ΚΡΥΠΤΑΝΑΛΥΤΙΚΕΣ ΜΕΘΟΔΟΙ

Ονοματεπώνυμο: Ευάγγελος Ν. Γκούμας

ΑΜ: 2021050105

Διαφορική κρυπτανάλυση συμμετρικών κρυπτογράφων με χρήση Python 3

Στρατιωτική Σχολή Ευελπίδων

Ακαδημαϊκό έτος 2022-2023

Αθήνα 2023

Περιεχόμενα

Περίληψη

Η παρούσα εργασία ασχολείται με την έννοια της διαφορικής κρυπτανάλυσης και εφαρμογή της στη γλώσσα προγραμματισμού Python 3. Παρουσιάζεται η λειτουργία της διαφορικής κρυπτανάλυσης, εμφανίζεται ένα παράδειγμα επίθεσης και στη συνέχεια εφαρμόζεται η διαφορική κρυπτανάλυση με χρήση Python 3 και αναλύονται τα αποτελέσματα.

Abstract

The present work aims to investigate the application of differential cryptanalysis in the analysis of symmetric key ciphers. By utilizing the programming language Python 3, the concept of differential cryptanalysis is studied, demonstrated through an example attack, and applied in a practical attack. The results of this attack are then analyzed and discussed in the context of symmetric key cryptography. The objective is to provide a deeper understanding of the differential cryptanalysis technique and its potential for evaluating the security of cryptographic algorithms.

1. Διαφορική Κρυπτανάλυση

Η διαφορική κρυπτανάλυση(differential cryptanalysis)είναι μια μορφή κρυπτανάλυσης που εφαρμόζεται σε αλγόριθμους συμμετρικών κλειδιών. Αυτό επινοήθηκε από τους Ισραηλινούς Eli Biham και τον Adi Shamir. Ουσιαστικά, είναι η εξέταση των διαφορών σε μια είσοδο και πώς αυτό επηρεάζει την προκύπτουσα διαφορά στην έξοδο. Αρχικά λειτουργούσε μόνο με επιλεγμένο απλό κείμενο. Ωστόσο, θα μπορούμε επίσης εργαστούμε μόνο με γνωστό απλό κείμενο και κρυπτογραφημένο κείμενο.

Η επίθεση βασίζεται στην εμφάνιση ζευγών εισόδων απλού κειμένου που σχετίζονται με κάποια σταθερή διαφορά(Biham and Shamir). Ο συνήθης τρόπος ορισμού των διαφορών είναι μέσω της λειτουργίας XOR, αλλά μπορούν να χρησιμοποιηθούν και άλλες μέθοδοι. Ο εισβολέας υπολογίζει τις διαφορές στα κρυπτογραφημένα κείμενα που προκύπτουν και αναζητά κάποιο στατιστικό μοτίβο. Οι διαφορές που προκύπτουν ονομάζονται διαφορικό. Με άλλα λόγια, "η διαφορική κρυπτανάλυση επικεντρώνεται στην εύρεση μιας σχέσης μεταξύ των αλλαγών που συμβαίνουν στα bit εξόδου ως αποτέλεσμα της αλλαγής ορισμένων από τα bit εισόδου".

Η βασική ιδέα στη διαφορική κρυπτανάλυση είναι ότι, αναλύοντας τις αλλαγές σε ορισμένα επιλεγμένα απλά κείμενα και τη διαφορά στις εξόδους που προκύπτουν από την κρυπτογράφηση του καθενός, είναι δυνατό να ανακτηθούν ορισμένες ιδιότητες του κλειδιού. Η διαφορική κρυπτανάλυση μετρά τη διαφορά XOR μεταξύ δύο τιμών.

Η διαφορική κρυπτανάλυση αφορά τις πιθανότητες. Έτσι, το ερώτημα που τίθεται είναι το εξής: Ποια είναι η πιθανότητα ότι μια δεδομένη διαφορά στην είσοδο ΔΧ θα οδηγήσει σε μια συγκεκριμένη διαφορά στην έξοδο ΔΥ; Στις περισσότερες περιπτώσεις, η διαφορική ανάλυση ξεκινά με το s-box. Δεδομένου ότι οι περισσότεροι συμμετρικοί κρυπτογραφικοί αλγόριθμοι χρησιμοποιούν s-boxes, αυτό είναι ένα φυσικό και βολικό μέρος για να ξεκινήσουν οι κρυπταναλυτές. Έστω μια είσοδος X_1 που παράγει μια έξοδος του Y_1 και μια είσοδος του X_2 που παράγει μια έξοδο του Y_2, αυτό παράγει μια διαφορά (δηλαδή, η διαφορά μεταξύ X_1 και X_2 παράγει τη διαφορά μεταξύ Y_1 και Y_2). Αυτό εκφράζεται ως εξής:

- $\Delta X = X_1 \oplus X_2$
- $\Delta Y = Y_1 \oplus Y_2$

Παρόλο που η διαφορική κρυπτανάλυση αποκαλύφθηκε δημόσια τη δεκαετία του 1980, το DES είναι ανθεκτικό στη διαφορική κρυπτανάλυση που βασίζεται στη δομή του s-box DES. Δεδομένου ότι το DES s-box είναι το τμήμα του DES που κατασκευάστηκε από την NSA, είναι λογικό ότι η NSA γνώριζε τη διαφορική κρυπτανάλυση στη δεκαετία του 1970.

2. Παραλλαγές της Διαφορικής Κρυπτανάλυσης

Σε αυτή την ενότητα υπάρχουν παραλλαγές διαφορικής κρυπτανάλυσης.

2.1 Higher-Order Διαφορική Κρυπτανάλυση

Αυτή είναι ουσιαστικά μια βελτίωση στη διαφορική κρυπτανάλυση που αναπτύχθηκε από τον Lars Knudsen το 1994. Η διαφορική κρυπτανάλυση ανώτερης τάξης επικεντρώνεται στις διαφορές μεταξύ των διαφορών που θα μπορούσαν να βρεθούν με τη συνηθισμένη διαφορική κρυπτανάλυση. Αυτή η τεχνική έχει αποδειχθεί ότι είναι πιο ισχυρή από τη συνηθισμένη διαφορική κρυπτανάλυση. Συγκεκριμένα, έχει εφαρμοστεί σε έναν συμμετρικό αλγόριθμο γνωστό ως KN-Cipher. Η διαφορική κρυπτανάλυση υψηλότερης τάξης έχει επίσης εφαρμοστεί σε μια ποικιλία άλλων αλγορίθμων, συμπεριλαμβανομένου του CAST.

2.2 Truncated Διαφορική Κρυπτανάλυση

Η συνηθισμένη διαφορική κρυπτανάλυση εστιάζει στην πλήρη διαφορά μεταξύ δύο κειμένων και του κρυπτογραφημένου κειμένου που προκύπτει, αλλά η Truncated διαφορική κρυπτανάλυση αναλύει μόνο μερικές διαφορές. Λαμβάνοντας υπόψη τις μερικές διαφορές, είναι δυνατό να χρησιμοποιηθούν δύο ή περισσότερες διαφορές στο ίδιο ζεύγος απλού κειμένου/κρυπτογραφημένου κειμένου που θα ληφθούν υπόψη λογαριασμός. Όπως υποδηλώνει το όνομα, αυτή η τεχνική ενδιαφέρεται μόνο να κάνει προβλέψεις για μερικά από τα bit, αντί για ολόκληρο το μπλοκ. Έχει εφαρμοστεί σε Twofsh, Camellia, Skipjack, IDEA και άλλους κωδικούς μπλοκ.

2.3 Impossible Διαφορική Κρυπτανάλυση

Με άλλα λόγια, η τυπική διαφορική κρυπτανάλυση ασχολείται με διαφορές που διαδίδονται μέσω του κρυπτογράφησης με μεγαλύτερη πιθανότητα από την αναμενόμενη. Η Impossible διαφορική κρυπτανάλυση αναζητά διαφορές που έχουν πιθανότητα 0 σε κάποιο σημείο του αλγόριθμου. Αυτό έχει χρησιμοποιηθεί ενάντια στους αλγόριθμους Camellia, ARIA,TEA, Rijndael, Twofsh, Serpent, Skipjack και άλλων.

2.4 Multi-key Διαφορική Κρυπτανάλυση

Αυτός ο τύπος επίθεσης περιλαμβάνει την εύρεση ζεύγων κειμένων που παράγουν κρυπτοκείμενα με μια προβλέψημη διαφορά όταν κρυπτογραφούνται με πολλαπλά κλειδιά.

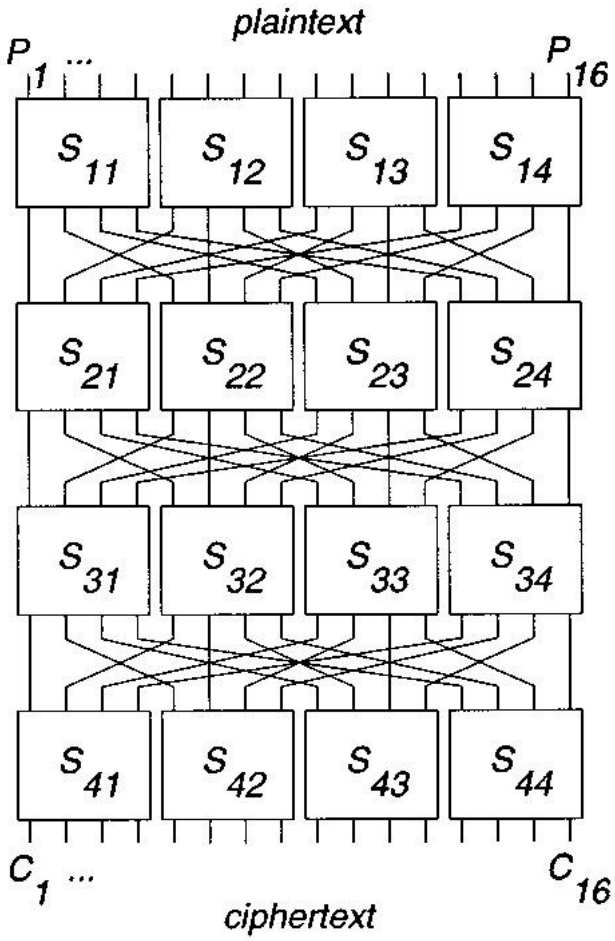
2.5 Approximate Διαφορική Κρυπτανάλυση

Αυτός ο τύπος επίθεσης περιλαμβάνει την εύρεση ζεύγων κειμένων που παράγουν κρυπτοκείμενα με μία πρόβλεψημένη διαφορά που είναι "αρκετά κοντά" στην επιθυμητή διαφορά.

3. Διαφορική κρυπτανάλυση του SPN

Ένα Δίκτυο Αντικατάστασης-Ανακατάταξης (SPN) είναι ένας τύπος κρυπτογραφικής δομής που μπορεί να χρησιμοποιηθεί για τη σχεδίαση κρυπτογράφησης μπλοκ. Σε ένα SPN, η διαδικασία κρυπτογράφησης περιλαμβάνει και τις λειτουργίες αντικατάστασης και ανακατάταξης, οι οποίες εφαρμόζονται στο κείμενο για να παράγουν το κρυπτοκείμενο.

Μία πιθανή επίθεση σε ένα κρυπτογράφημα βάσης SPN είναι γνωστή ως διαφορική επίθεση. Σε μία διαφορική επίθεση, ο επιθέμενος προσπαθεί να βρεί μοτίβα στις διαφορές μεταξύ των μπλοκ κειμένου(plaintext) και των αντίστοιχων μπλοκ κρυπτοκειμένου(ciphertext). Αναλύοντας αυτές τις διαφορές, ο επιθέμενος μπορεί να ανακτήσει πληροφορίες για το κλειδί που χρησιμοποιήθηκε για να κρυπτογραφήσει το κείμενο ή ακόμη και το κείμενο μόνο του.



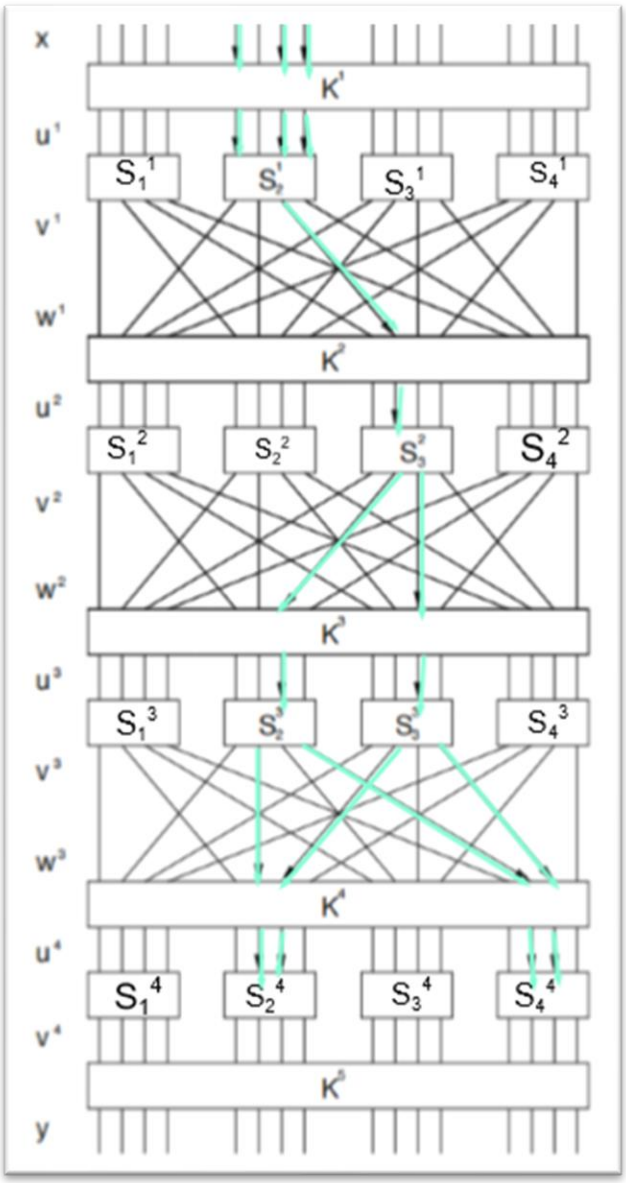
Εικόνα 1. Differential trail for a Substitution-Permutation Network (SPN)

Αφού η διαφορική κρυπτανάλυση είναι μια επιλεγμένη επίθεση απλού κειμένου σημαίνει ότι για το ακόλουθο παράδειγμα επίθεσης επιλέγονται οι είσοδοι και εξετάζονται οι έξοδοι για τη προσπάθεια **εξαγωγής κλειδιού**. Επιπλέον, επειδή το κλειδί του γύρου συνδυάζεται με την είσοδο με αποκλειστική διάζευξη τότε η πιθανότητα των διαφορικών χαρακτηριστικών είναι ανεξάρτητη από τα bits του κλειδιού και τα bits των κλειδιών μπορούν να αγνοηθούν.

Για τη διαφορική επίθεση στο Substitution-Permutation Network της **Εικόνα 2** θεωρούμε το διαφορικό χαρακτηριστικό του S-box:

- S_2^1, $P_r(\Delta X = 1011, \Delta Y = 0010) = \frac{1}{2}$
- S_3^2, $P_r(\Delta X = 0100, \Delta Y = 0110) = \frac{3}{8}$
- S_2^3, $P_r(\Delta X = 0010, \Delta Y = 0101) = \frac{3}{8}$
- S_3^3, $P_r(\Delta X = 0010, \Delta Y = 0101) = \frac{3}{8}$

Στην **Εικόνα 2** με το γαλαζιοπράσινο βέλος εννοούμε τα ‘’1’’ bits που εισέρχονται και εξέρχονται στα εκάστοτε S-boxes του κάθε γύρου.



Εικόνα 2. Differential trail for a Substitution-Permutation Network (SPN)

Η διαφορά εισόδου στο κρυπτογράφημα είναι ισοδύναμη με τη διαφορά εισόδου στο 1ο γύρο:

$\Rightarrow$ $\Delta P = \Delta U_1$ = [**0000 1011 0000 0000**]

Η διαφορά εξόδου του 1ο γύρου είναι:

$\Rightarrow$ ΔV_1 = [**0000 0010 0000 0000**]

- Θεωρώντας το ζεύγος διαφοράς για το S_2^1 πίνακα αντικατάστασης και ακολουθώντας τη μετάθεση του 1ο γύρου(όπως βλέπουμε στην **Εικόνα 2**) παίρνουμε τη ακόλουθη διαφορά.

Η διαφορά εισόδου στο 2ο γύρο:

$\Rightarrow$ ΔU_2 = [**0000 0000 0100 0000**], με πιθανότητα **½** δοθέντος της διαφοράς εισόδου ΔP του κρυπτογραφήματος.

Η διαφορά εξόδου στο 2ο γύρο:

$\Rightarrow$ ΔV_2 = [**0000 0000 0110 0000**]

- Θεωρώντας το ζεύγος διαφοράς για το S_3^2 πίνακα αντικατάστασης και ακολουθώντας τη μετάθεση του 2ο γύρου(όπως βλέπουμε στην **Εικόνα 2**) παίρνουμε τη ακόλουθη διαφορά.

Η διαφορά εισόδου στο 3ο γύρο:

$\Rightarrow$ ΔU_3 = [**0000 0010 0010 0000**], με πιθανότητα **(1/2)x(3/8)** δοθέντος της διαφοράς εισόδου ΔP του κρυπτογραφήματος και με πιθανότητα **3/8** δοθέντος της διαφοράς εισόδου ΔU_2.

Η διαφορά εξόδου στο 3ο γύρο:

$\Rightarrow$ ΔV_3 = [**0000 0101 0101 0000**]

- Θεωρώντας το ζεύγος διαφοράς για τα S_2^3 *και* S_3^3 πίνακες αντικατάστασης και ακολουθώντας τη μετάθεση του 3ο γύρου(όπως βλέπουμε στην **Εικόνα 2**) παίρνουμε τη ακόλουθη διαφορά.

Η διαφορά εισόδου στο 4ο γύρο:

$\Rightarrow$ ΔU_4 = [**0000 0110 0000 0110**], με πιθανότητα **$(1/2)x(3/8)^3$** δοθείσας της διαφοράς εισόδου ΔP του κρυπτογραφήματος και με πιθανότητα **$(3/8)^2$** δοθέντος της διαφοράς εισόδου ΔU_3.

Κατά τη διαδικασία κρυπτανάλυσης, θα κρυπτογραφηθούν πολλά ζεύγη κειμένων που έχουν ΔP = [**0000 1011 0000 0000**]. Με υψηλή πιθανότητα $\frac{27}{1024}$, θα παρουσιαστεί η διαφορική χαρακτηριστική που μας ενδιαφέρει. Χαρακτηρίζουμε τέτοια ζεύγη για ΔP ως σωστά ζεύγη τα ζεύγη διαφορών κειμένων που δεν παρουσιάζουν τη χαρακτηριστική αυτή χαρακτηρίζονται ως λανθασμένα ζεύγη.

Μόλις ανακαλυφθεί ένα διαφορικό χαρακτηριστικό του γύρου **R-1** για ένα κρυπτογράφημα **R** γύρων με επαρκώς μεγάλη πιθανότητα, είναι πιθανό να επιτεθεί ο κρυπταναλυτής ανακτώντας τα bits του τελευταίου δευτερεύωντος κλειδιού. Αυτό έχει σαν αποτέλεσμα την εξαγωγή bits από το δευτερεύον κλειδί K_5.

Η διαδικασία που ακολουθείται περιλαμβάνει τη μερική αποκρυπτογράφηση του τελευταίου γύρου κρυπτογράφησης S_4^2, S_4^4 . Έτσι, για κάθε ζεύγος του κρυπτοκειμένου, θα δοκιμάζαμε και τις 256 τιμές για τα [$K_{5,5}$,…,$K_{5,8}$,$K_{5,13}$,…,$K_{5,16}$].

Για κάθε τιμή του μερικού δευτερευόντος κλειδιού, αυξάνουμε την καταμέτρηση κάθε φορά που η διαφορά εισόδου στον τελικό γύρο που καθορίζεται από τη μερική αποκρυπτογράφηση είναι ίδια με τη διαφορά εισόδου ΔU_4 , όπου προσδιορίζουμε την τιμή

[$\Delta U_{4,5}$,…,$\Delta U_{4,8}$, $\Delta U_{4,13}$,…, $\Delta U_{4,16}$]

εκτελώντας τα δεδομένα προς τα πίσω μέσω του μερικού δευτερεύοντος κλειδιού και των S-boxes S_4^2, S_4^4. Για κάθε τιμή μερικού δευτερεύοντος κλειδιού, η μέτρηση αντιπροσωπεύει τον αριθμό εμφανίσεων διαφορών που είναι συνεπείς με τα σωστά ζεύγη. Η **μέτρηση** που είναι η μεγαλύτερη θεωρείται ότι είναι η σωστή τιμή αφού υποθέτουμε ότι παρατηρήθηκε η υψηλή πιθανότητα εμφάνισης του σωστού ζεύγους.

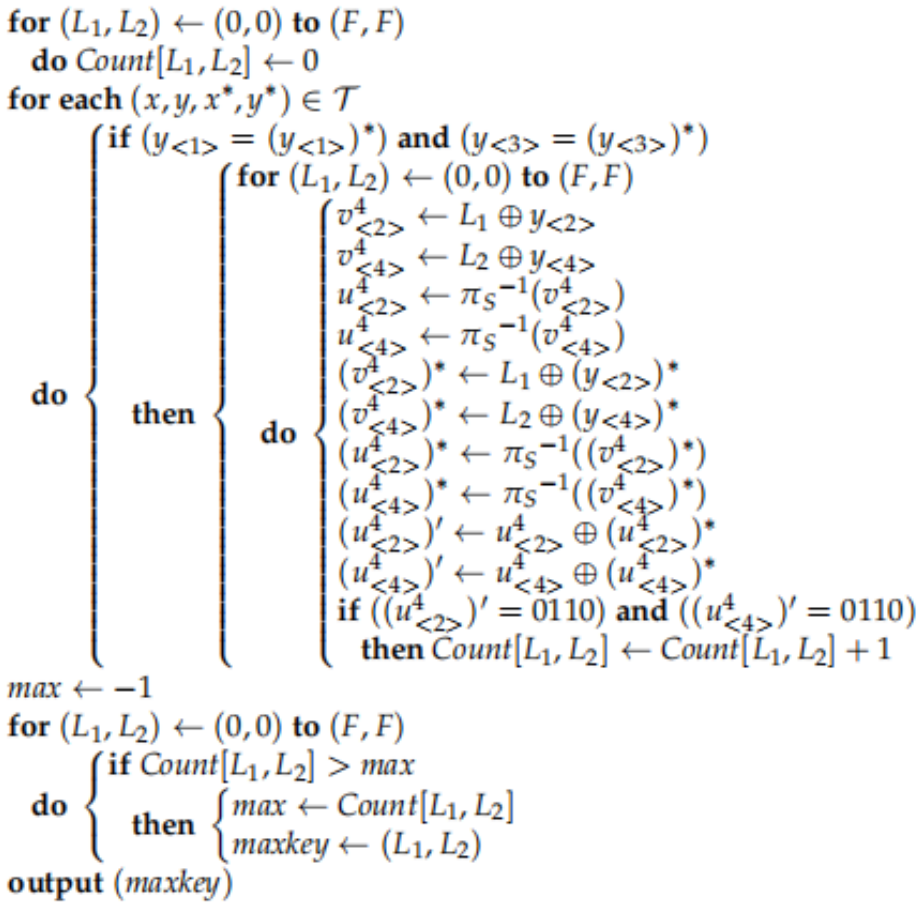
Και η πιθανότητα εμφάνισης του σωστού ζεύγους κλειδιού θα είναι $P_D = \frac{27}{1024} = 0{,}0264$

Στην ακόλουθη ενότητα το ανωτέρω παράδειγμα θα γίνει πιο κατανοητό με συγκεκριμένα κλειδιά που θα δώσουμε και θα βρούμε την πιθανότητα εμφάνισης τους μέσω της Python 3.

4. Διαφορική κρυπτανάλυση του SPN με τη χρήση της Python 3

Σε αυτή τη παράγραφο της εργασίας θα υλοποιήσουμε τη διαφορική κρυπτανάλυση στο SPN της **Εικόνα 2** με τη χρήση της γλώσσας προγραμματισμού Python 3.

Τον ακόλουθο αλγόριθμο επίθεσης θα τον εντάξουμε στο κώδικα που παρατίθεται στις επόμενες σελίδες.



Εικόνα 3. Αλγόριθμος Διαφορικής Επίθεσης

Ο αλγόριθμος της **Εικόνα 3** χρησιμοποιεί μία συγκεκριμένη λειτουργία φιλτραρίσματος. Τα (x, x*,y, y*) tuples για τα οποία ισχύει το διαφορικό, συχνά ονομάζονται δεξιά ζεύγη και είναι τα δεξιά ζεύγη που μας επιτρέπουν να προσδιορίσουμε τα σχετικά κλειδιά bit. Ένα δεξί ζεύγος έχει:

$$(u^4_{<1>})' = (u^4_{<3>})' = 0000$$

Έτσι, αποτελείται ότι ένα δεξί ζεύγος πρέπει να έχει $y_{<1>} = (y_{<1>})^*$και $y_{<3>}= (y_{<3>})^*$. Εάν το (x, x*,y, y*) δεν πληρεί αυτές τις προϋποθέσεις, τότε ξέρουμε ότι δεν είναι ένα δεξί ζεύγος και μπορούμε να το απορρίψουμε. Αυτή η διαδικασία φιλτραρίσματος αυξάνει την αποδοτικότητα του επίθεσης.

Ο τρόπος λειτουργίας του Αλγορίθμου της **Εικόνα 3** μπορεί να περιγραφεί ως εξής. Για κάθε σύνολο (x, x*, y, y*) ∈ T, πρώτα εκτελούμε τη λειτουργία φιλτραρίσματος. Αν (x, x*, y, y*) είναι

ένα δεξί ζεύγος, τότε ελέγχουμε κάθε πιθανό υποκλειδί υποβολής (L_1, L_2) και αυξάνουμε έναν κατάλληλο μετρητή εάν εντοπίσουμε ένα συγκεκριμένο xor. Τα βήματα περιλαμβάνουν τον υπολογισμό ενός exclusive-or με τα υποκλειδιά υποβολής και την εφαρμογή της αντίθετης S-box, ακολουθούμενος από τον υπολογισμό της σχετικής τιμής xor.

```
"""
Differential Cryptanalysis of SPN
"""
######## BEGIN ########

def SBox(n): # The implemention of the SBox function: n in an integer between
0 and 15
    val = hex(n)

    if val == '0x0':
        return int(0xe)
    elif val == '0x1':
        return int(0x4)
    elif val == '0x2':
        return int(0xd)
    elif val == '0x3':
        return int(0x1)
    elif val == '0x4':
        return int(0x2)
    elif val == '0x5':
        return int(0xf)
    elif val == '0x6':
        return int(0xb)
    elif val == '0x7':
        return int(0x8)
    elif val == '0x8':
        return int(0x3)
    elif val == '0x9':
        return int(0xa)
    elif val == '0xa':
        return int(0x6)
    elif val == '0xb':
        return int(0xc)
    elif val == '0xc':
        return int(0x5)
    elif val == '0xd':
        return int(0x9)
    elif val == '0xe':
        return int(0x0)
    elif val == '0xf':
        return int(0x7)
```

```
def PBox(n): #Implementation of the PBox function. Again, n in an integer
between 0 and 15


    val = n + 1
    retval = 0
    if val == 1:
        retval = 1
    elif val == 2:
        retval = 5
    elif val == 3:
        retval = val+6
    elif val == 4:
        retval = (val+9)
    elif val == 5:
        retval = val-3
    elif val == 6:
        retval = val
    elif val == 7:
        retval = val+3
    elif val == 8:
        retval = (val + 6)
    elif val == 9:
        retval = (val-6)
    elif val == 10:
        retval = (val-3)
    elif val == 11:
        retval = val
    elif val == 12:
        retval = (val+3)
    elif val == 13:
        retval = (val-9)
    elif val == 14:
        retval = (val-6)
    elif val == 15:
        retval = (val-3)
    elif val == 16:
        retval = val

    return retval-1

def convertBin(n): #takes an integer between 0 and 15 as input and converts it
to an array of zeroes and ones, representing the binary notation of that
number
    p = format(n, '#06b')
    p = p[2:]
    p = [int (x) for x in p]
    return p
```

```
def BackToInt(l):#takes an array of size 4 of zeroes and ones, and converts it
to integer
    r = 0
    for item in l:
        r = (r<<1)|item
    return r

def DeltaX(xp):#takes an integer x' in [0, 15] as input, and calculates all
possible x*'s corresponding to x's in [0, 15]
    delX = []
    for i in range(16):
        delX.append(i ^ xp)
    return delX
"""
The SPN class

"""
class SPN:

    K = [[], [], [], [], []]#5 round keys for an SPN of 4 rounds
    Key = [3, 10, 9, 4, 13, 6, 3, 15]#the 32-bit key of the SPN
    U = [[], [], [], []]
    V = [[], [], [], []]
    W = [[], [], []]
    Y = []#SPN output
    X =[]#SPN input

#The following lists have been initialised just for the ease of calculation,
and are not relevent since they do not correspond to any significant SPN
property
    u = []
    v = []
    w = []
    v_w = []

    def calcU(self, r, w_r_):#calculates u_r for the round r, given the value
of w_{r-1}
        u = []
        for j in range(len(w_r_)):
            u.append(w_r_[j] ^ self.K[r][j])
        self.U[r] = u

    def calcV(self, r, u_r):#calculates v_r for the round r, given the value
of u_r
        v = []
        for j in range(len(u_r)):
            v.append(SBox(u_r[j]))
        self.V[r] = v
```

```
    def calcW(self, r, v_r):#calculates w_r for the round r, given the value
of v_r
        w = []
        v_w = []
        for v in v_r:
        #We have represented all the 16-bit SPN parameters as an array of 4
integers in [0, 15].
        #But to calculate w, we need all the 16 bits. Hence, here the required
conversion is done

            v_w.append(convertBin(v))

        w_row = []
        for j in range(4):
            k = j*4
            for m in range (k, k+4):
                w_row.append(v_w[int(PBox(m)/4)][PBox(m)%4])
            w.append(w_row)
            w_row = []
        for j in range(4):
            w[j] = BackToInt(w[j])

        self.W[r] = w

    def calcY(self):#calculates the output Y of the SPN
        y = []
        l = len(self.V)
        k = len(self.K)
        for j in range(len(self.V[l-1])):
            y.append(self.V[l-1][j] ^ self.K[4][j])
        self.Y = y


    def __init__(self, X):
        """
        #Initialises the SPN using the given input X.
        We expect a list* conataining a single list of size 4

        *: (yes, a list, with a single element in it.... I had something else
in my mind when creating this architecture,
            but ended up doing something completely different)
        """
        for i in range(5):#Generates 5 round keys for the SPN of 4 rounds
            rkey = []
            for j in range(i, i+4):
                rkey.append(self.Key[j])
            self.K[i] = rkey
```

```
        for x in X:#This loop runs only for one iteration since X contains
just one element
            self._X = x
            if len(x)!= len(self.K[1]):
                print("invalid input")
                pass
            else:
                """
                    Since we are using arrays here, we are forced to used
zero-indexing.
                    So, here U0 will denote the SBox inputs for the first
round.
                    This also means that W0 here is the actual W1.
                    Instead of creating an actual W0, we've directly
calculated the actual U1's using the input X
                """
                for i in range(0, 4):
                    if i == 0:
                        self.calcU(i, x)
                    else:
                        self.calcU(i, self.W[i-1])

                    self.calcV(i, self.U[i])

                    if i != 3:#Calculate W's except for the last round
                        self.calcW(i, self.V[i])
                    else:
                        self.calcY()


    def display(self):#function that displays all the SPN parameters, over all
the 4 rounds
        K0_disp = []
        K1_disp = []
        K2_disp = []
        K3_disp = []
        K4_disp = []
        W_disp = []
        V_disp = []
        U_disp = []
        Y_disp = []
        X_disp = []

        for i in range(4):
            K0_disp.append(convertBin(self.K[0][i]))
            K1_disp.append(convertBin(self.K[1][i]))
            K2_disp.append(convertBin(self.K[2][i]))
            K3_disp.append(convertBin(self.K[3][i]))
```

```
            K4_disp.append(convertBin(self.K[4][i]))

        for w in self.W:
            w_d = []
            for i in range(len(w)):
                w_d.append(convertBin(w[i]))
            W_disp.append(w_d)

        for u in self.U:
            u_d = []
            for i in range(len(u)):
                u_d.append(convertBin(u[i]))
            U_disp.append(u_d)

        for v in self.V:
            v_d = []
            for i in range(len(v)):
                v_d.append(convertBin(v[i]))
            V_disp.append(v_d)

        y_d = []
        for y in self.Y:
            y_d.append(convertBin(y))
        Y_disp.append(y_d)

        x_d = []
        for x in self._X:
            x_d.append(convertBin(x))
        X_disp.append(x_d)

        for w in W_disp:
            print("W is", w)

        for w in U_disp:
            print("U is", w)

        for w in V_disp:
            print("V is", w)

        for w in Y_disp:
            print("Y is", w)

        print("K[0] is ", K0_disp)
        print("K[1] is ", K1_disp)
        print("K[2] is ", K2_disp)
        print("K[3] is ", K3_disp)
        print("K[4] is ", K4_disp)
```

```
### The Differential Attack ####

def DiffAttack(T, Pi_inv):
    """
        In this function, the tuple T consists of two objects of the SPN
class, one corresponding to the input x and other corresponding to the input
x*
        We assume no access to any parameter of the SPN, except the input x
and the output y.
    """
    count = []

    for l1 in range(0, 16):
        """
            We want to find the 8-bit candidate key for a given differential
trail. This 8-bit candidate forms half of the key in round 5
        """
        countRow = []
        for l2 in range(0, 16):
            countRow.append(0)
        count.append(countRow)

    for [sp, sp_] in T:
        x = sp._X
        x_ = sp_._X
        y = sp.Y
        y_ = sp_.Y
        u42 = 0
        u42_ = 0
        u44 = 0
        u44_ = 0
        if y[0] == y_[0] and y[2] == y_[2]:
            for l1 in range(0, 16):
                for l2 in range(0, 16):
                    v42 = l1 ^ y[1]
                    v44 = l2 ^ y[3]
                    u42 = Pi_inv[v42]
                    u44 = Pi_inv[v44]
                    v42_ = l1 ^ y_[1]
                    v44_ = l2 ^ y_[3]
                    u42_ = Pi_inv[v42_]
                    u44_ = Pi_inv[v44_]
                    u42Prime = u42 ^ u42_
                    u44Prime = u44 ^ u44_
                    if u42Prime == 0b0110 and u44Prime == 0b0110:
                        count[l1][l2] = count[l1][l2] + 1

    _max = -1
```

```
    print("\nThe candidate keys with their probabilities are (count,
probability, l1, l2)")
    for l1 in range(0, 16):
        for l2 in range(0, 16):
            if count[l1][l2] > _max:
                _max = count[l1][l2]
                print(_max, float(_max/256.0), l1, l2)
                maxkey = [l1, l2]

    maxkey_disp = [bin(maxkey[0]), bin(maxkey[1])]
    print("\nHence the most probable key candidates for S42 and S44 are:")
    print (maxkey)

    print("The original key for the final round(K5) is:")
    print(sp.K[len(sp.K) - 1])


###################### FINAL Algorithm ###################################
X_prime = []

for i in range(0, 16):
    X_prime.append(i)

DeltaX_ = []#A 2d array, such that DeltaX_[x'][x] = x*
X =  []
for i in range(0, 16):
    DeltaX_.append(DeltaX(i))

for i in range(0, 16):
    X.append(i)

XStar = []

for x in X:
    x_starRow = []
    for xp in X_prime:
        x_starRow.append(DeltaX_[x][xp])
    XStar.append(x_starRow)

Y = []
YStar = []

for x_ in X:
    Y.append(SBox(x_))

for x_ in XStar:
    yrow = []
    for j in range(16):
```

```
        yrow.append(SBox(x_[j]))
    YStar.append(yrow)

Y_Prime = []

for y in YStar:
    yrow = []
    for j in range(16):
        yrow.append(Y[j] ^ y[j])
    Y_Prime.append(yrow)

b_prime = []

for y in Y_Prime:
    brow = []
    for i in range(16):
        c = y.count(i)
        brow.append(c)
    b_prime.append(brow)


Nd = []
for b in b_prime:
    ndrow = []
    for j in range(16):
        ndrow.append(float(b[j]))
    Nd.append(ndrow)

def Rp(i, j):#Function that calculates the propogation ratio
    return(Nd[i][j]/16)

Pi_inv = {0:0}#Dictionary that stores the inverse SBox values

for i in range(0, 16):
    Pi_inv[SBox(i)] = i


INP = []####################### x
for i in range(16):
    for j in range(16):
        for k in range(16):
            INP.append([0, i, j, k])

sp = []######################### array of SPN's for all possible values x

for i in range(len(INP)):
    sp.append(SPN([INP[i]]))

INP1 = []####################### x*
```

```
for i in range(16):
    for j in range(16):
        for k in range(16):
            INP1.append([0, DeltaX_[11][i], DeltaX_[0][j], DeltaX_[0][k]])

sp1 = []##########################array of SPN's for all possible values of x*
for i in range(len(INP1)):
    sp1.append(SPN([INP1[i]]))

OP = []######################### y
for i in range(len(INP)):
    OP.append(sp[i].Y)


OP1 = []######################### y*
for i in range(len(INP1)):
    OP.append(sp1[i].Y)

T =[] # Tuples to be used in the Differential Attack
for i in range(len(sp)):
    T.append([sp[i], sp1[i]])

print("The dataset size is: ",  len(INP)) #Size of the input dataset

def calcMaxProbV(x):#Returns the value that is most likely to occur looking at
the propogation ratios for a given value of x
    prob = []
    retval = 0
    for i in range(16):
        prob.append(Rp(x, i))

    retval = prob.index(max(prob))
    return [retval, max(prob)]

testsp = SPN([[0, 11, 0, 0]])#Constant x' is 0000 1011 0000 0000

print("\nThe Differential Trail: ")
p = 0
for i in range(3):
    if i == 0:
        testsp.U[i] = testsp._X
        arr = testsp.U[i]
        for j in range(len(arr)):
            testsp.V[i][j] = calcMaxProbV(arr[j])[0]
            if arr[j] != 0:
                p = calcMaxProbV(arr[j])[1]
            # print(p)

        testsp.calcW(i, testsp.V[i])
```

```
    else:
        testsp.U[i] = testsp.W[i-1]
        arr = testsp.U[i]
        for j in range(len(arr)):
            testsp.V[i][j] = calcMaxProbV(arr[j])[0]
            if arr[j] != 0:
                p *= calcMaxProbV(arr[j])[1]

        if i != 3:
            testsp.calcW(i, testsp.V[i])

    print("Round", i+1, "U", testsp.U[i])
    print("Round", i+1, "V", testsp.V[i])
    if i != 3:
        print("Round", i+1, "W", testsp.W[i])
    print("probability", p)
DiffAttack(T, Pi_inv)
```

```
The Differential Trail:
Round 1 U [0, 11, 0, 0]
Round 1 V [0, 2, 0, 0]
Round 1 W [0, 0, 4, 0]
probability 0.5
Round 2 U [0, 0, 4, 0]
Round 2 V [0, 0, 6, 0]
Round 2 W [0, 2, 2, 0]
probability 0.1875
Round 3 U [0, 2, 2, 0]
Round 3 V [0, 5, 5, 0]
Round 3 W [0, 6, 0, 6]
probability 0.0263671875

The candidate keys with their probabilities are (count, probability, l1, l2)
0 0.0 0 0
2 0.0078125 0 1
8 0.03125 0 2
10 0.0390625 0 15
12 0.046875 1 12
16 0.0625 1 15
26 0.1015625 3 15
32 0.125 6 10
76 0.296875 6 15

Hence the most probable key candidates for S42 and S44 are:
[6, 15]
The original key for the final round(K5) is:
[13, 6, 3, 15]
```

Εικόνα 4. Output του κώδικα

Η μεγαλύτερη πιθανότητα **(0.296875)** εμφανίζεται για μερική τιμή δευτερεύοντος κλειδιού

$$[K_{5,5},\ldots,K_{5,8},K_{5,13},\ldots,K_{5,16}] = [6, 15]_{hex}$$

Η **πολυπλοκότητα** μίας επίθεσης διαφορικής κρυπτανάλυσης εξαρτάται από πολλούς παράγοντες, όπως:

1. **Το μέγεθος του κλειδιού:** Όσο μεγαλύτερο είναι το μέγεθος του κλειδιού, τόσο πιο δύσκολη είναι η εκτέλεση μίας επίθεσης διαφορικής κρυπτανάλυσης καθώς υπάρχουν περισσότερα δυνατά κλειδιά για δοκιμή.
2. **Ο αριθμός των γύρων**: Όσο περισσότερα γύρους έχει ένα κρυπτογράφημα, τόσο πιο δύσκολη είναι η εκτέλεση μίας επίθεσης διαφορικής κρυπτανάλυσης καθώς υπάρχουν περισσότερα βήματα στη διαδικασία κρυπτογράφησης για ανάλυση.
3. **Διαφορική πιθανότητα:** Όσο μεγαλύτερη είναι η διαφορική πιθανότητα των ενεργών S-boxes, τόσο μεγαλύτερη είναι η χαρακτηριστική πιθανότητα για το πλήρες κρυπτογράφημα.
4. **Υπολογιστικά πόροι που είναι διαθέσιμοι:** Η πολυπλοκότητα μίας επίθεσης διαφορικής κρυπτανάλυσης εξαρτάται επίσης από τους υπολογιστικούς πόρους που είναι διαθέσιμους για τον χάκερ. Μία επίθεση διαφορικής κρυπτανάλυσης μπορεί να είναι πιο πρακτική εάν ο επιτιθέμενος έχει πρόσβαση σε ένα ισχυρό υπολογιστή ή σε ένα cluster υπολογιστών.
5. **Το μέγεθος του ciphertext που είναι διαθέσιμο**: Όσο περισσότερο ciphertext είναι διαθέσιμο για ανάλυση, τόσο ευκολότερη είναι η εκτέλεση μίας επίθεσης διαφορικής κρυπτανάλυσης.
6. **Ενεργά S-boxes :** Όσα **λιγότερα** ενεργά S-boxes έχουμε, τόσο μεγαλύτερη είναι η χαρακτηριστική πιθανότητα.
7. Ο **αριθμός των επιλεγμένων ζευγών** απλού κειμένου N_D που απαιτούνται για τη διάκριση των σωστών ζευγών όταν δοκιμάζονται υποψήφια δευτερεύοντα κλειδία:
$$N_D \approx \frac{c}{p_D}.$$

Η **χρονική πολυπλοκότητα** της επίθεσης διαφορικής κρυπτογράφησης εξαρτάται από το συγκεκριμένο αλγόριθμο κρυπτογράφησης και την πολυπλοκότητα της διαφοράς που επιλέγεται για την επίθεση. Γενικά, η χρονική πολυπλοκότητα μιας επίθεσης διαφορικής κρυπτογράφησης εκφράζεται σε όρους της άγραφης σημαίας O, η οποία είναι μια τρόπος εκφώνησης του υψηλότερου όρου του χρόνου που απαιτείται για την εκτέλεση ενός αλγορίθμου.

Για παράδειγμα, η χρονική πολυπλοκότητα μιας επίθεσης διαφορικής κρυπτογράφησης στο Data Encryption Standard (DES) είναι $O(2^n)$, όπου n είναι ο αριθμός των γύρων στον αλγόριθμο κρυπτογράφησης. Αυτό σημαίνει ότι ο χρόνος που απαιτείται για την εκτέλεση της επίθεσης αυξάνεται εκθετικά με τον αριθμό των γύρων στον αλγόριθμο.

Υλοποίηση & Εργαλεία

Για την ανάγκη της υλοποίησης του κώδικα χρειαστήκαμε ένα Laptop Dell Inspiron 3505 με Ram: 12 GB, Windows 11 και γράφτηκε στο PyCharm Community edition.

Base 16	Base 2
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Πίνακας 1 Μετατροπής δεκαεξαδικού σε δυαδικό

Βιβλιογραφία

[1] Antoine Joux, Algorithmic cryptanalysis, CRC Press, 2009.

[2] Brij B. Gupta, Computer and Cyber Security Principles, Algorithm, Applications and Perspectives, 2019.

[3] Christopher Swenson, Modern cryptanalysis, Wiley, 2008.

[4] Chuck Easttom, Georgetown University and Vanderbilt University Plano, USA Modern Cryptography Applied Mathematics for Encryption and Information Security, Springer.

[5] Douglas R. Stinson and Maura B. Paterson, Cryptography Theory and Practice, Fourth Edition, CRC Press.

[6] Eli Biham, Adi Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer, 1993.

[7] Howard M. Heys, A Tutorial on Linear and Differential Cryptanalysis.

[8] https://cgi.di.uoa.gr/~klimn/cryptography/chapter_3-Block_Ciphers.pdf

[9] https://en.wikipedia.org/wiki/Differential_cryptanalysis

[10] https://github.com/thekoc11/Differential-Cryptanalysis

[11] Mark Stamp Richard M. Low, APPLIED CRYPTANALYSIS Breaking Ciphers in the Real World.

[12] Mark Stamp, Information Security: Principles and Practice.

[13] Mehak Khurana , Meena Kumari, Variants of Differential and Linear Cryptanalysis

[14] Samuel S. Wagstaff, Jr., CRYPTANALYSIS of NUMBER THEORETIC CIPHERS.

[15] Ρούπα Παρασκευή, Κρυπταναλυτικές Μέθοδοι, 5: Διαφορική Κρυπτανάλυση ΣΣΕ 2022-2023.