Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

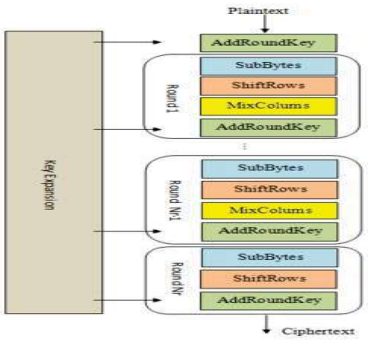
Οι προκλήσεις που αντιμετωπίσαμε αφορούν General, Mathematics, Symmetric Ciphers και Public-Key Cryptography.

Κεφάλαιο 2

Αλγόριθμοι Συμμετρικού Κλειδιού

Score: 505

Στο πρώτο κεφάλαιο θα προτείνουμε λύσεις για προκλήσεις που αφορούν Symmetric Ciphers (block και stream ciphers) με έμφαση τον AES. Στην πρώτη πρόκληση αναφέρεται ότι ο AES χρησιμοποιεί keyed permutation και είναι σημαντικό να είναι ένα-προς-ένα: $f : P \leftrightarrow C$. Το permutation αποτελεί μια από τις δύο τεχνικές για obscuring: το diffusion. Η άλλη είναι το confusion (substitution). Στη δεύτερη πρόκληση αναφέρεται η biclique attack ως μια key-related vulnerability για το AES (από 128->126).



Εικόνα 2.1: Structure of AES

1. SubBytes : substitute from S-box (confusion)
2. Shift Rows : transposition (diffusion)
3. Mix Columns : matrix multiplication

4. AddRoundKey : XoRed with key

2.1 Structure of AES:

$$\underset{4\times4}{\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix}}$$

```
return [list(text[i:i+4]) for i in range(0, len(text), 4)]
```

2.2 Round Keys:

keyed permutation: $A \oplus B$

```
def add_round_key(s, k):
    result = []
    for i in range(len(s)):
        row = []
        for j in range(len(s[i])):
            element = s[i][j] ^ k[i][j]
            row.append(element)
        result.append(row)
    return result
```

2.3 Confusion through Substitution

SubBytes: A cheat code for non linearity.

```
def sub_bytes(s, sbox=s_box):
    result = []
    for row in s:
        sub_row = []
        for element in row:
```

```
            substitution = sbox[element]
            sub_row.append(chr(substitution))
        result.append(sub_row)
    return result
```

2.4 Diffusion through Permutation

ShiftRows: a simple permutation

```
def shift_rows(s):
s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]


def inv_shift_rows(s):
s[0][1], s[3][1], s[2][1], s[1][1] = s[3][1], s[2][1], s[1][1], s[0][1]
s[1][2], s[0][2], s[3][2], s[2][2] = s[3][2], s[2][2], s[1][2], s[0][2]
s[2][3], s[1][3], s[0][3], s[3][3] = s[3][3], s[2][3], s[1][3], s[0][3]
```

Capturing the flag:

```
inv_mix_columns(state)
inv_shift_rows(state)

for i in range(len(state)):
for j in range(len(state[i])):
print(chr(state[i][j]), end='')
```

2.5 Bringing It All Together

Σε αυτή την πρόκληση θα χρησιμοποιήσουμε όλα τα προηγούμενα στάδια και επιπλέον θα αναπτύξουμε μια decrypt function:

```
def decrypt(key, ciphertext):
    round_keys = expand_key(key) # Remember to start from the last round key and work backwa

    # Convert ciphertext to state matrix
    text = bytes2matrix(ciphertext)
```

```
    # Initial add round key step
    add_round_key(text, round_keys[10])

    for i in range(N_ROUNDS - 1, 0, -1):
        inv_shift_rows(text)
        inv_sub_bytes(text)
        add_round_key(text, round_keys[i])
        inv_mix_columns(text)
    # Run final round (skips the InvMixColumns step)
    inv_shift_rows(text)
    inv_sub_bytes(text)
    add_round_key(text, round_keys[0])
    # Convert state matrix to plaintext

    plaintext = matrix2bytes(text)
    return plaintext
```

Σχόλια: Με `N_ROUNDS - 1` ξεκινάμε από τον τελευταίο γύρο (9) και κατευθυνόμαστε προς τον πρώτο γύρο (0). Ωστόσο παρατηρούμε ότι στον τελευταίο γύρο δεν πραγματοποιείται η `inv_mix_columns`.

2.6 Modes of Operation starter

Σε block ciphers ενα ερώτημα που πρέπει να απαντήσω είναι πως θα κρυπτογραφήσω κάτι μεγαλύτερο από ένα single block.
$\Rightarrow Encrypt flag \Rightarrow DecryptCiphertext \Rightarrow HexEncoder.$

2.7 Passwords as Keys

Στο Electronic Codebook Mode (ECB) κάθε block plaintext κρυπτογραφείται ανεξάρτητα. Από ίδια blocks plaintext θα προκύψουν ίδια blocks ciphertext. Αρχικά χρησιμοποιούμε ⇒το Encrypt Flag. Στην συνέχεια θα χρησιμοποιήσουμε το γεγονός ότι το flag είναι της μορφής crypto{sth_to_fill} και παράλληλα θα αξιοποιήσουμε το wordlist που κατεβάσαμε. Κάθε λέξη μέσα από το hash της δίνει ένα κλειδί που χρησιμοποιούμε για να κάνουμε Decrypt το ciphertext. Αν το decrypted περιέχει τη λέξη crypto, σημαίνει ότι βρήκαμε το flag.

```
with open('/home/kali/Desktop/words') as f:
```

```
    words = [w.strip() for w in f.readlines()]
keyword = random.choice(words)

KEY = hashlib.md5(keyword.encode()).digest()

l = len(words)

for i in range(l):
    KEY = hashlib.md5(words[i].encode()).digest()
    decrypted = decrypt("c92b7734070205bdf6c0087a751466ec13ae15e6f1bcdd3f3a535ec0f4bbae66",
    if b'crypto' in decrypted:
        print(decrypted)
```

2.8 ECB Oracle

Με το padding κάνουμε align το plaintext με το μέγεθος του block που γίνεται encrypt. Στην ουσία προσθέτουμε bytes για να 'γεμίσουμε' το block. Από τη γραμμή που δίνεται padded = pad(plaintext + FLAG.encode(), 16) καταλαβαίνουμε ότι το block έχει μέγεθος 16 bytes. Ας δούμε πώς γίνεται το padding:

```
from Crypto.Util.Padding import pad

padded_strings = [pad(b'?'*i, 16) for i in range(1, 17)]
for padded_string in padded_strings:
    print(padded_string)
```

Output:

```
b'?\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f'
b'??\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
b'???\r\r\r\r\r\r\r\r\r\r\r\r\r'
b'????\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c'
b'?????\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b'
b'??????\n\n\n\n\n\n\n\n\n\n'
b'???????\t\t\t\t\t\t\t\t\t'
b'????????\x08\x08\x08\x08\x08\x08\x08\x08'
b'?????????\x07\x07\x07\x07\x07\x07\x07'
b'??????????\x06\x06\x06\x06\x06\x06'
b'???????????\x05\x05\x05\x05\x05'
b'????????????\x04\x04\x04\x04'
b'?????????????\x03\x03\x03'
```

```
b'?????????????\x02\x02'
b'?????????????\x01'
b'??????????????\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10'
```

Ενδιαφέρον παρουσιάζει η χρήση ειδικών χαρακτήρων `\n`, `\r`, `\n`. Αν του δώσουμε 16 bytes, κάνει padding 16 επιπλέον bytes (`0x10` = 16). Αν βάλουμε ένα byte (αφού έχει γίνει μετατροπή σε hex) στο encrypt field παίρνουμε 32 bytes ciphertext. Το ίδιο μέγεθος έχουμε και αν βάλουμε 6 bytes. Ωστόσο με 7 bytes παίρνουμε 16 επιπλέον bytes πράγμα που σημαίνει ότι έχουμε γεμίσει το block size και μας δίνει επιπλέον 16 bytes.

- concatenate pt + flag
- δίνουμε 7 bytes pt
- crypto{flag} = 8 bytes χωρίς το flag
- 16 bytes padding

Συνεπώς απομένει flag x μεγέθους $7 + 8 + x + 16 = 48$. Δηλαδή το flag είναι 17 bytes. Στην συνέχεια κάνουμε brute forcing χαρακτήρες για τα εναπομείναντα 17 bytes. Το κόλπο είναι να γεμίζω με garbage bytes τις 16 θέσεις από τις 17 του flag ώστε να κάνω brute force σε 1 byte του flag. Όταν βρω τον πρώτο χαρακτήρα, μειώνω τα garbage bytes κατά 1 και κάνω brute force στον δεύτερο χαρακτήρα του flag.

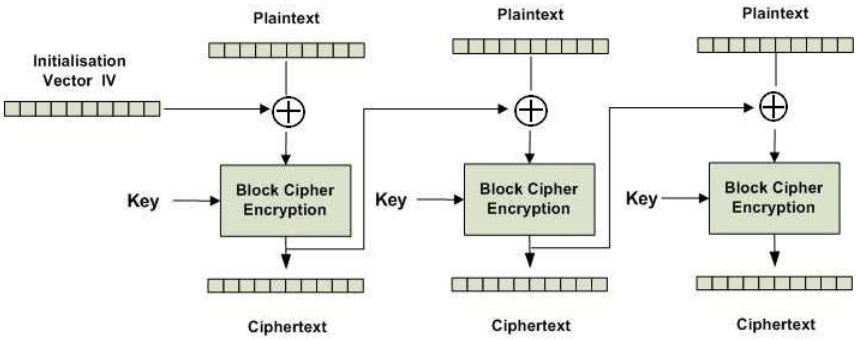
2.9 ECB CBC WTF

Το ECB έχει αδυναμίες όπως το lack of diffusion, block replay, padding issues κλπ. Σε απλά λόγια, είναι δυνατόν να απομακρύνουμε, επαναλάβουμε και να ανταλλάξουμε blocks. Θα κρυπτογραφήσουμε σε CBC και θα αποκρυπτογραφήσουμε σε ECB. Η λύση είναι το chaining. Κάθε cipher block προκύπτει όχι μόνο από το συγκεκριμένο plaintext block αλλά και από τα προηγούμενα plaintext blocks.

$C_i = E_k(P_i \oplus C_{i-1})$
$P_i = C_{i-1} \oplus D_k(C_i)$

Decrypt με ECB επιτρέπει να αποκρυπτογραφήσουμε blocks ανεξάρτητα και δεν χρειαζόμαστε το Initialization Vector.

```
ct = request_ct()
#print (ct)
```



Εικόνα 2.2: CBC Mode

```
#print(len(ct))
pt = request_pt(ct)
#print (pt)
ct_blocks = blockify(ct, 32) # 16 bytes * 2 since hex!
print (ct_blocks)
#['5a8fcbfd3d7b626557d114df8e43cdb4', '218215e2714d47098aed251490761fcb', '98c5a958cee31830f
flag = ''
for c,p in zip(ct_blocks, ct_blocks[1:]):
    current_block_ct = bytes.fromhex(c)
# The zip function will pair these blocks as follows:
#First iteration: c = 'block1', p = 'block2'
# Second iteration: c = 'block2', p = 'block3

    #print (current_block_ct)
    next_block_pt = bytes.fromhex(request_pt(p))
    #print(next_block_pt)
    flag += xor(current_block_ct, next_block_pt).decode()

print(flag)
```

2.10 Flipping Cookie

Να βρούμε ένα $IV_{payload}$ ώστε να καταλήξουμε σε $p_{payload} = admin = True; expiry$

$$c_t = p_t \oplus IV \oplus key \tag{2.1}$$

$$c_t = p_{payload} \oplus IV_{payload} \oplus key \tag{2.2}$$

$$IV_{payload} = p_{payload} \oplus IV \oplus p_t \tag{2.3}$$

```
ct = bytes.fromhex(get_ciphertext())
#print(ct)
iv = ct[:16]
#print(iv)
cookie = ct[16:].hex()
#print(cookie)

pt = b'admin=False;expiry='[:16]
#print (pt)
pt_payload = b'admin=True;expiry='[:16]
iv_payload = xor(pt_payload, pt, iv).hex()
print(check_admin(cookie, iv_payload)['flag'])
```

2.11 Symmetry

Output-Feedback Mode είναι μια μέθοδος για να μετατρέψω ένα block cipher σε stream cipher.

$$C_t = P_t \oplus S_i \quad \text{όπου} \quad S_i = E_k(S_{i-1}) \tag{2.4}$$
$$P_t = C_t \oplus S_i \quad \text{όπου} \quad S_i = E_k(S_{i-1}) \tag{2.5}$$

Παρατηρούμε ότι S_i είναι κοινό στο encryption και decryption και για αυτό λέμε ότι έχουμε συμμετρία.

```
ct = bytes.fromhex(get_ciphertext())
iv = ct[:16].hex()
ct = ct[16:].hex()
print(bytes.fromhex(get_flag(ct, iv)).decode())
```

2.12 Bean Counter

Το πρόβλημα μοιάζει με το προηγούμενο. Ωστόσο εδώ πρέπει να γνωρίζεις ότι τα .png έχουν συγκεκριμένο signature στην αρχή.

$89504e470d0a1a0a$	: 137 80 78 71 13 10 26 10
$0000000d$	: μήκος IHDR (Image Header) chunk
49484452	: ASCII κωδικοί για τους χαρακτήρες IHDR

```
import requests
from pwn import xor

png_signature = bytes.fromhex('89504e470d0a1a0a0000000d49484452')
'''3.1. PNG file signature
The first eight bytes of a PNG file always contain the following (decimal) values:
137 80 78 71 13 10 26 10 '''


def get_ciphertext():
    r = requests.get("https://aes.cryptohack.org/bean_counter/encrypt/")
    return r.json()['encrypted']

ct = bytes.fromhex(get_ciphertext())
key = xor(png_signature, ct[:16])
pt = xor(ct, key)

with open('bean.png', 'wb') as f:
    f.write(pt)
```

Κεφάλαιο 3

Αριθμητική Modular

Score: 395

Σε αυτό το κεφάλαιο θα ασχοληθούμε με τα μαθηματικά που σχετίζονται με την κρυπτογραφία. Πιο συγκεκριμένα θα δουλέψουμε με Αριθμητική Modulo και ισοτιμίες.

3.1 Greatest Common Divisor

Ο Μ.Κ.Δ. gcd(p,q) υπολογίζεται με τον Ευκλίδειο αλγόριθμο. Αν p kai q είναι ακέραιοι, υπάρχει ένας ακέραιος h (ο οποίος λέγεται και μ.κ.δ. των p και q) που είναι διαιρέτης των p και q, και επίσης υπάρχουν ακέραιοι P και Q, τέτοιοι ώστε $Pp - Qq = h$

```
num1 = 66528
num2 = 52920


def euclid(a,b):
    if b > a:
        a,b=b,a

    while b!=0:   #euclidean algorithm
        remainder=a%b

        a=b
        #print (c)
        b=remainder
```

```
        #print(d)

        #print (e)
    return a

result = euclid(num1,num2)
print (result)
```

3.2 Extended GCD

Το αντίστροφο του 4 είναι το $\frac{1}{4}$, γιατί

$$4 * (\frac{1}{4}) = 1. \tag{3.1}$$

$$4x \equiv 1 \pmod 7 \tag{3.2}$$

είναι ισοδύναμο με

$$4x = 7k + 1. \tag{3.3}$$

Ο γενικός τύπος είναι

$$1 = (a * x) mod n \tag{3.4}$$

που μπορεί να γραφεί και ως

$$a^{-1} \equiv x \pmod n \tag{3.5}$$

Γενικά το πρόβλημα έχει μοναδική λύση όταν a,n είναι σχετικά πρώτοι και υπολογίζεται με τον Extended Euclidean Algorithm.
Στο πρόβλημα ζητείται να βρούμε ακέραιους u,v τέτοιους ώστε $a * u + b * v = gcd(a, b)$.

```
    def extended_gcd(a, b):
    if b == 0:
        return a, 1, 0

    gcd, x, y = extended_gcd(b, a % b)
    print (b)
    return gcd, y, x - (a // b) * y
```

Κάναμε χρήση recursive function με base case όταν b=0.

3.3 Modular Arithmetic 1

Στο παρόν πρόβλημα θα ασχοληθούμε με ισοτιμίες.
Θα υπολογίσουμε:

$$11 \equiv x \pmod{6} \tag{3.6}$$

$$8146798528947 \equiv y \pmod{17} \tag{3.7}$$

```
dividend = 11
divisor =6

result = dividend % divisor

dividend1 = 8146798528947
divisor1 = 17

result1 = dividend1 % divisor1

print (result,result1)
print (min(result,result1))
```

3.4 Modular Arithmetic 2

Στο παρόν κεφάλαιο θα ασχοληθούμε με το πεδίο $\mathbb{F}_p$ ενός αριθμού p που είναι πρώτος.

$\forall a \in \mathbb{F}_p$ υπάρχει και ένα αντίστροφο στοιχείο b τέτοιο ώστε $a + b = 0$ και $a \cdot b = 1$.
Το b δεν είναι το ίδιο για πρόσθεση και πολ/μο.
Fermat's Little Theorem: αν m πρώτος αριθμός και a δεν είναι πολλαπλάσιο του m τότε $a^{m-1} \equiv 1 \pmod{m}$

$273246787654^{65536} \equiv 1 \pmod{65537}$. Η απάντηση είναι 1.
$273246787654^{65537} \equiv 1 \pmod{65537}$. Η απάντηση είναι 273246787654.

3.5 Modular Inverting

Καλούμαστε να βρούμε τον αντίστροφο στην πράξη του πολ/μου.

$$3 * d \equiv 1 \pmod{13} \tag{3.8}$$

$$a^{m-1} \equiv 1 \pmod{m}$$
$$a^{m-1} * a^{-1} \equiv a^{-1} \pmod{m}$$
$$\underbrace{a * a * a * \ldots * a}_{m-1 \text{ times}} * a^{-1} \equiv a^{-1} \pmod{m}$$
$$\underbrace{a * a * a * \ldots * a}_{m-2 \text{ times}} * \underbrace{a}_{1\text{time}} * a^{-1} \equiv a^{-1} \pmod{m}$$
$$\underbrace{a * a * a * \ldots * a}_{m-2 \text{ times}} * \underbrace{a * a^{-1}} \equiv a^{-1} \pmod{m}$$
$$\underbrace{a * a * a * \ldots * a}_{m-2 \text{ times}} * \underbrace{1} \equiv a^{-1} \pmod{m}$$
$$\underbrace{a * a * a * \ldots * a}_{m-2 \text{ times}} \equiv a^{-1} \pmod{m}$$
$$a^{m-2} \equiv a^{-1} \pmod{m}.$$
$$====$$

Εικόνα 3.1: From Fermat to Inverse of a

Συνεπώς

$$(3^{-1} \equiv 3^{13-2} \equiv 3^{11} \pmod{13}) \tag{3.9}$$

και αρκεί να υπολογίζουμε:

$$3^{11} \pmod{13} \tag{3.10}$$

```
a=(pow(3,11))
b=a%13
print("b:", b)
```

3.6 Quadratic Residues

Αν p είναι πρώτος και $a \in (0, p)$, τότε a είναι quadratic residue mod p αν:

$$x^2 \equiv a \,(\text{mod}\, p) \text{ για κάποιο } x. \tag{3.11}$$

Με απλά λόγια λέμε ότι το square root του 5 modulo 29 είναι το 11, δηλαδή αν υψώσουμε το 11 στη δύναμη 2 και το κάνουμε modulo 29 θα μας δώσει 5.

```
for a in range(1,29):
    if (a**2)%29 in ints: #list   [14, 6, 11]
        residues.append(a)
print ("Quadratic Residues:", residues)
```

3.7 Legendre Symbol

Προφανώς δεν είναι πάντα εύκολο να κάνουμε iteration για μεγάλο p. Σε αυτό το κεφάλαιο θα μελετήσουμε έναν τρόπο να υπολογίζουμε το quadratic residue. Πριν όμως προχωρήσουμε τρεις ιδιότητες:

Quadratic Residue $\times$ **Quadratic Residue** $=$ **Quadratic Residue**

Quadratic Residue $\times$ **Quadratic Non-residue** $=$ **Quadratic Non-residue**

Quadratic Non-residue $\times$ **Quadratic Non-residue** $=$ **Quadratic Residue**

Το Legendre Symbol είναι ένα κριτήριο να υπολογίσουμε αν a είναι quadratic residue mod p όταν το p είναι πρώτος. Το Legendre symbol:

$$L(a,p) = a^{\frac{p-1}{2}} \ (modp). \tag{3.12}$$

Αν

$$p \equiv 3 \pmod 4 \tag{3.13}$$

p+1 διαιρείται με το 4.

$$(\pm a^{(p+1)/4})^2 \equiv a^{(p+1)/2} \equiv a^{((p-1+2)/2)} \equiv a^{(p-1)/2} \cdot a \pmod p \tag{3.14}$$

$$a^{(p-1)/2} \equiv 1 \pmod p \tag{3.15}$$

$$\pm a^{(p+1)/4} \tag{3.16}$$

```
residuee = []
for a in int_list:
    legendre_symbol = pow(a, (p-1)//2, p)
    if legendre_symbol == 1:
        sqrt = pow(a, (p+1)//4, p)
        residuee.append(sqrt)
```

3.8 Modular Square Root

Εφαρμόζουμε Legendre Symbol να δούμε αν είναι quadratic residue, στην συνέχεια ελέγχουμε αν

$$p \equiv 3 \pmod 4. \tag{3.17}$$

Σε αυτή την περίπτωση εφαρμόζουμε κατευθείαν την φόρμουλα από την προηγούμενη ενότητα. Αλλιώς εφαρμόζουμε τον Tonelli-Shanks algorithm.

```
if pow(a, (p - 1) // 2, p) != 1:
    raise ValueError("No square root exists for the given input.")

if p % 4 == 3:
    return pow(a, (p + 1) // 4, p)

s = p - 1
    e = 0
    while s % 2 == 0:
        s //= 2
        e += 1

    n = 2
    while pow(n, (p - 1) // 2, p) != p - 1:
        n += 1

    x = pow(a, (s + 1) // 2, p)
    b = pow(a, s, p)
    g = pow(n, s, p)

for _ in range(1, e):
        m = 0
        while pow(b, 2 ** m, p) != 1:
            m += 1

        if m == 0:
            return x

        gs = pow(g, 2 ** (e - m - 1), p)
        g = (gs * gs) % p
        x = (x * gs) % p
        b = (b * g) % p
```

3.9 Chinese Remainder Theorem

Βρές τον ακέραιο a τέτοιο ώστε:

$$x \equiv a \mod 935 \tag{3.18}$$

με δεδομένα:

$$x \equiv 2 \mod 5, x \equiv 3 \mod 11, x \equiv 5 \mod 17 \tag{3.19}$$

```
lista= [(2, 5), (3, 11), (5, 17)] #list of tuples
```

Υπολογίζω το γινόμενο των moduli.

```
N = 1
for element in lista:
    N *= lista[1] #iterates through first element of each tuple
```

υπολογίζω Ni και modular inverse:

```
x = 0
for element in lista:
    ai, ni = lista
    Ni = N // ni}

    Mi = pow(Ni, -1, ni)
```

υπολογίζω partial solutions

```
xi = ai * Ni * Mi
x += xi
```

επιστρέφω τον μικρότερο:

```
return x % N
```

3.10 Adrien's Signs

Μελετώντας το source που δίνεται, παρατηρούμε n = pow(a, e, p) που χρησιμοποιούσαμε και παραπάνω. Ενδεχομένως να έχει να κάνει με quadratic residues.

Legendre Symbol:

```
flag = []
for i in ciphertext:
    if pow(i,(p-1)//2,p) == 1:
        flag.append('1')
    else:
        flag.append('0')
```

3.11 Modular Binomials

Δίνονται N, e_1, e_2, c_1, c_2 όπως στο RSA και ζητούνται τα p, q, όπου $p * q = N$. Επιπλέον δίνονται και μια σειρά από εξισώσεις που μπορούμε να αξιοποιήσουμε.

$$N = p * q \tag{3.20}$$

$$c1 = (2p + 3q)^{e_1} \mod N \tag{3.21}$$

$$c2 = (5p + 7q)^{e_2} \mod N \tag{3.22}$$

$$c_1^{e_2} \equiv (a_1 p + b_1 q)^{e_1 e_2} \mod N \tag{3.23}$$

$$c_2^{e_1} \equiv (a_2 p + b_2 q)^{e_1 e_2} \mod N \tag{3.24}$$

Μετά από πράξεις και μια αφαίρεση καταλήγω:

$$a_2^{-e_2 e_1} \cdot (b_2 \cdot q)^{e_2 e_1} + a_1^{-e_1 e_2} \cdot (b_1 \cdot q)^{e_1 e_2} \mod N \tag{3.25}$$

```
q = math.gcd(N, pow(c1, e2, N)*pow(5,e1*e2,N) pow(c2, e1, N)*pow(2,e1*e2,N))
p = N // q
```

Κεφάλαιο 4

Γενικά για Κρυπτογραφία

Score: 110

Το κεφάλαιο αυτό είνα μια εισαγωγή στην κρυπτογραφία. Οι προκλήσεις δεν παρουσιάζουν ιδιαίτερη δυσκολία και σκοπό έχουν να εξοικειώσοουν τους χρήστες με την πλατφόρμα.

4.1 Finding Flags

Το flag είναι της μορφής `crypto{y0ur_f1rst_fl4g}`.

4.2 Great Snakes

Εκτέλεση κώδικα σε γλώσσα python μέσα από το VS Code.

4.3 ASCII

Γνωριμία με ASCII χαρακτήρες

```
for flag in flags:
    print (chr(flag), end="")
```

4.4 Hex

Μετατροπή από hex σε bytes.

```
print (bytes.fromhex())
```

4.5 Base64

Άλλο ένα encoding scheme.

```
import base64
flag = bytes.fromhex('')
print (base64.b64encode(flag))
```

4.6 Bytes and Big Integers

message	HELLO				
ASCII	72	69	76	76	79
HEX	0x48	0x45	0x4c	0x4c	0x4f
base16	0x48454c4c4f				
base10	310400273487				

Πίνακας 4.1: Τα βήματα μετατροπής του μηνύματος σε αριθμούς με τους οποίους κάνουμε πράξεις

```
from Crypto.Util.number import *
print (long_to_bytes(bytes))
```

4.7 XOR Starter

Ζητείται να κάνουμε κάθε χαρακτήρα από το $label \oplus 13$.

```
for char in label:
        xor_result += chr(ord(char) ^ 13)
```

A	B	$A \oplus B$	Output
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Πίνακας 4.2: XOR Table

Βλέπουμε ότι μετατρέπουμε το χαρακτήρα σε ASCII και το κάνουμε xor με το 13. Στη συνέχεια το μετατρέπουμε πάλι σε χαρακτήρα.

4.8 XOR Properties

Property	**Expression**
Commutative	$A \oplus B = B \oplus A$
Associative	$A \oplus (B \oplus C) = (A \oplus B) \oplus C$
Identity	$A \oplus 0 = A$
Self-Inverse	$A \oplus A = 0$

θα εκμεταλλευτούμε κάποιες ιδιότητες της πράξης XOR. Δίνονται τιμές για:
KEY1,
Key1 ^ Key2,
Key2 ^ Key3,
Flag ^ Key1 ^ Key3 ^ Key2

```
key2 = xor_strings(key1, key2_xor_key1)
#we get key2
key3 = xor_strings(key2, key2_xor_key3)
#we get key3
flag = xor_strings(xor_strings(xor_strings(key1, key3), key2), flag_xor_keys)
#we get flag
```

4.9 Favourite byte

Το Key είναι ένα άγνωστο byte, στη συνέχεια κάνουμε Key XOR με το plaintext και τελικά προκύπτει το ciphertext. Εφόσον το flag είναι της μορφής `crypto{y0ur_f1rst_fl4g}`, αν κάνουμε XOR τον χαρακτήρα c με το πρωτο στοιχείο του ciphertext, θα πάρουμε το Key το οποίο μπορούμε να χρησιμοποιήσουμε μετά για να ανακτήσουμε ολόκληρο το plaintext.

```
key = input_str[0] ^ ord('c')
print(key)
print(''.join(chr(c ^ key) for c in input_str))
```

4.10 You either know, XOR you don't

Σε αυτό το πρόβλημα πάλι πρέπει να αξιοποιήσουμε τη μορφή που έχει το flag: `crypto{y0ur_f1rst_fl4g}`.

Συνεπώς θα κάνουμε XOR το ciphertext με το `crypto{ .`

```
xored_bytes = bytes(byte ^ ord(char) for byte, char in zip(hex_bytes[:7], key)
```

Το output μας δίνει myXORke
και μαντεύουμε ότι αναφέρεται στο myXORkey.

Τώρα αρκεί να κάνουμε XOR το myXORkey με το ciphertext για να πάρουμε το flag.

Κεφάλαιο 5

Κρυπτογραφία Δημοσίου Κλειδιού

Score: 125

Σε αυτό το κεφάλαιο θα αντιμετωπίσουμε προβλήματα στην κρυπτογραφία δημοσίου κλειδιού.

5.1 Factoring

Καλούμαστε να παραγοντοποίησουμε σε δυο πρώτους έναν μεγάλο αριθμό. Προτείνεται το primefac-fork αλλά θα χρησιμοποιήσω την sympy και θα καταγράψω τον χρόνο που χρειάστηκε η python.

```
from sympy import factorint
import time

n = 510143758735509025530880200653196460532653147

start_time = time.time()

factors = factorint(n)

end_time = time.time()

total_time = end_time - start_time

print("Prime Factors:", list(factors.keys()))
```

```
print("Total time:", total_time, "seconds")
```

5.2 RSA Starter 1

Σε αυτό το κεφάλαιο μαθαίνουμε για το modular exponentiation που υλοποιείται εύκολα στην python pow(base, exponent, modulus). Μια trapdoor function υπολογίζεται εύκολα προς μια κατεύθυνση αλλά είναι πολύ δύσκολο να την αντιστρέψουμε.

```
print(pow(101,17, 22663))
```

5.3 RSA Starter 2

Μαζί το (N,e) είναι το public Key.

```
x = 12 # message
y = 65537 # exponent
p,q= 17,23 #primes
z = p*q   #mod
print(pow (x,y,z))
```

5.4 RSA Starter 3

$$n = p * q$$

The Euler's totient function $\phi(n)$ υπολογίζει το πλήθος αριθμών ανάμεσα στο 1 και το n-1 που είναι σχετικά πρώτοι με τον n.
$\phi(p \cdot q) = (p-1) \cdot (q-1)$

```
return (p - 1) * (q - 1)
```

5.5 RSA Starter 4

Το d είναι το private Key και είναι το modular multiplicative inverse του e mod totient του n

$d \cdot e \mod \phi(n) = 1$
Με τη βοήθεια του extended Euclidean algorithm που έχουμε υλοποιήσει και σε προηγούμενο κεφάλαιο, θα βρούμε το private key.

```
x, y = extended_gcd(e, phi_N)
```

5.6 RSA Starter 5

Το d το βρήκαμε στο προηγούμενο βήμα. Τώρα καλούμαστε να βρούμε το m. Θυμίζουμε ότι:

$m^e \equiv c \pmod{N}$
$c^d \equiv m \pmod{N}$

```
m = pow(c, d, N)
```

5.7 RSA Starter 6

$C \equiv M^{e_0} \pmod{N_0} \rightarrow$ e_0: public key της άλλης πλευράς.

$S \equiv H(M)^{d_1} \pmod{N_1} \rightarrow$ Hash,d_1: δικό μου private key.

$m \equiv C^{d_0} \pmod{N_0} \rightarrow$ d_0: δικό της private key.

$s \equiv S^{e_1} \pmod{N_1} \rightarrow$ e_1: δικό μου public key.

s : assert $H(m) = s$

Ζητείται να κάνουμε sign το flag με το private key και αξιοποιώντας την hash function SHA256.

```
flag = b"crypto{Immut4ble_m3ssag1ng}"
hash_obj = SHA256.new()
hash_obj.update(flag)
hash_value = hash_obj.digest()

hash_int = bytes_to_long(hash_value)

signature = pow(hash_int, d, N)
```