήση του κλειδιού: RSIDPY DKAWOA
- Πώς δολοφονήθηκε ο νεαρός άνδρας;

Solution

Αυτό που μας ζητάει η εκφώνηση στην ουσία είναι, αντί για k1,k2 όπως στον Affine Cipher, να χρησιμοποιούμε μία μεταβλητή key , η οποία να έχει χαρακτήρες του αγγλικού αλφαβήτου (σε αντίθεση με τον Caesar Cipher που αν και μία μεταβλητή πάλι βάζαμε σταθερό αριθμό να κάνει shift). Επομένως, αντέγραψα τον ήδη υπάρχον κώδικα του Affine Cipher μέχρι ένα σημείο. Αφαίρεσα τις μεταβλητές k1,k2 και πρόσθεσα την key με την λέξη 'KEY'. Οι συναρτήσεις str2lst και lst2str παραμαίνουν ίδιες. Στο stream_enc μετατρέπουμε και την καινούργια μεταβλητή από string σε list. Απλώς για να κάνουμε την κρυπτογραφία πρέπει να ξέρουμε πόσοι χαρακτήρες είναι το αρχικό μας μήνυμα. Σε αυτό το σημείο, αν οι χαρακτήρες του key δεν "χωράνε" ακριβώς στην λέξη (Cryptology 10 χαρακτήρες, Key 3 χαρακτήρες) χρησιμοποιούμε και len(k) + 1 για να προσπελαστεί η λέξη μία τελευταία φορά. Παρακάτω έχουμε και το stream_dec η μόνη διαφορά είναι το πρόσημο (plaintextList[i] - extendedKey[i])

```
message = 'CRYPTOLOGY'

key = 'KEY'
def str2lst(s):
    return [ord(x)-65 for x in s]

def lst2str(lst):
    return ''.join([chr(x+65) for x in lst])

def stream_enc(m,k):
    plaintextList = str2lst(m)
    keyList = str2lst(k)
    extendedKey = (keyList * ((len(plaintextList) // len(k)) + 1))[:len(plaintextList)]
    ciphertextList = [(plaintextList[i] + extendedKey[i]) % 26 for i in range(len(plaintextList))]
    ciphertext = lst2str(ciphertextList)
    return ciphertext

print("The plaintext message: " + message + " becomes --> " + stream_enc(message, key))
```

```

ciphertext='BSASPPKKUOSR'

key = 'RSIDPYDKAWOA'
def str2lst(s):
    return [ord(x)-65 for x in s]

def lst2str(lst):
    return ''.join([chr(x+65) for x in lst])

def stream_dec(c,k):
    ciphertextList = str2lst(c)
    keyList = str2lst(k)
    extendedKey = (keyList * ((len(ciphertextList) // len(k)) + 1))[:len(ciphertextList)]
    plaintextList = [(ciphertextList[i] - extendedKey[i]) % 26 for i in
range(len(ciphertextList))]
    plaintext = lst2str(plaintextList)
    return plaintext

print("The ciphertext message: " + ciphertext + " becomes --> " +
stream_dec(ciphertext,key))

```

Αποκρυπτογραφώντας το κείμενο BSASPP KKUOSR με κλειδί RSIDPY DKAWOA, προκύπτει το ονοματεπώνυμο:
KASPAR HAUSER

```

[51]: ciphertext='BSASPPKKUOSR'

key = 'RSIDPYDKAWOA'
def str2lst(s):
    return [ord(x)-65 for x in s]

def lst2str(lst):
    return ''.join([chr(x+65) for x in lst])

def stream_dec(c,k):
    ciphertextList = str2lst(c)
    keyList = str2lst(k)
    extendedKey = (keyList * ((len(ciphertextList) // len(k)) + 1))[:len(ciphertextList)]
    plaintextList = [(ciphertextList[i] - extendedKey[i]) % 26 for i in range(len(ciphertextList))]
    plaintext = lst2str(plaintextList)
    return plaintext

print("The ciphertext message: " + ciphertext + " becomes --> " + stream_dec(ciphertext,key))
The ciphertext message: BSASPPKKUOSR becomes --> KASPARHAUSER

```

Ο Kaspar Hauser μαχαιρώθηκε στις 9 Δεκεμβρίου 1833 στην κοιλιά από έναν άγνωστο άντρα. Ο Hauser επιβίωσε αρχικά και ισχυρίστηκε ότι ο άγνωστος άντρας πριν τον μαχαιρώσει του είπε <<Είναι το τέλος σου>>. Στις 17 Δεκεμβρίου ο Hauser υπέκυψε στα τραύματα και πέθανε.

Exercise 2

Στα τμήματα κώδικα για κρυπτογράφηση και αποκρυπτογράφηση με τον ομοπαραλληλικό κρυπταλγόριθμο (Affine Cipher) υπολογίσαμε το **mod 26** διότι ορίσαμε $k_1, k_2 \in \mathbb{Z}$ και όχι $k_1, k_2 \in \mathbb{Z}_{26}$. **Αν ορίζαμε $k_1, k_2 \in \mathbb{Z}_{26}$ ποιες αλλαγές θα έπρεπε να γίνουν στις συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης ώστε να λειτουργούν ορθά;** Ορίστε τις νέες συναρτήσεις και εκτελέστε τον κώδικα τους θεωρώντας ως κλειδί το **$k = (15, 9)$** . Μπορούμε να ορίσουμε ως κλειδί το **$k = (12, 7)$** ; Παραθέστε την απάντησή σας τεκμηριώνοντας κατάλληλα.

Solution

Exercise 3

Υποθέτουμε μια κρυπτογράφηση τύπου OTP (One-Time-Pad) με ένα μικρού μήκους κλειδί, για παράδειγμα κλειδί **128 bit**. Το κλειδί αυτό εν συνεχεία χρησιμοποιείται περιοδικά για την κρυπτογράφηση μεγάλου όγκου δεδομένων. Περιγράψτε πώς μια επίθεση KPA θα μπορούσε να σπάσει αυτό το σύστημα.

Solution

Μία επίθεση KPA σημαίνει ότι έχουμε γνωστό και το plaintext και το ciphertext. Επειδή το κλειδί είναι μικρού μήκους και επαναλαμβάνεται, αρκεί ένα τμήμα 128 bit γνωστού κειμένου για να βρεθεί ολόκληρο το κλειδί. Η πράξη XOR είναι $C1 \text{ XOR } P1 = (P1 \text{ XOR } K) \text{ XOR } P1$ το οποίο μέσω της ανιτμεταθετικής ιδιότητας προκύπτει

$$K = C1 \text{ XOR } P1$$

Επειδή το σύστημα χρησιμοποιεί το ίδιο κλειδί **K περιοδικά** για ολόκληρο τον μεγάλο όγκο δεδομένων, μόλις το κλειδί K ανακτηθεί, ο επιτιθέμενος μπορεί να αποκρυπτογραφήσει όλο το υπόλοιπο κρυπτογραφημένο κείμενο.

Exercise 4

Θεωρούμε τον συμμετρικό κρυπταλγόριθμο Advanced Encryption Standard (AES) με μήκος κλειδιού **128-bit** και έστω ένα σενάριο επίθεσης εξαντλητικής δοκιμής. Υποθέτουμε ότι ο αντίταλος έχει στη διάθεσή του ειδικού τύπου υλικό Application Specific Integrated Circuit (ASIC) το οποίο διενεργεί έλεγχο εγκυρότητας ενός κλειδιού με ρυθμό **8x10¹⁵ keys/sec**, και **1 εκ. ευρώ**. Ένα κύκλωμα ASIC κοστίζει **100 ευρώ** συμπεριλαμβανομένου του κόστους ενσωμάτωσης και λειτουργίας του με άλλα ίδιου τύπου κυκλώματα. Πόσα τέτοια κυκλώματα μπορούν να αγοραστούν δεδομένου του κεφαλαίου που έχουμε στη διάθεσή μας; Αν τα κυκλώματα αυτά έτρεχαν παράλληλα, πόσο χρόνο κατά μέσο όρο θα απαιτούσε ο έλεγχος εγκυρότητας του κλειδιού; Συγκρίνετε το χρόνο αυτό με την ηλικία του Σύμπατος (περίπου 10^{10} χρόνια).

Solution

Εφόσον έχω στην διάθεσή μου 1 εκ. ευρώ και το κόστος ενός κυκλώματος ASIC είναι 100 ευρώ, μπορώ να αγοράσω $1. \text{εκ} / 100 = 10.000$ κυκλώματα. Άν ένας ASIC μπορεί να εκτελέσει 8×10^{15} keys/sec τότε με $10.000 * 8 \times 10^{15} = 8 \times 10^{19}$ keys/sec. Το μήκος κλειδιού είναι 128 bit, άρα ο χρόνος κατά μέσο όρο δοκιμών είναι $2^{128}/2 =$

$$2^{127}. \text{ Τμεσος} = \frac{2^{127}}{8 \times 10^{19}} = \frac{1.7014 \times 10^{38}}{8 \times 10^{19}} = 2.126 \times 10^{18} \text{ δευτερόλεπτα. Μετατροπή σε χρόνια: Ένα}$$

έτος είναι περίπου 3.155×10^7 δευτερόλεπτα άρα Τμεσος (έτη) =

$$\frac{2.12675 \times 10^{18} \text{ δευτερόλεπτα}}{3.15576 \times 10^7 \text{ δευτερόλεπτα/έτος}} = 67.39 \text{ δισεκατομμύρια έτη.}$$

$$\text{Σε σχέση με την ηλικία του Σύμπαντος είναι } \frac{6.739 \times 10^{10} \text{ έτη}}{10^{10} \text{ έτη}} \text{ περίπου 6.739}$$

Δηλαδή ο μέσος χρόνος εξαντλητικής δοκιμής είναι 6.7 φορές μεγαλύτερος από την ηλικία του σύμπαντος. Αυτό επιβεβαιώνει τη **θεωρητική ασφάλεια** του AES-128 έναντι επιθέσεων εξαντλητικής δοκιμής με την τρέχουσα υπολογιστική ισχύ.

Exercise 2

Στα τμήματα κώδικα για κρυπτογράφηση και αποκρυπτογράφηση με τον ομοπαραλληλικό κρυπταλγόριθμο (Affine Cipher) υπολογίσαμε το **mod 26** διότι ορίσαμε $k_1, k_2 \in \mathbb{Z}$ και όχι $k_1, k_2 \in \mathbb{Z}_{26}$. **Αν ορίζαμε $k_1, k_2 \in \mathbb{Z}_{26}$ ποιες αλλαγές θα έπρεπε να γίνουν στις συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης ώστε να λειτουργούν ορθά**;**** Ορίστε τις νέες συναρτήσεις και εκτελέστε τον κώδικα τους θεωρώντας ως κλειδί το **k = (15,9)**. Μπορούμε να ορίσουμε ως κλειδί το **k = (12,7)**; Παραθέστε την απάντησή σας τεκμηριώνοντας κατάλληλα.

Solution

Αρχικά πρέπει να προσθέσουμε ότι ανήκει στο \mathbb{Z}_{26} και στις δύο συναρτήσεις. Για την `affine_enc` όπου x βάζουμε $\mathbb{Z}_{26}(x)$ και στο `lst2str` προσθέτουμε `int(x)` αφού αλλάζει ο τύπος.

```
# A τρόπος με a,b ακεραίους
message = 'CRYPTOLOGY'
Z26 = Integers(26)
a=Z26(15)
b=Z26(9)

def str2lst(s):
    return [ord(x)-65 for x in s]

def lst2str(lst):
    return ''.join([chr(int(x) + 65) for x in lst])

def affine_enc(m,k1,k2):
    plaintextList = str2lst(m)
    ciphertextList = [k1 * Z26(x) + k2 for x in plaintextList]
    return lst2str(ciphertextList)

print("The plaintext message: " + message + " becomes --> " + affine_enc(message,a,b))
```

The plaintext message: CRYPTOLOGY becomes --> FDJFBTJTBJ

Στην `affine_dec` ομοίως όπου x βάζουμε $\mathbb{Z}_{26}(x)$ και προσθέτουμε την συνάρτηση `k1.inverse_of_unit` το οποίο υπολογίζει το αντίστροφο χωρίς δυσκολία με μετατροπές τύπων.

```

: # Α τρόπος με a, b ακεραίους
ciphertext='AJINFCVCSI'
Z26 = Integers(26)
a=Z26(15)
b=Z26(9)

def affine_dec(c,k1,k2):
    k1_inverse = k1.inverse_of_unit()
    ciphertextList = str2lst(c)
    plaintextList = [k1_inverse * (Z26(x) - k2) for x in ciphertextList]
    return lst2str(plaintextList)

print("The ciphertext message: " + ciphertext + " becomes --> " + affine_dec(ciphertext,a,b))

```

The ciphertext message: AJINFCVCSI becomes --> PATCYDGDLT

Στην affine_enc μπορούμε να βάλουμε ως τιμές το 12 και 7 αφού δεν χρειάζεται να υπολογίσουμε κάποιο αντίστροφο σε αντίθεση με το affine_dec το οποίο όταν το υπολογίζει θα μας βγάλει σφάλμα καθώς δεν υπάρχει αντίστροφο.