

Capture the Flag Manual

Alessio Gjergji

Author 2

Author 3

Author 4

Author 5

Indice

1	Crittografia	2
1.1	Conversioni	2
1.1.1	Decodifica da numero in base 2 a big endian	2
1.1.2	Decodifica da numero in base 2 a little endian	2
1.1.3	Decodifica da numero in base 8 a big endian	2
1.1.4	Decodifica da numero in base 8 a little endian	2
1.1.5	Decodifica da numero in base 10 a big endian	2
1.1.6	Decodifica da numero in base 10 a little endian	3
1.1.7	Decodifica da numero in base 16 a big endian	3
1.1.8	Decodifica da numero in base 16 a little endian	3
1.1.9	Decodifica da numero in base 32 a big endian	3
1.1.10	Decodifica da numero in base 32 a little endian	3
1.1.11	Decodifica da numero in base 58 a big endian	3
1.1.12	Decodifica da numero in base 58 a little endian	3
1.1.13	Decodifica da numero in base 64 a big endian	3
1.1.14	Decodifica da numero in base 64 a little endian	4
1.2	One-Time Pad	4
1.3	Crittoanalisi differenziale	4
1.4	Libreria PyCryptodome	4
1.4.1	DES	4
1.4.2	AES	5
1.4.3	Stream cipher	6
1.4.4	Hash	6
1.4.5	RSA	7
1.5	Aritmetica Modulare	7
1.5.1	Proprietà	7
1.5.2	Aritmetica Modulare	8

Capitolo 1

Crittografia

1.1 Conversioni

1.1.1 Decodifica da numero in base 2 a big endian

```
1 result = '101010'  
2 result = int(result, 2).to_bytes((result.bit_length() + 7) // 8, 'big').  
    decode('utf-8')
```

1.1.2 Decodifica da numero in base 2 a little endian

```
1 result = '101010'  
2 result = int(result, 2).to_bytes((result.bit_length() + 7) // 8, 'little').  
    .decode('utf-8')
```

1.1.3 Decodifica da numero in base 8 a big endian

```
1 result = '1234'  
2 result = int(result, 8).to_bytes((result.bit_length() + 7) // 8, 'big').  
    decode('utf-8')
```

1.1.4 Decodifica da numero in base 8 a little endian

```
1 result = '1234'  
2 result = int(result, 8).to_bytes((result.bit_length() + 7) // 8, 'little').  
    .decode('utf-8')
```

1.1.5 Decodifica da numero in base 10 a big endian

```
1 result = 1234  
2 result = result.to_bytes((result.bit_length() + 7) // 8, 'big').decode('utf-8')
```

1.1.6 Decodifica da numero in base 10 a little endian

```
1 result = 1234
2 result = result.to_bytes((result.bit_length() + 7) // 8, 'little').decode('utf-8')
```

1.1.7 Decodifica da numero in base 16 a big endian

```
1 result = '1234'
2 result = bytes.fromhex(result).decode('utf-8')
```

1.1.8 Decodifica da numero in base 16 a little endian

```
1 result = '1234'
2 result = bytes.fromhex(result[::-1]).decode('utf-8')
```

1.1.9 Decodifica da numero in base 32 a big endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b32decode(result).decode('utf-8')
```

1.1.10 Decodifica da numero in base 32 a little endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b32decode(result[::-1]).decode('utf-8')
```

1.1.11 Decodifica da numero in base 58 a big endian

```
1 import base58
2
3 result = '1234'
4 result = base58.b58decode(result).decode('utf-8')
```

1.1.12 Decodifica da numero in base 58 a little endian

```
1 import base58
2
3 result = '1234'
4 result = base58.b58decode(result[::-1]).decode('utf-8')
```

1.1.13 Decodifica da numero in base 64 a big endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b64decode(result).decode('utf-8')
```

1.1.14 Decodifica da numero in base 64 a little endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b64decode(result)[::-1].decode('utf-8')
```

1.2 One-Time Pad

Un dettaglio fondamentale per ottenere sicurezza perfetta è che la chiave sia lunga tanto quanto il messaggio.

Nel caso in cui la chiave sia molto corta (*e ripetuta, per esempio*) potrebbe essere possibile un attacco a forza bruta: provare tutte le chiavi candidate e vedere per quale si ottiene un risultato sensato. La ripetizione di pattern all'interno del messaggio cifrato è un indizio che la chiave sia corta.

```
1 def xor(a, b):
2     return bytes([x^y for x,y in zip(a,b)])
3
4 num = bytes.fromhex('104e137f42')
5
6 for i in range(256):
7     key = i.to_bytes(1, 'big')*len(num)
8     result = xor(num, key)
9     if result.isascii():
10         result = result.decode('ascii')
11         print('key: ', i, 'result: ', result)
```

Nel caso descritto la chiave è di un solo byte, quindi si prova a cifrare il messaggio con tutte le chiavi possibili e si controlla se il risultato è un testo ASCII.

1.3 Crittoanalisi differenziale

Per utilizzare la crittoanalisi differenziale è necessario avere a disposizione un *oracolo* che cifri un messaggio arbitrario con una chiave segreta. Installando il tool MTP possiamo ottenere una parziale implementazione del messaggio.

```
1 $ pip3 install mtp
2 $ mtp <file>
```

1.4 Libreria PyCryptodome

1.4.1 DES

```
1 from Crypto.Cipher import DES
2 from Crypto.Util.Padding import pad
3 from Crypto.Random import get_random_bytes
4 from binascii import hexlify, unhexlify
5
```

```
6     # Chiave in formato esadecimale
7     key_hex = '1bb4545801ca1e93'
8     key = unhexlify(key_hex)
9
10    # Testo in chiaro
11    plaintext = 'La lunghezza di questa frase non e divisibile per 8'
12
13    # Applicare lo schema di riempimento x923
14    plaintext = pad(plaintext.encode('utf-8'), DES.block_size, style='x923')
15
16    # Generare un vettore di inizializzazione casuale
17    iv = get_random_bytes(DES.block_size)
18
19    # stampo Che IV hai utilizzato (in esadecimale)
20    print('IV: ' + hexlify(iv).decode('utf-8') + '\n')
21
22    # Creare un oggetto DES in modalita' CBC
23    cipher = DES.new(key, DES.MODE_CBC, iv)
24
25    # Cifrare il testo in chiaro
26    ciphertext = cipher.encrypt(plaintext)
27
28    # Stampare il testo cifrato in formato esadecimale
29    print("Text: " + hexlify(ciphertext).decode('utf-8') + "\n")
```

1.4.2 AES

```
1     from Crypto.Cipher import AES
2     from Crypto.Util.Padding import pad
3     from Crypto.Random import get_random_bytes
4     from binascii import hexlify, unhexlify
5
6     # Generare una chiave casuale AES256 in formato esadecimale
7     key = get_random_bytes(32) # 32 byte per AES256
8
9     # Testo in chiaro
10    plaintext = 'Mi chiedo cosa significhi il numero nel nome di questo
11               algoritmo.'
12
13    # Applicare lo schema di riempimento PKCS7
14    plaintext = pad(plaintext.encode('utf-8'), AES.block_size)
15
16    # Generare un vettore di inizializzazione casuale
17    iv = get_random_bytes(AES.block_size)
18
19    # Stampo che IV hai utilizzato (in esadecimale)
20    print("IV:", hexlify(iv).decode('utf-8'))
21
22    # Creare un oggetto AES in modalita' CFB con segment size di 24
23    cipher = AES.new(key, AES.MODE_CFB, iv, segment_size=24)
24
25    # Cifrare il testo in chiaro
26    ciphertext = cipher.encrypt(plaintext)
27
28    # Stampare la chiave in formato esadecimale
```

```
28     print("Chiave:", hexlify(key).decode('utf-8'))
29
30     # Stampare il testo cifrato in formato esadecimale
31     print("Testo cifrato:", hexlify(ciphertext).decode('utf-8'))
```

1.4.3 Stream cipher

```
1     from Crypto.Cipher import ChaCha20
2     from binascii import unhexlify
3
4     # Chiave in formato esadecimale
5     key_hex = '9d6a8f...'
6     key = unhexlify(key_hex)
7
8     # Testo cifrato in formato esadecimale
9     ciphertext_hex = 'b7e9ac6...'
10    ciphertext = unhexlify(ciphertext_hex)
11
12    # Nonce in formato esadecimale
13    nonce_hex = '76c24201d...'
14    nonce = unhexlify(nonce_hex)
15
16    # Creare un oggetto ChaCha20
17    cipher = ChaCha20.new(key=key, nonce=nonce)
18
19    # Decifrare il testo cifrato
20    plaintext = cipher.decrypt(ciphertext)
21
22    # Stampa il testo in chiaro
23    print(plaintext.decode('utf-8'))
```

1.4.4 Hash

```
1     from Crypto.Hash import SHA256
2     from binascii import hexlify
3
4     # Testo in chiaro
5     plaintext = 'La lunghezza di questa frase non e divisibile per 8'
6
7     # Creare un oggetto SHA256
8     h = SHA256.new()
9
10    # Aggiornare l'hash con il testo in chiaro
11    h.update(plaintext.encode('utf-8'))
12
13    # Calcolare l'hash
14    digest = h.digest()
15
16    # Stampa l'hash in formato esadecimale
17    print(hexlify(digest).decode('utf-8'))
```

1.4.5 RSA

L'algoritmo funziona con due primi p e q e un esponente pubblico e . L'algoritmo è il seguente:

1. Calcolare $n = p \cdot q$
2. Calcolare $\phi(n) = (p - 1)(q - 1)$
3. Calcolare $d = e^{-1} \pmod{\phi(n)}$
4. La chiave pubblica è (n, e) , la chiave privata è (n, d)

Per cifrare un messaggio m si calcola $c = m^e \pmod{n}$, per decifrarlo si calcola $m = c^d \pmod{n}$.

```

1      p = 36652254321470221037
2      q = 31629674241453983353
3      e = 65537
4      n = p * q
5      phi = (p - 1) * (q - 1)
6      d = pow(e, -1, phi)
7      print("n = ", n)
8
9      print("phi(n) = ", phi)
10
11     print("d = ", d)
12
13     stringa = "Stringa di prova"
14
15     numero = int(''.join(format(ord(char), '08b') for char in stringa), 2)
16
17     cifrato = pow(numero, d, n)
18
19     print("La stringa cifrata: ", cifrato)

```

1.5 Aritmetica Modulare

Diciamo che $a \equiv b \pmod{n}$ se n divide $a - b$.

- $a \equiv b \pmod{n} \iff a \bmod n = b \bmod n$
- $a \equiv b \pmod{n} \iff \exists k \in \mathbb{Z} \mid a = b + kn$
- $a \equiv b \pmod{n} \iff \exists k \in \mathbb{Z} \mid a \bmod n = b + kn$

1.5.1 Proprietà

- $a \equiv a \pmod{n}$
- $a \equiv b \pmod{n} \implies b \equiv a \pmod{n}$
- $a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \implies a \equiv c \pmod{n}$
- $a \equiv b \pmod{n} \wedge c \equiv d \pmod{n} \implies a + c \equiv b + d \pmod{n}$

- $a \equiv b \pmod{n} \wedge c \equiv d \pmod{n} \implies ac \equiv bd \pmod{n}$

1.5.2 Aritmetica Modulare

La *aritmetica modulare* è una branca dell'aritmetica che si occupa delle operazioni sui numeri interi modulari rispetto a un modulo dato. Un concetto fondamentale è il *resto* di una divisione, spesso indicato come $a \bmod m$, che rappresenta il residuo della divisione di a per m .

Un'applicazione comune dell'aritmetica modulare è nel calcolo del *mcd* (massimo comune divisore) di due numeri. La funzione di Euclide è una tecnica classica che sfrutta l'aritmetica modulare per calcolare l'mcd. Ad esempio, se abbiamo due numeri a e b , possiamo utilizzare la funzione di Euclide estesa per trovare i coefficienti di Bezout x e y tali che $a \cdot x + b \cdot y = \text{mcd}(a, b)$.

$$\text{gcd}(a, b) = ax + by \quad (1.1)$$

In Python creiamo una funzione che calcola l'mcd di due numeri utilizzando la funzione di Euclide estesa.

```
1 def gcd(a, b):
2     if a == 0:
3         return b, 0, 1
4     else:
5         g, y, x = gcd(b % a, a)
6         return g, x - (b // a) * y, y
```

Questo concetto è spesso applicato in crittografia, dove la sicurezza si basa su operazioni modulari e il teorema di Fermat.