Introducere în criptografie și algoritmi de criptare, encodare sau hashing

Resurse utile pentru incepatori din UNbreakable România

unbreakable.ro

Declinarea responsabilității

Aceste materiale și resurse sunt destinate exclusiv informării și discuțiilor, avand ca obiectiv conștientizarea riscurilor si amenintarilor informatice dar și pregatirea unor noi generatii de specialisti in securitate informatica.

Organizatorii și partenerii UNbreakable România nu oferă nicio garanție de niciun fel cu privire la aceste informații. În niciun caz, organizatorii și partenerii UNbreakable România, sau contractanții, sau subcontractanții săi nu vor fi răspunzători pentru niciun fel de daune, inclusiv, dar fără a se limita la, daune directe, indirecte, speciale sau ulterioare, care rezultă din orice mod ce are legătură cu aceste informații, indiferent dacă se bazează sau nu pe garanție, contract, delict sau altfel, indiferent dacă este sau nu din neglijență și dacă vătămarea a fost sau nu rezultată din rezultatele sau dependența de informații.

Organizatorii UNbreakable România nu aprobă niciun produs sau serviciu comercial, inclusiv subiectele analizei. Orice referire la produse comerciale, procese sau servicii specifice prin marca de servicii, marca comercială, producător sau altfel, nu constituie sau implică aprobarea, recomandarea sau favorizarea acestora de către UNbreakable România.

Organizatorii UNbreakable România recomanda folosirea cunoștințelor și tehnologiilor prezentate în aceste resurse doar în scop educațional sau profesional pe calculatoare, site-uri, servere, servicii sau alte sisteme informatice doar după obținerea acordului explicit în prealabil din partea proprietarilor.

Utilizarea unor tehnici sau unelte prezentate în aceste materiale împotriva unor sisteme informatice, fără acordul proprietarilor, poate fi considerata infractiune in diverse tari.

În România, accesul ilegal la un sistem informatic este considerată infracţiune contra siguranţei şi integrităţii sistemelor şi datelor informatice si poate fi pedepsită conform legii.

Introducere

În fiecare secundă a zilei, informațiile sensibile, de la numerele cardurilor de credit până la dosare de sănătate, secrete militare samd, sunt transmise prin Internet. Datorită criptării, totul poate fi realizat în siguranță. Criptarea permite transferul datelor confidențiale dintr-o rețea în alta fără a fi compromise. **Când datele sunt criptate, acestea nu pot fi accesate și exploatate de către utilizatori neautorizați.**

Ce este criptarea datelor?

Parolele, limitarea accesului la servere si servicii, firewall-urile și stocarea folosită dispozitive externe sunt toate mijloace adecvate de securizare a datelor, dar criptarea este metoda cea mai utilizată. Criptarea convertește mesajele text, e-mailurile și datele încărcările în text cifrat, ceea ce le face ilizibile de către oameni.

Procesele de criptare utilizează algoritmi care convertesc datele în coduri atât de complexe încât cele mai puternice computere ar dura ani de zile pentru a le sparge. Numai o persoană sau un computer care are cheia corectă poate decripta rapid informațiile sau le poate readuce în forma originală. Cheia de decriptare este un alt algoritm care inversează procesul algoritmului de criptare.

De ce este important sa invatam despre criptarea datelor?

Criptarea datelor ajuta oamenii de sute de ani sa transfere informații in mod sigur, fără ca un tert neautorizat să poată înțelege aceste informații.

De-a lungul timpului au apărut doua provocări, ce continua sa stimuleze si astăzi:

- **Crearea unor algoritmi sau tehnici criptografice suficient de puternice** pentru ca datele originale sa nu poată fi recuperate decat cei autorizați sa le citească
- **Identificarea unor vulnerabilitati în algoritmii criptografici** utilizati in comunicare dintre parti

Tipuri de tehnici și algoritmi criptografici

Pe măsură ce metodele de criptografie au evoluat în timp, acestea pot fi separate în două categorii principale: criptografie clasică și modernă.

Criptografia clasica

Spre deosebire de criptografia modernă care lucrează cu date binare, metodele clasice de criptografie au fost dezvoltate folosind doar cifre și litere.

Pe baza acestor informații avem două categorii principale:

- Cifrele monoalfabetice care utilizează o substituție fixă pe întregul mesaj
- Cifrele polialfabetice care folosesc o serie de substituții diferite pe întregul mesaj

Exemplu - cifrul Caesar

Aceasta este una dintre cele mai simple metode de encodare, cunoscut și ca cifru prin substituție sau mutare/transpozitie (cunoscut in engleza ca shifted cipher), deoarece pentru a-l utiliza peste un mesaj, o literă este inlocuita / mutată cu alta bazată pe un număr fix de la 0 la 25 (numărul total de litere în limba engleză alfabet.)

Exemplu practic

Alex și Andreea sunt studenți. Pentru a-și proteja conținutul de e-mailuri de colegii curioși, au decis să adopte cifrul Caesar ca metodă de protecție, folosind „3” ca si cheie de schimbare. Folosind tabelul de mai jos putem vedea că:

```
HELLO = KHOOR
```

Avem această ieșire deoarece folosim 3 ca element de mutare. Asta înseamnă că litera “A”, mutata cu 3 poziții după aceasta, este litera “D”.

Alfabetul simplu

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Alfabetul cifrului Caesar cu mutare 3

Alti algoritmi criptografici clasici

În competitiile de tip Capture the Flag, cum este și UNbreakable România, exista foarte mulți algoritmi criptografici clasici folosit, unii avand abordări netraditionale. Cu titlul, de exemplu lista

de mai jos conține o serie de algoritmi criptografici clasici populari, după numele cunoscut in limba engleza:

- **Simple substitution Cipher** - Este un alt cifru monoalfabet. Principala diferență între Caesar Cipher și acesta este că algoritmul “Simple substitution Cipher” acceptă mai mult decât o singură unitate de mutare. De aici putem extrapola la: litere duble, triplete de litere și așa mai departe.
- **Playfair cipher** - un algoritm ce folosește o matrice de 5x5 pentru a înlocui fiecare litera din alfabet
- **Vigenere cipher -** o varianta imbunatatita a algoritmului Caesar, ce are câteva abordări printre care Vernam cipher sau One-time pad cipher
- **Algoritmi de transpunere**, algoritmi ce reordoneaza literele, fără a fi înlocuite:
    - Rail Fence cipher
    - Scytale cipher
    - Route cipher
    - Columnar Transposition
    - Double Transposition
    - Disrupted Transposition
- **Beaufort cipher -** tehnica ce are la baza Tabula recta, inventat in 1508 de Johannes Trithemius

Criptografie moderna

Foarte diferită de criptografia clasică, criptografia modernă se bazează pe principii matematice, computer science, inginerie samd.

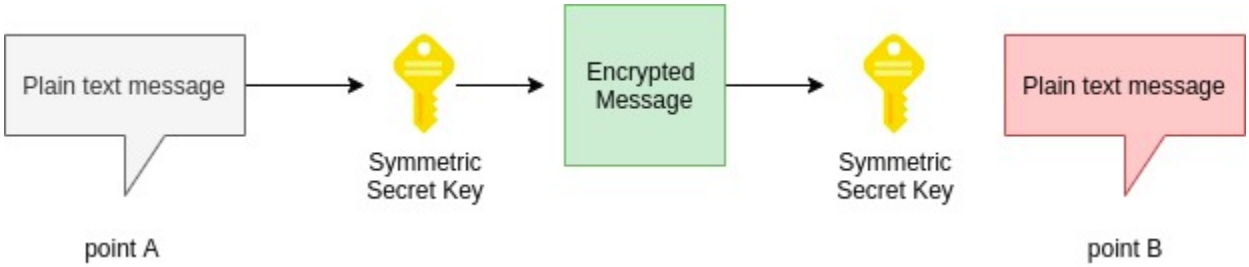
Principalele caracteristici ale criptografiei moderne sunt:

- Funcționează cu date binare
- Unele dintre metodele criptografiei moderne sunt imposibil de spart
- Toate procesele sunt automatizate, nu este necesară interacțiunea manuală, deoarece fiecare metodă de criptare este realizată de un computer

Exemplu - Criptare folosind cheie simetrică

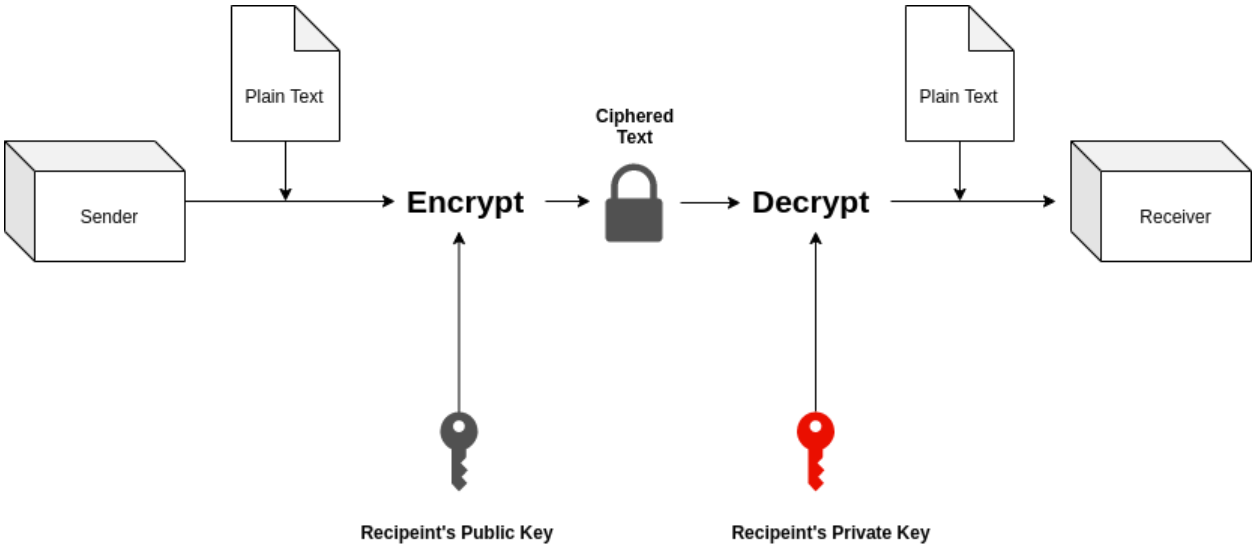
Principalul aspect al acestei metode este că folosește aceeași cheie pentru criptare și decriptare atunci când lucrează cu informații digitale.

Pentru ca două puncte (A & B) să comunice pe un canal criptat, trebuie să partajeze aceeași cheie secretă. Mai multe detalii despre modul de funcționare a criptării cheii simetrice pot fi găsite în imaginea de mai jos:



În competitiile de tip Capture the Flag, cum este și UNbreakable România, sunt prezentate aplicații ce au algoritmi criptografici moderni ce sunt implementate greșit, au vulnerabilitati cunoscute ce permit recuperarea mesajului original într-un termen rezonabil sau sunt reimplementari ale unor algoritmi consacrați, dar care au probleme de securitate. Printre cele mai popular algoritmi criptografici moderni, amintim de:

- **AES (Advanced Encryption Standard)**, cunoscut și sub numele de **Rijndael**, este un algoritm standardizat pentru criptarea simetrică, pe blocuri, folosit astăzi pe scară largă în aplicații și adoptat ca standard de organizația guvernamentală americană NIST.
- **PGP (Public Key Encryption)** - Asymmetric Key Encryption. Criptografia folosind cheie publică sau criptografie asimetrică este un sistem criptografic care folosește perechi de chei: chei publice (care pot fi cunoscute de alții) și chei private (care nu pot fi cunoscute niciodată de nimeni, cu excepția proprietarului). Generarea unor astfel de perechi de chei depinde de algoritmi criptografici care se bazează pe probleme matematice denumite funcții unidirecționale. Securitatea eficientă necesită păstrarea cheii private private; cheia publică poate fi distribuită fără a compromite securitatea datelor.



- **Funcțiile hash (clasă de funcții denumite în lucrări de specialitate și funcții de dispersie sau funcții de rezumat).** În informatică, procesul “hashing” este o practică obișnuită care implică conversia valorilor în alte date, conform algoritmilor de hash aleși. Funcțiile Hash sunt, de asemenea, numite funcții de compresie, deoarece procesul oferă o ieșire cu lungime fixă, chiar și în cazurile în care datele de intrare sunt foarte mari. Alții

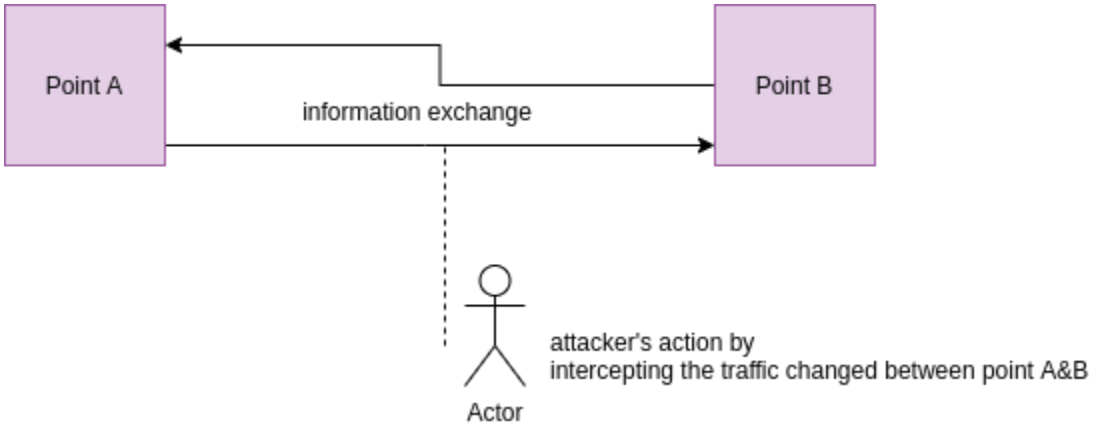
se referă la hash-uri ca funcții digest sau compresie deoarece, în general, reprezintă o valoare mai mică a unei date mai mari. Exemple: MD2, MD4, MD5, MD6, clasa SHA-0, clasa SHA-1, clasa SHA-2 (SHA224, SHA256, SHA384, SHA512) SHA-3, RIPEMD, WHIRLPOOL, CRC32

Tipuri de atacuri sau vulnerabilitati în sistemele criptografice

Exista două categorii mari ce definesc tipurile de atacuri asupra sistemelor criptografice: atacuri pasive și atacuri active.

Atacurile pasive

Scopul atacurilor pasive este de a obține acces neautorizat la date private / secrete fara implicarea activa in comunicatii. Putem numi acțiuni de atac pasiv, cum ar fi interceptarea traficului de rețea, ascultarea (ascultarea conversațiilor și canalelor de comunicare). Atacurile pasive funcționează astfel:

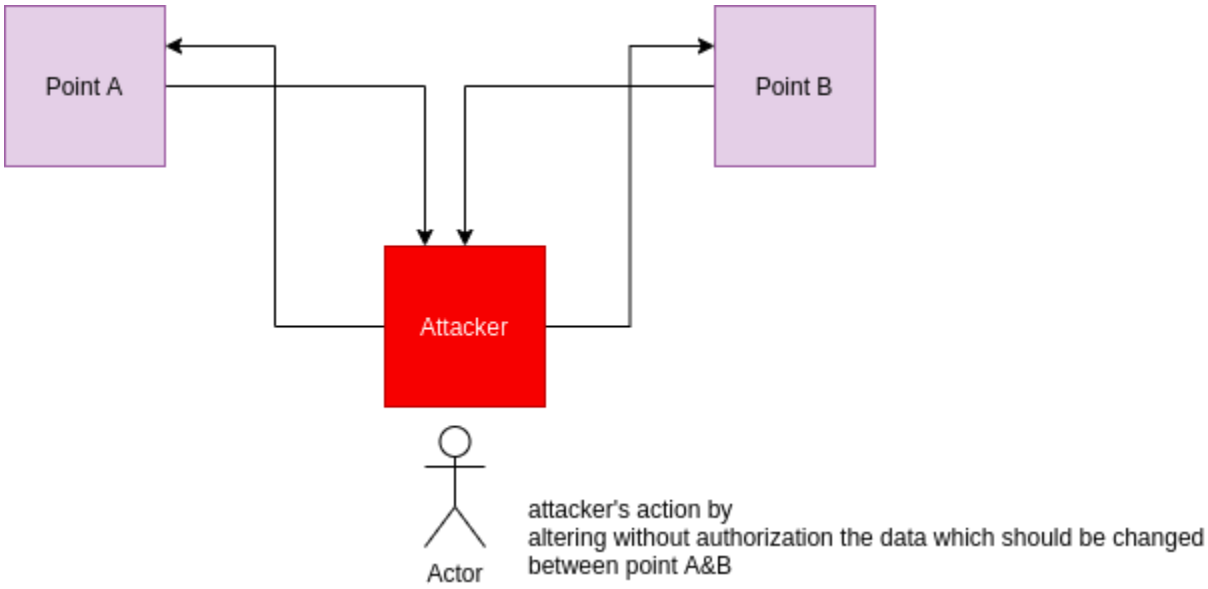


Atacurile active

Spre deosebire de atacurile pasive în care actorii rău intenționați ascultă sau interceptează informațiile, atacurile active implică participarea directă a unui atacator.
În această categorie putem găsi atacuri precum:

- Modificarea datelor s-a schimbat între punctul A&B
- Modificări neautorizate, ștergere, creare de fișiere pe o anumită țintă
- Atacuri DoS & DDoS

Atacurile active funcționează astfel:



Exemple de atacuri asupra algoritmilor criptografici

Fiecare algoritm criptografic poate avea vulnerabilitati din mai multe categorii, dar un bun punct de plecare pentru a va familiariza cu tehnicile folosite pentru a recupera sau intercepta datele criptate sunt acestea:

- Dictionary Attack
- Brute-force
- COA (Ciphered Only Attack)
- KPA (Known Plain Attack)
- CPA (Chosen Plaintext Attack)
- MIM (Man In Middle Attack)
- SCA (Side Channel Attack)
- Frequency Attacks
- Interpolation Attack
- Padding Oracle Attack
- Replay Attacks

Resurse utile

- https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_quick_guide.htm
- https://www.geeksforgeeks.org/playfair-cipher-with-examples/?ref=lbp
- https://secretpy.readthedocs.io/en/latest/
- https://pycryptodome.readthedocs.io/en/latest/src/features.html

Librarii si unelte utile în rezolvarea exercițiilor

- Python Crypto library
- https://cryptii.com/
- https://www.quipqiup.com/
- https://www.alpertron.com.ar/ECM.HTM
- http://factordb.com/

Exerciții și rezolvări

Lost-message (ridicat)

Concurs: UNbreakable #1 (2020)

Descriere:

```
My father was very paranoid with his communication and he encrypted every
message he has ever sent.

This message was for me. Can you help me to decrypt the message ?

enc_message="FNFWCiZJGWWAWZTKYLLKDVNiWCVYViBYHXDiXFBEMiKYEZQMMPKNRiQXZVBQ"

Flag format = ctf{sha256(message)}

Goal: Perform a reverse engineering against the encryption algorithm and
recover the original message.

The challenge was created by Bit Sentinel.
```

Rezolvare:

Pagina exercițiului contine și sursa programului folosit pentru a cripta mesajul:

```
#!/usr/bin/python3
import sys, random, binascii, hashlib, re, math, os
from string import ascii_uppercase as asc
from itertools import product as d


upper = {ascii:chr(ascii) for ascii in range(65,91)}
lower = {ascii:chr(ascii) for ascii in range(97,123)}
digit = {ascii:chr(ascii) for ascii in range(48,58)}

with open("text.txt", "r") as myfile:
```

```
    data = myfile.readline()


def premessage(text):

    text = text.replace("_", "Q")

    return text

def enc4(text, key, debug=False):

    r = [['\n' for i in range(len(text))]
                  for j in range(key)]


    dir_down = False
    row, col = 0, 0

    for i in range(len(text)):


        if (row == 0) or (row == key - 1):
            dir_down = not dir_down


        r[row][col] = text[i]
        col += 1


        if dir_down:
            row += 1
        else:
            row -= 1
```

```
    result = []
    for i in range(key):
        for j in range(len(text)):
            if r[i][j] != '\n':
                result.append(r[i][j])
    return("" . join(result))


def enc3(text, key):
    t=lambda x: x.upper().replace('J','I')
    s=[]
    for _ in t(key+asc):

        if _ not in s and _ in asc:

            s.append(_)

    m=[s[i:i+5] for i in range(0,len(s),5)]
    enc={row[i]+row[j]:row[(i+1)%5]+row[(j+1)%5] for row in m
for i,j in d(range(5),repeat=2) if i!=j}
    enc.update({col[i]+col[j]:col[(i+1)%5]+col[(j+1)%5] for
col in zip(*m) for i,j in d(range(5),repeat=2) if i!=j})
    enc.update({m[i1][j1]+m[i2][j2]:m[i1][j2]+m[i2][j1] for
i1,j1,i2,j2 in d(range(5),repeat=4) if i1!=i2 and j1!=j2})
    l=re.findall(r'(.)(?:(?!\1)(.))?','.join([_ for _ in
t(text) if _ in asc]))

    return ''.join(enc[a+(b if b else 'Z')] for a,b in l)


def enc2(string, key):
    for c in string:
```

```
        o = ord(c)
        if (o not in upper and o not in lower) or o in digit:
            yield o
        else:
            if o in upper and o + key % 26 in upper:
                yield o + key % 26
            elif o in lower and o + key % 26 in lower:
                yield o + key % 26.
            else:
                yield o + key % 26 -26




def enc1(msg, key):
    cipher = ""
    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    col = len(key)
    row = int(math.ceil(msg_len / col))
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('z' * fill_null)
    matrix = [msg_lst[i: i + col]
            for i in range(0, len(msg_lst), col)]


    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                           for row in matrix])
        k_indx += 1
```

```
    return cipher

cipher = enc1(enc3(enc4(premessage(data), 13),"recomanded"),
"zxdfiuypka")
cipher = "".join(map(chr, enc2(cipher, 35)))

print(cipher)
```

Textul inițial, a fost trecut prin patru funcții: enc1, enc2, enc3 și enc4. Pentru a putea recupera mesajul secret, va trebui sa le inversam pe toate.

Prima funcție, enc1, primește un text și o cheie de tip int. Va afisa apoi caracterele textului într-o matrice într-un mod determinabil din cheie si lungimea textului si le va citi pe coloane. Un mod interesant de a inversa aceasta functie e de a apela funcția enc1 cu un text de lungime egala care contine numere de la 1 la l și apoi de a reconstrui textul pe baza output-ului. În alte cuvinte, funcția enc1 creeaza o permutare a textului pe baza lungimii acestuia și a cheii. Folosind cheia de criptare și un text inițial cu caractere unice, putem determina permutarea care e identică pentru toate textele cu lungimi egale. Odată ce permutarea e determinată, aceasta poate fi inversată foarte ușor:

```
def dec1(msg, key):
    hack = [chr(_) for _ in range(len(msg))]
    hack = enc1(hack, key)
    dec = ['' for i in range(len(msg))]
    for i, v in enumerate(hack):
      pos = ord(v)
      dec[pos] = msg[i]
    return ''.join(dec).rstrip('z')
```

Funcția enc2 este o implementare a cifrului Cezar și poate fi inversată foarte ușor folosind proprietățile acestui cifru:

```
def dec2(string, key):
     return enc2(string, 26 - (key % 26))
```

Functia enc4 este asemanatoare din acest punct de vedere: reprezinta o implementare a cifrului Rail fence, ceea ce face posibila aplicarea metodei folosite pentru inversarea funcției enc1:

```
def dec4(text, key, debug=False):
      text = list(text)
      dec = [0 for _ in range(len(text))]
      hack = ''.join([chr(11 + i) for i in range(len(text))]) # \n is 10
and gets ignored
      hack = enc4(hack, key)
      for i, v in enumerate(hack):
            pos = ord(v) - 11
            dec[pos] = text[i]
      return ''.join(dec)
```

Scriptul care decripteaza mesajul secret poate fi găsit mai jos:

```
import sys, random, binascii, hashlib, re, math, os
from string import ascii_uppercase as asc
from itertools import product as d
import itertools, string

enc_message="FNFWCiZJGWWAWZTKYLLKDVNiWCVYViBYHXDiXFBEMiKYEZQMMPKNRiQXZVBQ"

upper = {ascii:chr(ascii) for ascii in range(65,91)}
lower = {ascii:chr(ascii) for ascii in range(97,123)}
digit = {ascii:chr(ascii) for ascii in range(48,58)}

#with open("text.txt", "r") as myfile:
#    data = myfile.readline()


def premessage(text):
    text = text.replace("_", "Q")
    return text

def postmessage(text):
      text = text.replace("Q", "_")
      return text

def dec4(text, key, debug=False):
```

```
      text = list(text)
      dec = [0 for _ in range(len(text))]
      hack = ''.join([chr(11 + i) for i in range(len(text))]) # \n is 10
and gets ignored
      hack = enc4(hack, key)
      for i, v in enumerate(hack):
            pos = ord(v) - 11
            dec[pos] = text[i]
      return ''.join(dec)

def enc4(text, key, debug=False):

    r = [['\n' for i in range(len(text))]
                 for j in range(key)]


    dir_down = False
    row, col = 0, 0

    for i in range(len(text)):


        if (row == 0) or (row == key - 1):
            dir_down = not dir_down


        r[row][col] = text[i]
        col += 1


        if dir_down:
            row += 1
        else:
            row -= 1

    result = []
    for i in range(key):
        for j in range(len(text)):
            if r[i][j] != '\n':
                result.append(r[i][j])
    return("" . join(result))
```

```
double_counter = 0
def dec3(text, key):
    global double_counter
    t=lambda x: x.upper().replace('J','I')
    s=[]
    # asc = ascii_uppercase
    for _ in t(key+asc):
        if _ not in s and _ in asc:
            s.append(_)

    m=[s[i:i+5] for i in range(0,len(s),5)]
    enc={row[i] + row[j] : row[(i+1)%5] + row[(j+1)%5] for row in m for i,j
in itertools.product(range(5),repeat=2) if i!=j}
    enc.update({col[i]+col[j]:col[(i+1)%5]+col[(j+1)%5] for col in zip(*m)
for i,j in itertools.product(range(5),repeat=2) if i!=j})
    enc.update({m[i1][j1]+m[i2][j2]:m[i1][j2]+m[i2][j1] for i1,j1,i2,j2 in
itertools.product(range(5),repeat=4) if i1!=i2 and j1!=j2})

    rev_enc = {}
    for k, v in enc.items():
      rev_enc[v] = k

    l=re.findall(r'(.)(?:(?!\1)(.))?','.join([_ for _ in t(text) if _ in
string.ascii_uppercase]))

    dec = ""
    for a, b in l:
      val = rev_enc[a + b]
      if val[1] == 'Z':
            double_counter += 1
            if double_counter == 1:
                  val = val[0] + val[0]
            else:
                  val = val[0]
      dec += val

    return dec
```

```
def enc3(text, key):
    t=lambda x: x.upper().replace('J','I')
    s=[]
    for _ in t(key+asc):
        if _ not in s and _ in asc:
            s.append(_)

    m=[s[i:i+5] for i in range(0,len(s),5)]
    enc={row[i]+row[j]:row[(i+1)%5]+row[(j+1)%5] for row in m for i,j in
d(range(5),repeat=2) if i!=j}
    enc.update({col[i]+col[j]:col[(i+1)%5]+col[(j+1)%5] for col in zip(*m)
for i,j in d(range(5),repeat=2) if i!=j})
    enc.update({m[i1][j1]+m[i2][j2]:m[i1][j2]+m[i2][j1] for i1,j1,i2,j2 in
d(range(5),repeat=4) if i1!=i2 and j1!=j2})

    l=re.findall(r'(.)(?:(?!\1)(.))?','.join([_ for _ in t(text) if _ in
string.ascii_uppercase]))

    return ''.join(enc[a+(b if b else 'Z')] for a,b in l)


def dec2(string, key):
      return enc2(string, 26 - (key % 26))


def enc2(string, key):
    for c in string:
        o = ord(c)
        if (o not in upper and o not in lower) or o in digit:
            yield o
        else:
            if o in upper and o + key % 26 in upper:
                yield o + key % 26
            elif o in lower and o + key % 26 in lower:
                yield o + key % 26
            else:
                yield o + key % 26 -26



def dec1(msg, key):
```

```
    hack = [chr(_) for _ in range(len(msg))]
    hack = enc1(hack, key)
    dec = ['' for i in range(len(msg))]
    for i, v in enumerate(hack):
      pos = ord(v)
      dec[pos] = msg[i]
    return ''.join(dec).rstrip('z')

def enc1(msg, key):
    cipher = ""
    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    col = len(key)
    row = int(math.ceil(msg_len / col))
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('z' * fill_null)
    matrix = [msg_lst[i: i + col]
            for i in range(0, len(msg_lst), col)]


    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                           for row in matrix])
        k_indx += 1

    return cipher

# cipher = enc1(enc3(enc4(premessage(data), 13),"recomanded"),
"zxdfiuypka")
#cipher = "".join(map(chr, enc2(cipher, 35)))

#print(cipher)


print("Test time!")
test_string = "salut_pe_mine_nu_ma_cheama_george".replace("_", "q").upper()
print(dec4(enc4(test_string, 13), 13) == test_string)
print(dec3(enc3(test_string, "recomanded"), "recomanded")[:-1] ==
```

```
test_string)
test_string_enc2 = "".join(map(chr, enc2(test_string, 35)))
print("".join(map(chr, dec2(test_string_enc2, 35))) == test_string)
print(dec1(enc1(test_string, "zxdfiuypka"), "zxdfiuypka") == test_string)


flag = enc_message
flag = "".join(map(chr, dec2(flag, 35))) # ok
flag = dec1(flag, "zxdfiuypka")
flag = dec3(flag, "recomanded")
flag = dec4(flag, 13) # ok
flag = postmessage(flag) # ok
print(flag)
```

Odată rulat, scriptul va face niște teste care ar trebui sa returneze valoarea **True** și apoi va decripta mesajul secret. Flag-ul este, de fapt, mesajul trecut prin funcția de hashing **md5**:

```
yakuhito@furry-catstation:~/ctf/unbr1/lost-message$ python solve.py
Test time!
True
True
True
True
KEEP_YOUR_COMUNICATION_ENCRYPTED_ALIENS_ARE_WATCHING
yakuhito@furry-catstation:~/ctf/unbr1/lost-message$ echo -n
KEEP_YOUR_COMUNICATION_ENCRYPTED_ALIENS_ARE_WATCHING | sha256sum
f5a2b03dedff103725131a2ce238bdc31b00accba79091237d566561cdfe6ec5  -
yakuhito@furry-catstation:~/ctf/unbr1/lost-message$
```

Rezolvare în engleză: https://blog.kuhi.to/unbreakable-romania-1-writeup#lost-message

War-plan (usor)

Concurs: UNbreakable #2 (2020)

Descriere:

```
There is a hidden message in this file.

Find the message and win the war.

flag = ctf{sha256(message)}
```

Rezolvare:

Fisierul **wav** prezent e pagina exercițiului are 30 de minute si reprezinta un mesaj transmis prin codul morse. Putem folosi [acest site] pentru a recupera mesajul inițial:

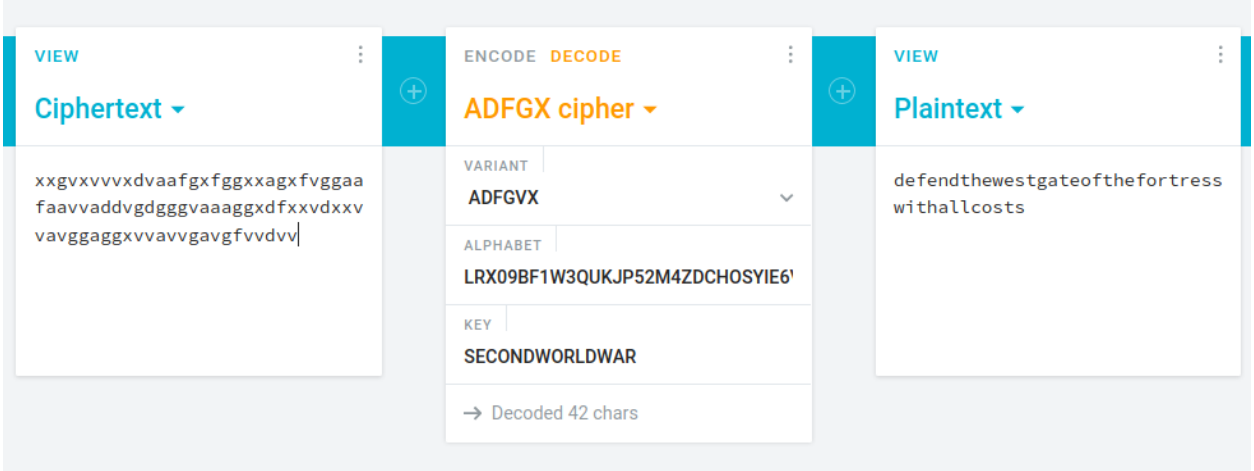
```
THE BATTLE OF THE BULGE, ALSO KNOWN AS THE ARDENNES COUNTEROFFENSIVE, WAS
THE LAST MAJOR GERMAN OFFENSIVE CAMPAIGN ON THE WESTERN FRONT DURING WORLD
WAR II, AND TOOK PLACE FROM 16 DECEMBER 1944 TO 25 JANUARY 1945. IT WAS
LAUNCHED THROUGH THE DENSELY FORESTED ARDENNES REGION OF WALLONIA IN
EASTERN BELGIUM, NORTHEAST FRANCE, AND LUXEMBOURG, TOWARDS THE END OF THE
WAR IN EUROPE. THE OFFENSIVE WAS INTENDED TO STOP ALLIED USE OF THE BELGIAN
PORT OF ANTWERP AND TO SPLIT THE ALLIED LINES, ALLOWING THE GERMANS TO
ENCIRCLE AND DESTROY FOUR ALLIED ARMIES AND FORCE THE WESTERN ALLIES TO
NEGOTIATE A PEACE TREATY IN THE AXIS POWERS\' FAVOR.
XXGVXVVVXDVAAFGXFGGXXAGXFVGGAAFAAVVADDVGDGGGVAAAGGXDFXXVDXXVVAVGGAGGXVVAVVG
AVGFVVDVV BEFORE THE OFFENSIVE THE ALLIES WERE VIRTUALLY BLIND TO GERMAN
TROOP MOVEMENT. DURING THE LIBERATION OF FRANCE, THE EXTENSIVE NETWORK OF
THE FRENCH RESISTANCE HAD PROVIDED VALUABLE INTELLIGENCE ABOUT GERMAN
DISPOSITIONS. ONCE THEY REACHED THE GERMAN BORDER, THIS SOURCE DRIED UP. IN
FRANCE, ORDERS HAD BEEN RELAYED WITHIN THE GERMAN ARMY USING RADIO MESSAGES
ENCIPHERED BY THE ENIGMA MACHINE, AND THESE COULD BE PICKED UP AND
DECRYPTED BY ALLIED CODE-BREAKERS HEADQUARTERED AT BLETCHLEY PARK, TO GIVE
THE INTELLIGENCE KNOWN AS ULTRA. IN GERMANY SUCH ORDERS WERE TYPICALLY
TRANSMITTED USING TELEPHONE AND TELEPRINTER, AND A SPECIAL RADIO SILENCE
ORDER WAS IMPOSED ON ALL MATTERS CONCERNING THE UPCOMING OFFENSIVE. 42 THE
MAJOR CRACKDOWN IN THE WEHRMACHT AFTER THE 20 JULY PLOT TO ASSASSINATE
HITLER RESULTED IN MUCH TIGHTER SECURITY
LRX09BF1W3QUKJP52M4ZDCHOSYIE6VG8NAT7 AND FEWER LEAKS. THE ATTACKS BY THE
SIXTH PANZER ARMY\'S INFANTRY UNITS IN THE NORTH FARED BADLY BECAUSE OF
UNEXPECTEDLY FIERCE RESISTANCE BY THE U.S. 2ND AND 99TH INFANTRY DIVISIONS.
KAMPFGRUPPE PEIPER, AT THE HEAD OF SEPP DIETRICH\'S SIXTH PANZER ARMY, HAD
BEEN DESIGNATED TO TAKE THE LOSHEIM-LOSHEIMERGRABEN ROAD, A KEY2:
SECONDWORLDWAR ROUTE THROUGH THE LOSHEIM GAP, BUT IT WAS CLOSED BY TWO
COLLAPSED OVERPASSES THAT GERMAN ENGINEERS FAILED TO REPAIR DURING THE
```

```
FIRST DAY.
```

Pe parcursul mesajului se pot observa 3 stringuri ‘interesante’:

```
1.
XXGVXVVVXDVAAFGXFGGXXAGXFVGGAAFAAVVADDVGDGGGVAAAGGXDFXXVDXXVVAVGGAGGXVVAVVG
AVGFVVDVV
2. LRX09BF1W3QUKJP52M4ZDCHOSYIE6VG8NAT7
3. KEY2: SECONDWORLDWAR
```

Cum exercițiul este în categoria **crypto**, putem presupune ca este vorba de încă un cifru. Al doilea string pare fi un alfabet sau o permutatie si al treilea cheia, ceea ce înseamnă că primul este **ciphertext**-ul. O particularitate usor observabila este prezenta a doar 6 litere în componenta ciphertext-ului: A, D, F, G, V si X. Dacă căutam aceste litere pe google alături de cuvinte precum ‘crypto’ sau ‘cipher’ descoperim ca cifrul poate fi ADFGVX Cipher. Putem folosi Cryptii pentru a recupera mesajul inițial:



Flag-ul este **ctf{sha256('defendthewes[redactat]sswithallcosts')}**.

Rezolvare în engleză: https://blog.kuhi.to/unbreakable-romania-2-writeup#warplan

Contribuitori

- Mihai Dancaescu (yakuhito)
- Popovici Daniel (betaflash)
- Andrei Avadanei