Lezione S3/L4

Crittografia

Nella lezione pratica di oggi vedremo come decriptare dei messaggi criptati. Ovviamente non possediamo alcun tipo di key è il metodo di criptaggio non ci è noto.
per poter risolvere i problemi possiamo basarci su nozioni logiche usando la ragione tipo:

- il messaggio decriptato deve dare, come soluzione finale, una frase sensata.
- in caso di fallimento di ogni metodo iniziale è possibile una doppia ma anche tripla criptazione.
- alle volte, i file possono essere criptati così tanto che risulta impossibile trovare soluzione.

il primo codice da decriptare:

HSLFRGH

osservando questo codice notiamo che si tratta di un'unica parola composta solo da lettere, oltretutto le lettere si ripetono tra di loro. Vale sempre la pena utilizzare questi dati per trovare un'eventuale soluzione semplice.
Questi dati puntano verso un Caesar cypher , ovvero un metodo di crittografia dove le lettere di una parola sono "shiftate" di una certa posizione in questo caso si può andare su siti come https://www.dcode.fr/caesar-cipher per ottenere tutti i possibili risultati.

Brute-Force mode: the 25 shifts (for the alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ) are tested and sorted from most probable to least probable.

↑↓	↑↓
□3 (□23)	EPICODE
□14 (□12)	TEXRDST
□18 (□8)	PATNZOP
□11 (□15)	WHAUGVW
□5 (□21)	CNGAMBC
□7 (□19)	ALEYKZA
□25 (□1)	ITMGSHI
□19 (□7)	OZSMYNO
□24 (□2)	JUNHTIJ
□13 (□13)	UFYSETU
□4 (□22)	DOHBNCD
□1 (□25)	GRKEQFG
□20 (□6)	NYRLXMN
□15 (□11)	SDWQCRS
□23 (□3)	KVOIUJK
□2 (□24)	FQJDPEF
□17 (□9)	QBUOAPQ
□9 (□17)	YJCWIXY
□10 (□16)	XIBVHWX
□6 (□20)	BMFZLAB
□22 (□4)	LWPJVKL
□16 (□10)	RCVPBQR
□12 (□14)	VGZTFUV
□21 (□5)	MXQKWLM
□8 (□18)	ZKDXJYZ
#25	

figura 1: caesar cypher, brute force.

chiaramente il risultato più sensato è la parola EPICODE.

il secondo codice è :

QWJhIHZ6b2VidHl2bmdyIHB1ciB6ciBhciBucHBiZXRi

come osserviamo questo è immediatamente più complicato, infatti, oltre ad essere un unica parola (ma chiaramente formata da diverse parole) possiede anche dei numeri e lettere maiuscole.
facendo una veloce ricerca appare chiaro che la tipologia di encoding è in base64.
Esistono ovviamente anche altri tipi di encoding simili tipo:

- **Codifica Base32**: Utilizza 32 caratteri (lettere maiuscole da `A-Z` e numeri da `2-7`) per codificare i dati.
- **Codifica Base58**: Elimina i caratteri ambigui (ad esempio, `0`, `O`, `I` e `l`) per facilitare la lettura. Spesso utilizzata negli indirizzi Bitcoin.
- **Codifica Esadecimale (Hexadecimal Encoding)**: Converte i dati in un formato base-16, utilizzando lettere maiuscole (`A-F`) e numeri (`0-9`).
- **UUEncoding (Unix-to-Unix Encoding)**: Codifica i dati binari in caratteri ASCII stampabili utilizzando lettere maiuscole, cifre e simboli.
- **Percent-Encoding**: Utilizzata negli URL, rappresenta i caratteri come `%` seguito da due cifre esadecimali (esempio: `A → %41`).
- **Codifica Quoted-Printable**: Converte i caratteri ASCII non stampabili in una forma stampabile utilizzando il segno `=` seguito da valori esadecimali (esempio: `=20` per lo spazio).
- **Codifiche MIME**: Codificano le email con testo alfanumerico leggibile, utilizzando formati come Base64 o Quoted-Printable.
- **Ascii85 (Base85)**: Codifica i dati in 85 caratteri utilizzando lettere maiuscole, minuscole, numeri e alcuni simboli (`A-Z`, `a-z`, `0-9`, `!#$%&()*+-;<=>?@^_`).

sebbene tutti questi metodi siano buone possibilità alcuni dettagli ci portano ad accettare specificatamente base64.

Lunghezza della Stringa:

- Le stringhe Base64 sono spesso divisibili per 4, poiché Base64 codifica ogni 3 byte di dati in 4 caratteri.
- il codice ha 48 caratteri, un multiplo di 4, il che suggerisce fortemente una codifica Base64.

Set di Caratteri:

- Base64 utilizza solo questi 64 caratteri: `A-Z`, `a-z`, `0-9`, `+`, `/` e `=` per il padding.
- La stringa **QWJhIHZ6b2VidHl2bmdyIHB1ciB6ciBhciBucHBiZXRi** si adatta perfettamente a questo set di caratteri.

Padding (Opzionale):

- Base64 termina spesso con `=` o `==` per rendere la lunghezza divisibile per 4.
- La tua stringa non ha padding, il che è normale in Base64 quando la lunghezza dell'input è un multiplo esatto di 3.

il risultato della decodifica è :

Aba vzoebtyvngr pur zr ar nppbetb

sebbene la frase non abbia senso siamo riusciti ad ottenere delle lettere con spazi e possiamo quindi tornare a pensarlo come un Caesar cypher infatti :

↑↓	↑↓
□13 (□13)	Non imbrogliate che me ne accorgo
□23 (□3)	Ded ycrhewbyqju sxu cu du qssehwe
□7 (□19)	Tut oshxumrogzk ink sk tk giiuxmu
□12 (□14)	Opo jncsphmjbuf dif nf of bddpshp
□19 (□7)	Hih cgvliafcuny wby gy hy uwwilai
□17 (□9)	Jkj eixnkchewpa yda ia ja wyyknck
□9 (□17)	Rsr mqfvskpmexi gli qi ri eggsvks
□6 (□20)	Uvu ptiyvnsphal jol tl ul hjjvynv
□15 (□11)	Lml gkzpmejgyrc afc kc lc yaampem
□22 (□4)	Efe zdsifxczrkv tyv dv ev rttfixf
□25 (□1)	Bcb wapfcuzwohs qvs as bs oqqcfuc
□3 (□23)	Xyx swlbyqvskdo mro wo xo kmmybqy
□10 (□16)	Qrq lpeurjoldwh fkh ph qh dffrujr
□21 (□5)	Fgf aetjgydaslw uzw ew fw suugjyg
□24 (□2)	Cdc xbqgdvaxpit rwt bt ct prrdgvd
□11 (□15)	Pqp kodtqinkcvg ejg og pg ceeqtiq
□1 (□25)	Zaz uyndasxumfq otq yq zq mooadsa
□2 (□24)	Yzy txmczrwtlep nsp xp yp lnnzcrz
□8 (□18)	Sts nrgwtlqnfyj hmj rj sj fhhtwlt
□20 (□6)	Ghg bfukhzebtmx vax fx gx tvvhkzh
□5 (□21)	Vwv qujzwotqibm kpm um vm ikkwzow
□16 (□10)	Klk fjyoldifxqb zeb jb kb xzzlodl
□14 (□12)	Mnm hlaqnfkhzsd bgd ld md zbbnqfn
□4 (□22)	Wxw rvkaxpurjcn lqn vn wn jllxapx
□18 (□8)	Iji dhwmjbgdvoz xcz hz iz vxxjmbj
	#25

figura 2: brute force secondo codice.

BONUS

L'esercizio bonus di oggi ci richiede di utilizzare la Criptazione e Firmatura con OpenSSL e Python. Più specificatamente :

- Generare chiavi RSA. Esercizio Traccia e requisiti
- Estrarre la chiave pubblica da chiave privata.
- Criptare e decriptare messaggi.
- Firmare e verificare messaggi.

utilizzando i comandi elencati nella figura 3 presa dalle slide possiamo installare sia openSSL che la libreria cryptography, per poter generale la chiave privata e la chiave pubblica.

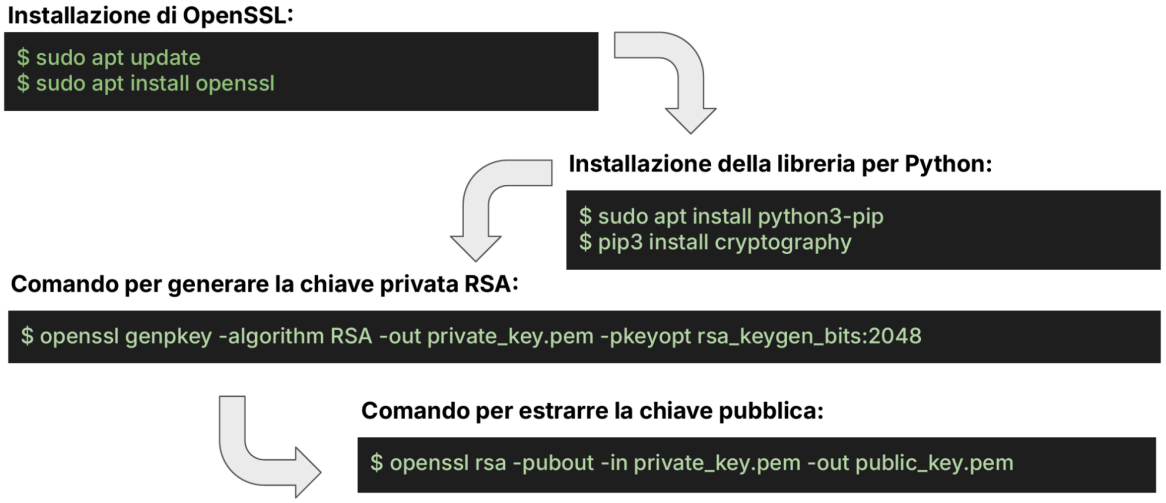

figura 3: istruzioni installazione

possiamo ora procedere con la creazione del file encdec.py come da slide che ci permette di codificare e decodificare codice in base64. seguendo le istruzioni otteniamo il seguente programma:

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization
import base64

with open ('private_key.pem','rb') as key_file:
        private_key = serialization.load_pem_private_key(

                key_file.read(),
                password=None )

with open ('public_key.pem','rb') as key_file:
        public_key = serialization.load_pem_public_key(key_file.read())

message = 'Ciao, mi chiamo Matteo'
encrypted = public_key.encrypt(message.encode(), padding.PKCS1v15())
decrypted = private_key.decrypt( encrypted, padding.PKCS1v15())

print("messaggio originale:", message)
print("messaggio criptato:", base64.b64encode(encrypted).decode('utf-8'))
print("messaggio decriptato:", decrypted.decode('utf-8'))
```

figura 4: encdec.py

per poter utilizzare il programma cambiamo i permessi con chmod e poi utilizziamo creiamo con i comandi in figura tre le 2 chiavi:

figura 5: keygen.

figura 6: encdec.py output

come ultimo step creiamo il codice di firma.py che utilizza il processo di hashing per rendere più sicure le firme digitali.

```python
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
import base64

with open ('private_key.pem','rb') as key_file:
        private_key = serialization.load_pem_private_key(

                key_file.read(),
                password=None )

with open ('public_key.pem','rb') as key_file:
        public_key = serialization.load_pem_public_key(key_file.read())

message = 'Ciao, questo codice è hashato'
signed = private_key.sign(message.encode(), padding.PKCS1v15(), hashes.SHA256())

try :
        encrypted_b64 = base64.b64encode(signed).decode('utf-8')
        public_key.verify(signed, message.encode(), padding.PKCS1v15(), hashes.SHA256())
        print ("Base64 della firma:", encrypted_b64)
        print ("messaggio originale da confrontare: ", message)
        print ("la firma è valida.")
except Exception as e:
        print("la firma non è valida.", str(e))
```

figura 7: firma.py

figura 8: firma.py output