ΑΝΑΦΟΡΑ 1ου ΠΑΡΑΔΟΤΕΟΥ
ΤΟΥΛΚΕΡΙΔΗΣ ΝΙΚΟΣ 10718 ΗΜΜΥ
11/2025

1. ΚΡΥΠΤΟΑΝΑΛΥΣΗ VIGENERE CIPHER

ΕΙΣΑΓΩΓΗ

Στην πρώτη εργασία μας δόθηκε το κρυπτογράφημα "**LXFOPVEFRNHR**" και μας ζητήθηκε να ανακτήσουμε το κλειδί κρυπτογράφησης Vigenère, γνωρίζοντας ότι το **μήκος** του κλειδιού είναι **5** γράμματα και ότι το αρχικό κείμενο είναι αγγλικό χωρίς σημεία στίξης.

ΜΕΘΟΔΟΛΟΓΙΑ

Για την επίλυση του προβλήματος, ακολούθησα τα παρακάτω βήματα:

1. ΚΑΝΟΝΙΚΟΠΟΙΗΣΗ ΚΡΥΠΤΟΓΡΑΦΗΜΑΤΟΣ
   Αρχικά, δημιούργησα τη συνάρτηση normalize_ciphertext() που κρατάει μόνο αλφαβητικούς χαρακτήρες και τους μετατρέπει σε κεφαλαία. Το κρυπτογράφημα μας είναι ήδη καθαρό: "LXFOPVEFRNHR"

2. ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ VIGENERE
   Υλοποίησα τη συνάρτηση decrypt_vigenere() που δέχεται ένα κρυπτογράφημα και ένα υποψήφιο κλειδί. Για κάθε χαρακτήρα του κρυπτογραφήματος:
   - Υπολογίζω τη θέση του στο αλφάβητο (0-25)
   - Αφαιρώ την αντίστοιχη θέση του χαρακτήρα του κλειδιού (με modulo)
   - Μετατρέπω το αποτέλεσμα πίσω σε γράμμα

3. ΣΤΑΤΙΣΤΙΚΗ ΑΝΑΛΥΣΗ - CHI-SQUARE TEST
   Για να αξιολογήσω αν ένα αποκρυπτογραφημένο κείμενο είναι έγκυρο αγγλικό κείμενο, χρησιμοποίησα το τεστ χι-τετράγωνο (chi-square test). Η συνάρτηση chi_square_score() συγκρίνει τη συχνότητα εμφάνισης των γραμμάτων στο αποκρυπτογραφημένο κείμενο με την αναμενόμενη συχνότητα στην αγγλική γλώσσα. **Χαμηλότερη βαθμολογία σημαίνει καλύτερη αντιστοιχία με την αγγλική**.

4. ΛΕΞΙΚΟ ΥΠΟΨΗΦΙΩΝ ΚΛΕΙΔΙΩΝ
   Αντί να δοκιμάσω όλους τους πιθανούς συνδυασμούς 5 γραμμάτων (26^5 =
   11,881,376 περιπτώσεις), φόρτωσα ένα λεξικό με πραγματικές αγγλικές λέξεις 5 γραμμάτων από το αρχείο "wordlist_5_letter.txt". Αυτό μειώνει δραματικά τον χώρο αναζήτησης και βασίζεται στην **παραδοχή** ότι το κλειδί είναι μια έγκυρη αγγλική λέξη.

5. ΒΑΘΜΟΛΟΓΗΣΗ ΚΑΙ ΚΑΤΑΤΑΞΗ
   Η συνάρτηση rank_keys_by_plaintext() δοκιμάζει όλες τις υποψήφιες λέξεις ως κλειδιά, αποκρυπτογραφεί το κείμενο με το καθένα, υπολογίζει το chi-

square score και επιστρέφει τα 5 καλύτερα αποτελέσματα ταξινομημένα κατά αύξουσα βαθμολογία.

ΤΕΛΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Το πρόγραμμα εντόπισε το σωστό κλειδί και αποκρυπτογράφησε επιτυχώς το μήνυμα:

ΒΕΛΤΙΣΤΟ ΚΛΕΙΔΙ: **LEMON**
ΒΑΘΜΟΛΟΓΙΑ: 33.12
ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΜΕΝΟ ΜΗΝΥΜΑ: **ATTACKATDAWN**

Το αποτέλεσμα "ATTACKATDAWN" (Attack at Dawn) είναι σαφώς ένα νοηματικό αγγλικό μήνυμα, επιβεβαιώνοντας την ορθότητα του κλειδιού.

Άλλα πιθανά κλειδιά που εξετάστηκαν (με χειρότερες βαθμολογίες):

2. AMBER: score=33.63, plaintext=LLEKYVSENWHF (μη νοηματικό)
3. DREAM: score=52.29, plaintext=IGBODSNBRBEA (μη νοηματικό)
4. ZEBRA: score=68.63, plaintext=MTEXPWAEANIN (μη νοηματικό)
5. BLACK: score=73.07, plaintext=KMFMFUTFPDGG (μη νοηματικό)

Παρατηρούμε ότι το "LEMON" έχει τη χαμηλότερη βαθμολογία chi-square (33.12), υποδεικνύοντας την καλύτερη αντιστοιχία με τη στατιστική κατανομή της αγγλικής γλώσσας. Το αμέσως επόμενο κλειδί **"AMBER"** έχει score 33.63, πολύ κοντά, αλλά παράγει ένα **εντελώς ακατανόητο κείμενο**.

ΕΠΑΛΗΘΕΥΣΗ

Για να επαληθεύσουμε το αποτέλεσμα, μπορούμε να κρυπτογραφήσουμε το "ATTACKATDAWN" με το κλειδί "LEMON":

A(0) + L(11) mod 26 = 11 = **L**
T(19) + E(4) mod 26 = 23 = **X**
T(19) + M(12) mod 26 = 5 = **F**
A(0) + O(14) mod 26 = 14 = **O**
C(2) + N(13) mod 26 = 15 = **P**
K(10) + L(11) mod 26 = 21 = **V**
A(0) + E(4) mod 26   = 4 = **E**
T(19) + M(12) mod 26 = 5 = **F**
D(3) + O(14) mod 26 = 17 = **R**
A(0) + N(13) mod 26 = 13 = **N**
W(22) + L(11) mod 26 = 7 = **H**
N(13) + E(4) mod 26 = 17 = **R**

Το αποτέλεσμα είναι πράγματι "**LXFOPVEFRNHR**", επιβεβαιώνοντας την ορθότητα.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η προσέγγιση που ακολουθήθηκε ήταν επιτυχής:

1. Η χρήση πραγματικού λεξικού αγγλικών λέξεων μείωσε δραστικά τον χώρο αναζήτησης και οδήγησε σε γρήγορη εύρεση του σωστού κλειδιού.

2. Το στατιστικό τεστ chi-square αποδείχθηκε αξιόπιστος δείκτης για την αναγνώριση έγκυρου αγγλικού κειμένου.

3. Το κλειδί "LEMON" και το αποκρυπτογραφημένο μήνυμα "ATTACKATDAWN" εντοπίστηκαν με βεβαιότητα.

2. RSA ΚΡΥΠΤΟΣΥΣΤΗΜΑ ΜΕ ΤΑΧΕΙΣ ΥΨΩΣΕΙΣ

ΕΙΣΑΓΩΓΗ

Στη δεύτερη εργασία μας ζητήθηκε να υλοποιήσουμε έναν πλήρη κύκλο κρυπτογράφησης και αποκρυπτογράφησης με το κρυπτοσύστημα RSA. Δόθηκαν οι πρώτοι αριθμοί **p=197** και **q=211**, ο δημόσιος εκθέτης **e=24377**, και το μήνυμα **m=1234**.

Ζητούμενα:
α) Υπολογισμός n, φ(n) και εύρεση ιδιωτικού εκθέτη d
β) Κρυπτογράφηση του μηνύματος και αποκρυπτογράφηση για επαλήθευση
γ) Εμφάνιση βημάτων square-and-multiply για έναν από τους εκθέτες

ΜΕΘΟΔΟΛΟΓΙΑ

Για την επίλυση του προβλήματος υλοποίησα τις παρακάτω συναρτήσεις:

1. ΕΠΕΚΤΑΜΕΝΟΣ ΑΛΓΟΡΙΘΜΟΣ ΕΥΚΛΕΙΔΗ (Extended GCD)
   Η συνάρτηση extended_gcd(a, b) υπολογίζει το μέγιστο κοινό διαιρέτη και τους συντελεστές x, y τέτοιους ώστε: ax + by = gcd(a, b)

   Αυτό είναι απαραίτητο για την εύρεση του πολλαπλασιαστικού αντίστροφου.

2. ΠΟΛΛΑΠΛΑΣΙΑΣΤΙΚΟΣ ΑΝΤΙΣΤΡΟΦΟΣ (Modular Inverse)
   Η συνάρτηση mod_inverse(a, modulus) χρησιμοποιεί τον επεκταμένο αλγόριθμο Ευκλείδη για να βρει τον αριθμό d τέτοιο ώστε: (a * d) mod modulus = 1.
   Στο RSA, χρειάζεται για να υπολογίσουμε το d από το e και το φ(n).

3. SQUARE-AND-MULTIPLY ΜΕ ΚΑΤΑΓΡΑΦΗ ΒΗΜΑΤΩΝ
Η συνάρτηση square_and_multiply(base, exponent, modulus) υλοποιεί τον αλγόριθμο ταχείας ύψωσης σε δύναμη με modulo. Ο αλγόριθμος:

α) Μετατρέπει τον εκθέτη σε δυαδική μορφή
β) Για κάθε bit (από αριστερά προς τα δεξιά):
- Τετραγωνίζει το τρέχον αποτέλεσμα
- Αν το bit είναι 1, πολλαπλασιάζει με τη βάση
γ) Καταγράφει κάθε βήμα για διαφάνεια

Αυτός ο αλγόριθμος είναι εξαιρετικά αποδοτικός για μεγάλους εκθέτες.

4. RSA KEY GENERATION
Ακολούθησα τα κλασικά βήματα δημιουργίας κλειδιών RSA:
- n = p * q
- φ(n) = (p-1) * (q-1)
- d = e^(-1) mod φ(n)

5. ΚΡΥΠΤΟΓΡΑΦΗΣΗ ΚΑΙ ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ
- Κρυπτογράφημα: c = m^e mod n
- Αποκρυπτογράφηση: m' = c^d mod n

ΥΠΟΛΟΓΙΣΜΟΙ ΚΑΙ ΕΝΔΙΑΜΕΣΑ ΒΗΜΑΤΑ

ΒΗΜΑ 1: ΥΠΟΛΟΓΙΣΜΟΣ ΠΑΡΑΜΕΤΡΩΝ RSA

Δοσμένα: p = 197, q = 211, e = 24377

n = p × q = 197 × 211 = **41567**

φ(n) = (p-1) × (q-1) = 196 × 210 = **41160**

Για την εύρεση του d, πρέπει να λύσουμε:
e × d ≡ 1 (mod φ(n))
24377 × d ≡ 1 (mod 41160)

Χρησιμοποιώντας τον επεκταμένο αλγόριθμο Ευκλείδη:
d = 17393

Επαλήθευση: (24377 × 17393) mod 41160 = 1 ✓

ΒΗΜΑ 2: ΚΡΥΠΤΟΓΡΑΦΗΣΗ ΤΟΥ ΜΗΝΥΜΑΤΟΣ

Μήνυμα: m = 1234
Κρυπτογράφηση: c = m^e mod n = 1234^24377 mod 41567

Χρησιμοποιώντας square-and-multiply:
c = 26476

ΒΗΜΑ 3: ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ ΤΟΥ ΚΡΥΠΤΟΓΡΑΦΗΜΑΤΟΣ

Κρυπτογράφημα: c = 26476
Αποκρυπτογράφηση: m' = c^d mod n = 26476^17393 mod 41567

Χρησιμοποιώντας square-and-multiply:
m' = 1234 ✓

Το αποκρυπτογραφημένο μήνυμα ταυτίζεται με το αρχικό, επιβεβαιώνοντας την ορθότητα των υπολογισμών!

ΑΝΑΛΥΤΙΚΑ ΒΗΜΑΤΑ SQUARE-AND-MULTIPLY ΓΙΑ ΚΡΥΠΤΟΓΡΑΦΗΣΗ (εκθέτης e)

Εκθέτης e = 24377 σε δυαδική μορφή: 101.1111.0011.1001

Αρχική τιμή result = 1
Βάση base = 1234
Modulo n = 41567

Βήμα προς βήμα:

Βήμα 0: bit=1
  Square: 1^2 mod 41567 = 1
  Multiply (bit=1): 1 × 1234 mod 41567 = 1234

Βήμα 1: bit=0
  Square: 1234^2 mod 41567 = 26344
  (Δεν πολλαπλασιάζουμε γιατί bit=0)

Βήμα 2: bit=1
  Square: 26344^2 mod 41567 = 3704
  Multiply: 3704 × 1234 mod 41567 = 39933

Βήμα 3: bit=1
  Square: 39933^2 mod 41567 = 9668
  Multiply: 9668 × 1234 mod 41567 = 583

Βήμα 4: bit=1
  Square: 583^2 mod 41567 = 7353
  Multiply: 7353 × 1234 mod 41567 = 11996

Βήμα 5: bit=1
  Square: 11996^2 mod 41567 = 40629
  Multiply: 40629 × 1234 mod 41567 = 6384

Βήμα 6: bit=1
  Square: 6384^2 mod 41567 = 19796

Multiply: 19796 × 1234 mod 41567 = 28435

Βήμα 7: bit=0
Square: 28435² mod 41567 = 29508

Βήμα 8: bit=0
Square: 29508² mod 41567 = 18115

Βήμα 9: bit=1
Square: 18115² mod 41567 = 23327
Multiply: 23327 × 1234 mod 41567 = 21154

Βήμα 10: bit=1
Square: 21154² mod 41567 = 22961
Multiply: 22961 × 1234 mod 41567 = 26747

Βήμα 11: bit=1
Square: 26747² mod 41567 = 33939
Multiply: 33939 × 1234 mod 41567 = 22757

Βήμα 12: bit=0
Square: 22757² mod 41567 = 39363

Βήμα 13: bit=0
Square: 39363² mod 41567 = 35844

Βήμα 14: bit=1 (τελευταίο bit)
Square: 35844² mod 41567 = 39500
Multiply: 39500 × 1234 mod 41567 = 26476

ΤΕΛΙΚΟ ΑΠΟΤΕΛΕΣΜΑ: **c = 26476**

ΑΝΑΛΥΤΙΚΑ ΒΗΜΑΤΑ SQUARE-AND-MULTIPLY ΓΙΑ ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ (εκθέτης d)

Εκθέτης d = 17393 σε δυαδική μορφή: 100.0011.1111.0001

Αρχική τιμή result = 1
Βάση base = 26476 (το κρυπτογράφημα)
Modulo n = 41567

Βήμα προς βήμα:

Βήμα 0: bit=1
Square: 1² mod 41567 = 1
Multiply: 1 × 26476 mod 41567 = 26476

Βήμα 1: bit=0
Square: 26476² mod 41567 = 34255

Βήμα 2: bit=0
  Square: $34255^2 \bmod 41567 = 10182$

Βήμα 3: bit=0
  Square: $10182^2 \bmod 41567 = 5026$

Βήμα 4: bit=0
  Square: $5026^2 \bmod 41567 = 29507$

Βήμα 5: bit=1
  Square: $29507^2 \bmod 41567 = 667$
  Multiply: $667 \times 26476 \bmod 41567 = 35084$

Βήμα 6: bit=1
  Square: $35084^2 \bmod 41567 = 5052$
  Multiply: $5052 \times 26476 \bmod 41567 = 35713$

Βήμα 7: bit=1
  Square: $35713^2 \bmod 41567 = 18108$
  Multiply: $18108 \times 26476 \bmod 41567 = 35197$

Βήμα 8: bit=1
  Square: $35197^2 \bmod 41567 = 7508$
  Multiply: $7508 \times 26476 \bmod 41567 = 8414$

Βήμα 9: bit=1
  Square: $8414^2 \bmod 41567 = 6795$
  Multiply: $6795 \times 26476 \bmod 41567 = 2444$

Βήμα 10: bit=1
  Square: $2444^2 \bmod 41567 = 29055$
  Multiply: $29055 \times 26476 \bmod 41567 = 21278$

Βήμα 11: bit=0
  Square: $21278^2 \bmod 41567 = 5520$

Βήμα 12: bit=0
  Square: $5520^2 \bmod 41567 = 1789$

Βήμα 13: bit=0
  Square: $1789^2 \bmod 41567 = 41429$

Βήμα 14: bit=1 (τελευταίο bit)
  Square: $41429^2 \bmod 41567 = 19044$
  Multiply: $19044 \times 26476 \bmod 41567 = 1234$

ΤΕΛΙΚΟ ΑΠΟΤΕΛΕΣΜΑ: m' = 1234

ΤΕΛΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

ΠΑΡΑΜΕΤΡΟΙ RSA:
- n = 41567
- φ(n) = 41160
- Δημόσιο κλειδί: (e, n) = (24377, 41567)
- Ιδιωτικό κλειδί: (d, n) = (17393, 41567)

ΚΡΥΠΤΟΓΡΑΦΗΣΗ:
- Αρχικό μήνυμα: m = 1234
- Κρυπτογράφημα: c = 26476

ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ:
- Κρυπτογράφημα: c = 26476
- Αποκρυπτογραφημένο μήνυμα: m' = 1234 ✓

ΕΠΑΛΗΘΕΥΣΗ:
m = m' = 1234, επιβεβαιώνοντας την ορθότητα της υλοποίησης!

ΑΝΑΛΥΣΗ ΑΠΟΔΟΤΙΚΟΤΗΤΑΣ SQUARE-AND-MULTIPLY

Ο αλγόριθμος square-and-multiply είναι εξαιρετικά αποδοτικός:

Για την κρυπτογράφηση:
- Εκθέτης e = 24377
- Δυαδική αναπαράσταση: 15 bits
- Πράξεις που εκτελέστηκαν: 15 τετραγωνισμοί + 10 πολλαπλασιασμοί = **25 πράξεις**
- Naive approach: 24377 πολλαπλασιασμοί!

Για την αποκρυπτογράφηση:
- Εκθέτης d = 17393
- Δυαδική αναπαράσταση: 15 bits
- Πράξεις που εκτελέστηκαν: 15 τετραγωνισμοί + 8 πολλαπλασιασμοί = **23 πράξεις**
- Naive approach: 17393 πολλαπλασιασμοί!

3. ΤΡΟΠΟΙ ΛΕΙΤΟΥΡΓΙΑΣ ΜΠΛΟΚ CBC & CFB: ΔΙΑΔΟΣΗ ΣΦΑΛΜΑΤΟΣ

ΕΙΣΑΓΩΓΗ

Στην τρίτη εργασία μας ζητήθηκε να μελετήσουμε τη διάδοση σφαλμάτων σε δύο τρόπους λειτουργίας κρυπτογράφησης μπλοκ: **CBC** (Cipher Block Chaining) και **CFB** (Cipher Feedback).

Δεδομένα:
- Block cipher 128-bit (Ek)
- Δύο μπλοκ plaintext: P1, P2
- Initialization Vector (IV) = C0
- Ένα μόνο bit του C1 αλλοιώνεται κατά τη μετάδοση

Ζητούμενο:
Να περιγράψουμε ακριβώς τι παθαίνουν τα P1' και P2' στην αποκρυπτογράφηση για CBC και CFB (ποια/πόσα bits επηρεάζονται και γιατί).

ΤΥΠΟΙ

1. CBC MODE (Cipher Block Chaining)

   ΚΡΥΠΤΟΓΡΑΦΗΣΗ:
   C1 = Ek(P1 ⊕ IV)
   C2 = Ek(P2 ⊕ C1)

   ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ:
   P1 = Dk(C1) ⊕ IV
   P2 = Dk(C2) ⊕ C1

2. CFB MODE (Cipher Feedback)

   ΚΡΥΠΤΟΓΡΑΦΗΣΗ:
   C1 = P1 ⊕ Ek(IV)
   C2 = P2 ⊕ Ek(C1)

   ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ:
   P1 = C1 ⊕ Ek(IV)
   P2 = C2 ⊕ Ek(C1)

ΜΕΘΟΔΟΛΟΓΙΑ

Για να μελετήσω τη διάδοση σφαλμάτων, υλοποίησα:

1. FEISTEL BLOCK CIPHER (128-bit)
   Δημιούργησα ένα custom block cipher με δομή Feistel:
   - Χρήση BLAKE2s για key derivation και round function
   - 8 rounds για επαρκή διάχυση (diffusion)
   - Μέγεθος μπλοκ: 128 bits (16 bytes)
   - Διασφαλίζει πλήρη diffusion: αλλαγή 1 bit εισόδου → ~50% αλλαγή εξόδου

2. CBC ENCRYPTION/DECRYPTION
   Υλοποίησα τις κλασικές συναρτήσεις:
   - cbc_encrypt(): Κρυπτογραφεί με chaining των blocks
   - cbc_decrypt(): Αποκρυπτογραφεί και κάνει XOR με προηγούμενο C

3. CFB ENCRYPTION/DECRYPTION
   Υλοποίησα τις συναρτήσεις stream-cipher style:
   - cfb_encrypt(): Χρησιμοποιεί Ek για keystream generation
   - cfb_decrypt(): Ίδια λογική με την κρυπτογράφηση

4. ERROR INJECTION & ANALYSIS
   - Δημιουργώ "καθαρό" ciphertext
   - Αναστρέφω το bit 17 του C1 (simulating transmission error)
   - Αποκρυπτογραφώ με το corrupted C1
   - Μετρώ πόσα bits διαφέρουν στα P1' και P2' από τα αρχικά

ΕΝΔΙΑΜΕΣΑ ΒΗΜΑΤΑ ΚΑΙ ΠΑΡΑΤΗΡΗΣΕΙΣ

ΑΡΧΙΚΑ ΔΕΔΟΜΕΝΑ:

Κλειδί: key = "task3_demo_key!!" (16 bytes = 128 bits)
IV: iv = "task3_demo_iv__" (16 bytes = 128 bits)

Plaintext blocks:
P1 = "Plaintext block1" = 0x506c61696e7465787420626c6f636b31
P2 = "Plaintext block2" = 0x506c61696e7465787420626c6f636b32

Σφάλμα μετάδοσης: Flip bit 17 του C1 (θέση 17 από το LSB)

ΑΝΑΛΥΣΗ CBC MODE

ΘΕΩΡΗΤΙΚΗ ΑΝΑΛΥΣΗ:

Όταν το C1 καταφτάνει με 1 bit σφάλμα:

Για το P1':
  P1' = Dk(C1_corrupted) ⊕ IV

Το C1_corrupted περνάει μέσα από την αποκρυπτογράφηση Dk. Επειδή το block cipher έχει καλή diffusion property, η αλλαγή ενός bit στην είσοδο του Dk προκαλεί αλλαγή σε ~50% των bits της εξόδου. Αυτό σημαίνει ότι το Dk(C1_corrupted) θα διαφέρει σημαντικά από το Dk(C1_clean), επηρεάζοντας πολλά bits του P1'.

Για το P2':
  P2' = Dk(C2) ⊕ C1_corrupted

Το C2 παραμένει αναλλοίωτο, άρα το Dk(C2) είναι σωστό. Ωστόσο, το XOR γίνεται με το C1_corrupted αντί για το C1_clean. Άρα:

  P2_clean = Dk(C2) ⊕ C1_clean
  P2'   = Dk(C2) ⊕ C1_corrupted

  P2_clean ⊕ P2' = C1_clean ⊕ C1_corrupted = 1 bit διαφορά

Δηλαδή, στο P2' επηρεάζεται μόνο το ίδιο bit που άλλαξε στο C1!

ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ:

CBC mode (bit flip at position 17):
** P1' differs in 62/128 bits (48.4%)**
** P2' differs in 1/128 bits (0.78%)**

Επεξήγηση:
- Το P1' είναι σχεδόν πλήρως κατεστραμμένο λόγω της diffusion του block cipher
- Το P2' έχει μόνο 1 bit λάθος (το bit που αλλοίωσε στο C1)

ΣΥΜΠΕΡΑΣΜΑ CBC:
"Το P1 καταστρέφεται ολοσχερώς (unusable), αλλά το σφάλμα μεταδίδεται
στο P2 με ελάχιστη επίδραση (1 bit μόνο)."

ΑΝΑΛΥΣΗ CFB MODE

ΘΕΩΡΗΤΙΚΗ ΑΝΑΛΥΣΗ:

Όταν το C1 καταφτάνει με 1 bit σφάλμα:

Για το P1':
  P1' = C1_corrupted ⊕ Ek(IV)

Το Ek(IV) παραμένει το ίδιο (το IV δεν άλλαξε). Άρα:

  P1_clean = C1_clean ⊕ Ek(IV)
  P1'   = C1_corrupted ⊕ Ek(IV)

  P1_clean ⊕ P1' = C1_clean ⊕ C1_corrupted = 1 bit διαφορά

Το σφάλμα παραμένει εντοπισμένο στο ίδιο bit!

Για το P2':
  P2' = C2 ⊕ Ek(C1_corrupted)

Το πρόβλημα είναι ότι το keystream για το P2 παράγεται από το Ek(C1_corrupted) αντί για Ek(C1_clean). Επειδή το block cipher έχει καλή diffusion, η αλλαγή 1 bit στο C1 προκαλεί αλλαγή ~50% των bits στο Ek(C1). Άρα:

  Ek(C1_clean) ≠≠≠ Ek(C1_corrupted) (διαφέρουν σε ~50% των bits)

Αυτό σημαίνει ότι το keystream είναι λάθος και το P2' θα είναι σχεδόν εντελώς διαφορετικό από το P2_clean.

ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ:

CFB mode (bit flip at position 17):
** P1' differs in 1/128 bits (0.78%)**
** P2' differs in 63/128 bits (49.2%)**

Επεξήγηση:
- Το P1' έχει μόνο 1 bit λάθος (το bit που αλλοίωσε στο C1)
- Το P2' είναι σχεδόν πλήρως κατεστραμμένο λόγω λάθους keystream

ΣΥΜΠΕΡΑΣΜΑ CFB:
"Το P1 διατηρεί το σφάλμα τοπικό (1 bit), αλλά το P2 καταστρέφεται ολοσχερώς επειδή το keystream άλλαξε παντού."

ΣΥΓΚΡΙΤΙΚΗ ΑΝΑΛΥΣΗ

ΒΑΣΙΚΕΣ ΔΙΑΦΟΡΕΣ:

1. ΕΠΙΔΡΑΣΗ ΣΤΟ P1:
   CBC: Το σφάλμα περνά από Dk → πλήρης diffusion → P1' unusable
   CFB: Το σφάλμα είναι στο XOR → τοπική επίδραση → P1' σχεδόν σωστό

2. ΕΠΙΔΡΑΣΗ ΣΤΟ P2:
   CBC: Το σφάλμα είναι στο XOR mask → τοπική επίδραση → P2' σχεδόν σωστό
   CFB: Το σφάλμα περνά από Ek → πλήρης diffusion → P2' unusable

3. PATTERN ΔΙΑΔΟΣΗΣ:
   CBC: Catastrophic → Self-healing (το σφάλμα "απορροφάται")
   CFB: Self-limiting → Catastrophic (το σφάλμα "εκρήγνυται")

4. ΣΥΝΟΛΙΚΗ ΕΠΙΔΡΑΣΗ:
   CBC: ~63 bits κατεστραμμένα συνολικά (P1 κυρίως)
   CFB: ~64 bits κατεστραμμένα συνολικά (P2 κυρίως)

ΤΕΛΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

CBC MODE:
- Αρχικά blocks: P1 και P2 (από 128 bits το καθένα)
- Σφάλμα: 1 bit flip στο C1 (position 17)
- Αποτέλεσμα αποκρυπτογράφησης:
  * P1': 62 bits διαφέρουν (48.4% του block) - ΚΑΤΕΣΤΡΑΜΜΕΝΟ
  * P2': 1 bit διαφέρει (0.78% του block) - ΣΧΕΔΟΝ ΑΝΕΠΗΡΕΑΣΤΟ

CFB MODE:
- Αρχικά blocks: P1 και P2 (από 128 bits το καθένα)
- Σφάλμα: 1 bit flip στο C1 (position 17)
- Αποτέλεσμα αποκρυπτογράφησης:
  * P1': 1 bit διαφέρει (0.78% του block) - ΣΧΕΔΟΝ ΑΝΕΠΗΡΕΑΣΤΟ
  * P2': 63 bits διαφέρουν (49.2% του block) - ΚΑΤΕΣΤΡΑΜΜΕΝΟ

ΕΡΜΗΝΕΙΑ ΚΑΙ ΠΡΑΚΤΙΚΕΣ ΕΠΙΠΤΩΣΕΙΣ

1. ERROR RECOVERY:

CBC: Το σφάλμα είναι "self-healing". Μετά από 2 blocks, η μετάδοση επανέρχεται στο φυσιολογικό. Ο δέκτης μπορεί να απορρίψει το P1 και να συνεχίσει με σχεδόν σωστό P2.

CFB: Το σφάλμα είναι "self-limiting" αλλά με delayed effect. Το P1 είναι σχεδόν σωστό, αλλά το P2 καταστρέφεται. Μετά το P2, η μετάδοση επανέρχεται.

2. ΧΡΗΣΗ ΣΕ NOISY CHANNELS:

CBC: Προτιμότερο όταν θέλουμε γρήγορη ανάκαμψη από σφάλματα. Το πρώτο block μετά το σφάλμα καταστρέφεται, αλλά τα επόμενα είναι σχεδόν καλά.

CFB: Καλύτερο για εφαρμογές όπου κάθε bit έχει σημασία και θέλουμε να περιορίσουμε την αρχική ζημιά, ακόμα και αν το επόμενο block πάθει.

3. ERROR DETECTION:

Και τα δύο modes έχουν παρόμοια συνολική επίδραση (~64 bits corruption), αλλά με διαφορετική κατανομή. Για error detection, αυτό σημαίνει:

CBC: Αν δούμε garbage στο P1', ξέρουμε ότι υπήρξε πρόβλημα στο C1
CFB: Αν δούμε garbage στο P2', ξέρουμε ότι υπήρξε πρόβλημα στο C1

4. ELGAMAL ΑΡΙΘΜΗΤΙΚΟ ΠΑΡΑΔΕΙΓΜΑ ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗΣ

ΕΙΣΑΓΩΓΗ

Στην τέταρτη εργασία μας ζητήθηκε να υλοποιήσουμε έναν πλήρη κύκλο αποκρυπτογράφησης με το κρυπτοσύστημα ElGamal.

Δεδομένα:
- Πρώτος αριθμός: p = 23
- Γεννήτορας: g = 5
- Ιδιωτικό κλειδί: x = 6
- Δημόσιο κλειδί: y = g^x mod p
- Κρυπτογράφημα: C = (C1, C2) = (20, 22)

Ζητούμενα:
α) Υπολογισμός του C1^x mod p
β) Εύρεση του πολλαπλασιαστικού αντίστροφου του C1^x mod p
γ) Ανάκτηση του μηνύματος M και εμφάνιση ενδιάμεσων βημάτων

ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ - ELGAMAL ΚΡΥΠΤΟΣΥΣΤΗΜΑ

Το ElGamal είναι ένα κρυπτοσύστημα δημόσιου κλειδιού που βασίζεται στο πρόβλημα του διακριτού λογαρίθμου.

ΔΗΜΙΟΥΡΓΙΑ ΚΛΕΙΔΙΩΝ:
1. Επιλέγεται ένας μεγάλος πρώτος p και γεννήτορας g της Z*p
2. Επιλέγεται τυχαίο ιδιωτικό κλειδί x ∈ {1, 2, ..., p-2}
3. Υπολογίζεται το δημόσιο κλειδί: y = g^x mod p
4. Δημόσιο κλειδί: (p, g, y)
5. Ιδιωτικό κλειδί: x

ΚΡΥΠΤΟΓΡΑΦΗΣΗ (από τον αποστολέα):
Για να κρυπτογραφήσει μήνυμα M:
1. Επιλέγει τυχαίο k ∈ {1, 2, ..., p-2}
2. Υπολογίζει: C1 = g^k mod p
3. Υπολογίζει: C2 = M · y^k mod p
4. Κρυπτογράφημα: C = (C1, C2)

ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ (από τον παραλήπτη):
Για να αποκρυπτογραφήσει το C = (C1, C2):
1. Υπολογίζει το shared secret: s = C1^x mod p
2. Βρίσκει τον αντίστροφο: s^(-1) mod p
3. Ανακτά το μήνυμα: M = C2 · s^(-1) mod p

ΜΑΘΗΜΑΤΙΚΗ ΕΠΑΛΗΘΕΥΣΗ:
Το σύστημα λειτουργεί επειδή:
  s = C1^x mod p = (g^k)^x mod p = g^(kx) mod p
  y^k mod p = (g^x)^k mod p = g^(kx) mod p

Άρα: C2 · s^(-1) = M · y^k · (C1^x)^(-1)
                = M · g^(kx) · (g^(kx))^(-1)
                = M · 1 = M

ΜΕΘΟΔΟΛΟΓΙΑ

Για την επίλυση του προβλήματος υλοποίησα:

1. ΕΠΕΚΤΑΜΕΝΟΣ ΑΛΓΟΡΙΘΜΟΣ ΕΥΚΛΕΙΔΗ (Extended Euclidean Algorithm)
   Η συνάρτηση extended_euclid(a, b) υπολογίζει:
   - Το gcd(a, b)
   - Τους συντελεστές Bézout s και t τέτοιους ώστε: a·s + b·t = gcd(a,b)
   - Καταγράφει κάθε βήμα του αλγορίθμου για διαφάνεια

2. ΠΟΛΛΑΠΛΑΣΙΑΣΤΙΚΟΣ ΑΝΤΙΣΤΡΟΦΟΣ (Modular Inverse)
   Η συνάρτηση mod_inverse(a, modulus) χρησιμοποιεί τον επεκταμένο αλγόριθμο Ευκλείδη για να βρει τον αριθμό inv τέτοιο ώστε:
   (a · inv) mod modulus = 1

3. ΤΑΧΕΙΑ ΥΨΩΣΗ ΣΕ ΔΥΝΑΜΗ (Fast Exponentiation)
   Χρησιμοποίησα την ενσωματωμένη συνάρτηση pow(base, exp, mod) της Python που υλοποιεί τον αλγόριθμο square-and-multiply για αποδοτικούς υπολογισμούς.

4. ELGAMAL ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗ
   Ακολούθησα τα βήματα του πρωτοκόλλου:
   - Υπολογισμός shared secret
   - Εύρεση αντίστροφου
   - Ανάκτηση plaintext

ΑΝΑΛΥΤΙΚΟΙ ΥΠΟΛΟΓΙΣΜΟΙ ΚΑΙ ΕΝΔΙΑΜΕΣΑ ΒΗΜΑΤΑ

ΔΟΣΜΕΝΕΣ ΠΑΡΑΜΕΤΡΟΙ:

Πρώτος: p = 23
Γεννήτορας: g = 5
Ιδιωτικό κλειδί: x = 6
Κρυπτογράφημα: C = (C1, C2) = (20, 22)

ΒΗΜΑ 0: ΥΠΟΛΟΓΙΣΜΟΣ ΔΗΜΟΣΙΟΥ ΚΛΕΙΔΙΟΥ

Το δημόσιο κλειδί είναι:
y = g^x mod p = 5^6 mod 23

Υπολογισμός του 5^6 mod 23:
5^1 = 5
5^2 = 25 mod 23 = 2
5^3 = 5 · 2 = 10 mod 23 = 10
5^4 = 5 · 10 = 50 mod 23 = 4
5^5 = 5 · 4 = 20 mod 23 = 20
5^6 = 5 · 20 = 100 mod 23 = 8

Άρα: **y = 8**

ΒΗΜΑ α: ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΥ C1^x mod p

Shared secret = C1^x mod p = 20^6 mod 23

Υπολογισμός του 20^6 mod 23:
20^1 = 20
20^2 = 400 mod 23 = 8
20^3 = 20 · 8 = 160 mod 23 = 22
20^4 = 20 · 22 = 440 mod 23 = 2
20^5 = 20 · 2 = 40 mod 23 = 17
20^6 = 20 · 17 = 340 mod 23 = 16

Άρα: **C1^x mod p = 16**

ΒΗΜΑ β: ΕΥΡΕΣΗ ΑΝΤΙΣΤΡΟΦΟΥ ΤΟΥ 16 mod 23

Πρέπει να βρούμε τον αριθμό inv τέτοιο ώστε:
16 · inv ≡ 1 (mod 23)

Χρησιμοποιούμε τον επεκταμένο αλγόριθμο Ευκλείδη για τα (16, 23).

ΒΗΜΑ γ: ΑΝΑΛΥΤΙΚΑ ΒΗΜΑΤΑ ΕΠΕΚΤΑΜΕΝΟΥ ΑΛΓΟΡΙΘΜΟΥ ΕΥΚΛΕΙΔΗ

Ο επεκταμένος αλγόριθμος Ευκλείδη βρίσκει συντελεστές s, t τέτοιους ώστε:
16 · s + 23 · t = gcd(16, 23) = 1

Αρχικοποίηση:
$r_0 = 23$, $s_0 = 0$, $t_0 = 1$
$r_1 = 16$, $s_1 = 1$, $t_1 = 0$

Επανάληψη 1:
$q_1 = \lfloor 23 / 16 \rfloor = 1$
$r_2 = 23 - 1 \cdot 16 = 7$
$s_2 = 0 - 1 \cdot 1 = -1$
$t_2 = 1 - 1 \cdot 0 = 1$
Έλεγχος: 16·(-1) + 23·1 = -16 + 23 = **7**

Επανάληψη 2:
$q_2 = \lfloor 16 / 7 \rfloor = 2$
$r_3 = 16 - 2 \cdot 7 = 2$
$s_3 = 1 - 2 \cdot (-1) = 3$

$t_3 = 0 - 2 \cdot 1 = -2$
Έλεγχος: $16 \cdot 3 + 23 \cdot (-2) = 48 - 46 = \mathbf{2}$

Επανάληψη 3:
$q_3 = \lfloor 7 / 2 \rfloor = 3$
$r_4 = 7 - 3 \cdot 2 = 1$
$s_4 = -1 - 3 \cdot 3 = -10$
$t_4 = 1 - 3 \cdot (-2) = 7$
Έλεγχος: $16 \cdot (-10) + 23 \cdot 7 = -160 + 161 = \mathbf{1}$

Επανάληψη 4:
$q_4 = \lfloor 2 / 1 \rfloor = 2$
$r_5 = 2 - 2 \cdot 1 = 0$
Τέλος αλγορίθμου (υπόλοιπο = **0**)

ΑΠΟΤΕΛΕΣΜΑ:
gcd(16, 23) = 1
Συντελεστές Bézout: **s = -10, t = 7**
Επαλήθευση: $16 \cdot (-10) + 23 \cdot 7 = -160 + 161 = 1$

Ο **πολλαπλασιαστικός αντίστροφος** του 16 mod 23 είναι:
inv = s mod 23 = -10 mod 23 = **13**

Επαλήθευση: $16 \cdot 13 = 208 = 9 \cdot 23 + 1 \equiv 1 \pmod{23}$

ΠΙΝΑΚΑΣ ΒΗΜΑΤΩΝ EXTENDED EUCLIDEAN:

step	**q**	**r**	**s**	**t**
0	-	23	0	1
1	1	16	1	0
2	2	7	-1	1
3	3	2	3	-2
4	2	1	-10	7
END	-	1	-10	7

ΒΗΜΑ ΑΠΟΚΡΥΠΤΟΓΡΑΦΗΣΗΣ: ΑΝΑΚΤΗΣΗ ΜΗΝΥΜΑΤΟΣ M

M = C2 · (C1^x)^(-1) mod p
M = 22 · 13 mod 23
M = 286 mod 23
M = 286 - 12·23
M = 286 - 276
M = 10

Άρα **το αποκρυπτογραφημένο μήνυμα είναι: M = 10**

ΕΠΑΛΗΘΕΥΣΗ ΟΡΘΟΤΗΤΑΣ

Για να επαληθεύσουμε ότι το M = 10 είναι σωστό, μπορούμε να προσομοιώσουμε την κρυπτογράφηση και να δούμε αν παράγει το ίδιο κρυπτογράφημα.

Έστω ότι ο αποστολέας θέλει να κρυπτογραφήσει M = 10 και επέλεξε k = 5 (το οποίο δεν γνωρίζουμε εκ των προτέρων).

ΕΠΑΛΗΘΕΥΣΗ ΜΕ k = 5:

C1 = g^k mod p = 5^5 mod 23 = **20**

C2 = M · y^k mod p = 10 · 8^5 mod 23

Υπολογισμός 8^5 mod 23:
8^1 = 8
8^2 = 64 mod 23 = 18
8^3 = 8 · 18 = 144 mod 23 = 6
8^4 = 8 · 6 = 48 mod 23 = 2
8^5 = 8 · 2 = 16 mod 23 = 16

C2 = 10 · 16 mod 23 = 160 mod 23 = **22**

Επαλήθευση: C = (20, 22)

Το κρυπτογράφημα ταιριάζει! Άρα το M = 10 είναι το σωστό μήνυμα.