This is an old revision of the document!

Tema 2

- Deadline: Joi, 7.12.2017 23:55
- Data publicării: 18.11.2017, 02:00
- Data ultimei modificări: 06.12.2017, 14:05
- Responsabili:
  - Vladimir Diaconescu [mailto:vladimir.diaconescu@cs.pub.ro]
  - Alexandru Ivan [mailto:alexnivan@gmail.com]

Enunț

Să se implementeze un program care decodează o serie de șiruri codificate prin diferite metode criptografice.

Structură și detalii de implementare

Tema este formată din mai multe subpuncte, fiecare subpunct constând în decodarea unui șir codificat printr-o metodă specificată. Subpunctele pot fi rezolvate independent, însă puteți refolosi fragmente din rezolvarea unui subpunct în rezolvarea altor subpuncte acolo unde considerați necesar.

1. XOR între două șiruri de octeți - 10p

Multe metode de codificare folosite în criptografie utilizează operația XOR datorită proprietăților matematice ale acesteia. Una dintre metodele cele mai simple folosite în criptografie constă în realizarea operației XOR între fiecare octet din mesaj cu octetul corespondent din cadrul cheii. În mod uzual, pentru această metodă, cheia are aceiași lungime cu mesajul și este folosită în criptarea unui singur mesaj. Această tehnică poartă denumirea de one time pad.

Pentru acest subpunct, este necesară implementarea unei funcții care primește mesajul criptat și cheia folosită la criptare, ambele în reprezentare binară, și decodează mesajul in-place (mesajul decriptat va suprascrie mesajul criptat).

Hint: $(x \oplus k) \oplus k = x$

2. Rolling XOR - 10p

O altă tehnică folosită în criptografie presupune folosirea rezultatului criptării unui bloc de date în criptarea următorului bloc. Mai exact, se efectuează întâi operația XOR între rezultatul respectiv și blocul ce urmează a fi criptat, după care are loc criptarea propriu-zisă folosind o cheie de criptare. Această tehnică se numește cipher block chaining.

Pentru simplitate, în cadrul acestui subpunct, un bloc de date va fi format dintr-un singur octet, iar pasul de criptare va fi omis. Astfel, algoritmul poate fi sumarizat după cum urmează:

- $c_1 = m_1$
- $c_2 = m_2 \oplus c_1$ ($= m_2 \oplus m_1$)
- $c_3 = m_3 \oplus c_2$ ($= m_3 \oplus m_2 \oplus m_1$)
- ș.a.m.d.

Unde c_n reprezintă octetul de pe poziția n din rezultatul operației de criptare, iar m_n reprezintă octetul de pe poziția n din mesajul inițial.

Pentru acest subpunct, se cere implementarea unei funcții care primește un mesaj criptat prin algoritmul descris mai sus și face decodarea in-place (mesajul decriptat va suprascrie mesajul criptat) a acestuia.

3. XOR între două șiruri reprezentate prin caractere hexazecimale - 10p

Acest subpunct presupune realizarea unei operații similare subpunctului 1. Diferența constă în modul de reprezentare a datelor. Atât mesajul criptat cât și cheia care a fost folosită la criptate vor fi reprezentate prin caractere hexazecimale, fiind necesară conversia datelor în formă binară.

Spre exemplu, șirul "deadbeef" va fi convertit în șirul format din octeții 0xde, 0xad, 0xbe, 0xef.

4. Decodificarea unui șir în reprezentare base32 - 20p

base32, la fel ca mai cunoscutul base64, este o metodă de codificare de tip binary to text, însemnând că transformă un sir de date binare într-unul care conține doar caractere tipăribile. Utilitatea acestei codificări nu este neapărat legată de domeniul criptografiei, ci de o gamă mai largă de domenii în care reprezentarea sub formă de caractere ASCII a datelor binare este

necesară.

Codificarea base32 funcționează în baza următorului algoritm: pentru fiecare 5 octeți, se vor genera 8 valori cuprinse între 0 și 31 (inclusiv), conform schemei de mai jos. Valorile generate vor fi folosite ca indecși în alfabetul base32 pentru a determina cele 8 caractere care vor fi folosite pentru codificarea datelor.

```
                    1        2        3      3
 0        8         6        4        2      9
 +--------+--------+--------+--------+--------+
 |< 1 >< 2| >< 3 ><|.4 >< 5.|>< 6 ><.|7 >< 8 >|
 +--------+--------+--------+--------+--------+
             1    1     2     2    3     3   3
 0    5     0    5     0     5    0     5   9
```

Caracterul '|' delimitează octeții din datele de intrare, iar parantezele '< >' delimitează cele 8 valori rezultate: prima de la bitul 0 la 4 inclusiv, a doua de la bitul 5 la 9 inclusiv, ș.a.m.d până la ultima valoare, formată din biții 35 până la 39 inclusiv.

```
   Value Encoding  Value Encoding  Value Encoding  Value Encoding
       0 A             9 J            18 S            27 3
       1 B            10 K            19 T            28 4
       2 C            11 L            20 U            29 5
       3 D            12 M            21 V            30 6
       4 E            13 N            22 W            31 7
       5 F            14 O            23 X
       6 G            15 P            24 Y
       7 H            16 Q            25 Z
       8 I            17 R            26 2
```

În cazul în care dimensiunea datelor de intrare nu este multiplu de 5, se va realiza padding cu caractere '=' după o regulă prestabilită (a se vedea https://tools.ietf.org/html/rfc3548#page-7 [https://tools.ietf.org/html/rfc3548#page-7]).

Pentru acest subpunct, se cere implementarea unei funcții care realizează decodificarea unui șir codificat base32.

5. Bruteforce pe XOR cu cheie de un octet - 10p

În mod uzual în criptografie, dimensiunea cheii de criptare va fi mai mică decât a datelor de intrare. Există mai multe mecanisme care permit folosirea unei chei mai scurte decât mesajul pentru a realiza criptarea. Cel mai simplu dintre acestea reprezintă construirea unei noi chei prin repetarea cheii actuale până se ajunge la dimensiunea necesară. Vulnerabilitatea acestei abordări constă în faptul că este suficientă pentru un potențial atacator obținerea unui substring (din mesajul decriptat) de lungime mai mare sau egală decât cheia intială pentru a putea determina cheia prin brute force.

Pentru a exemplifica, vom folosi o cheie de un octet din care vom obține prin repetare o cheie de dimensiunea mesajului. Criptarea se va face apoi prin XOR între mesaj și cheia rezultată.

Pentru acest subpunct, funcția care trebuie implementată va primi un mesajul criptat și trebuie să returneze cheia folosită în criptarea mesajului, precum și mesajul decriptat in-place.

Hint: mesajul decriptat este în limba engleză și conține "force".

6. Decriptarea unui cifru de substituție - 20p

Cifrul de substituție este o metodă de criptare bazată pe substituția caracterelor din alfabetul în care este scris mesajul cu caractere din alt alfabet. În mod frecvent, alfabetul cu care se înlocuiește conține aceleași caractere ca cel sursă, dar în altă ordine. Spre exemplu, alfabetul sursă poate fi cel al limbii engleze, "abcdefghijklmnopqrstuvwxyz", iar cel destinație "qwertyuiopasdfghjklzxcvbnm". În acest caz, toate aparițiile caracterului 'a' din mesaj vor fi înlocuite cu 'q', 'b' cu 'w' ș.a.m.d.

Vulnerabilitatea acestei metode de criptare constă în faptul că, pentru un text de dimensiune suficient de mare, fiecare literă din alfabetul sursă va apărea cu o anumită frecvență. Astfel, dacă un atacator cunoaște limba mesajului original, poate calcula frecvența de apariție a fiecărei litere din mesajul criptat și poate corela cele două informații pentru a obține alfabetul cu care a fost făcută substituția și, în final, a face operația inversă și a decripta mesajul.

Pentru limba engleză, literele ordonate de la cea mai frecventă la cea mai puțin frecventă sunt: "etaoinshrdlucmfwypvbgkjqxz".

Pentru acest subpunct, se cere implemetarea unei funcții care primește un mesaj criptat prin folosirea unui cifru de substituție și care decriptează in-place mesajul. Funcția va mai primi, de asemenea, adresa unui tabel de substituție ce va trebui completat pe parcurs. "Tabelul" va fi de fapt un vector de octeți de forma: 'a', 'q', 'b', 'p', etc., unde indicii pari (indexarea începând de la 0) reprezintă litera din alfabetul original, iar indicii impari reprezintă litera cu care a fost substituită în mesajul criptat.

Notă: mesajul original, precum și cel criptat, conțin literele a-z (doar lowercase), împreună cu ' ' (spațiu) și '.' (punct). Deși mesajul este în limba engleză, ordinea aparițiilor nu va fi neapărat "etaoin…", însă va fi suficient de aproape pentru a putea deduce pe parcurs din rezultate intermediare.

Notă 2: Spațiu și punct au fost și ele, de asemenea, translatate.

Precizări suplimentare

Toate variabilele, datele, rezultatele parțiale, în afară de cele deja declarate, trebuie stocate pe stivă, nu în secțiunea de date.

Aveți voie (și se recomandă) să implementați oricâte funcții adiționale care să vă ajute în rezolvarea mai multor task-uri, însă structura din main (în sensul de apeluri de funcții) trebuie să rămână neschimbată (excepție făcând rezervarea de spațiu pe stivă pentru variabile auxiliare, și apeluri de funcții auxiliare care să separe sau să determine lungimile șirurilor de intrare).

Scheletul de cod deschide fișierul ”input.dat”, care conține toate string-urile codificate (respectiv, cheile aferente unde este cazul), separate prin '\0' (adică byte-ul cu valoare 0x00). Vă puteți folosi de această informație pentru a vă putea delimita șirurile/intrările.

Pentru claritate, șirurile sunt următoarele (deși se poate deduce și din schelet):

- string1.1 (mesajul criptat) - task1
- string1.2 (cheia de criptare) - task1
- string2 - task2
- string3.1 (mesaj criptat encodat ca hex) - task3
- string3.2 (cheie encodată ca hex) - task3
- string4 (mesaj encodat în base32) - task4
- string5 (mesaj criptat cu cheie pe un byte) - task5
- string6 (mesaj criptat prin tabelă de substituție) - task6

Trimitere și notare

Temele vor trebui încărcate pe platforma vmchecker [https://vmchecker.cs.pub.ro/ui/#IOCLA] (în secțiunea IOCLA) și vor fi testate automat. Arhiva încărcată va fi o arhivă `.zip` care trebuie să conțină:

- fișierul sursă ce conține implementarea temei: `tema2.asm`
- `Makefile`
- `README` ce conține descrierea implementării

Punctajul final acordat pe o temă este compus din:

- punctajul obținut prin testarea automată de pe vmchecker - 80%
- coding style - 10%.

Se va ține cont de:

- claritatea codului
- indentare coerentă
- comentarii
- nume sugestive pentru label-uri
- fișier README - 10%

Temele care nu trec de procesul de asamblare (build) nu vor fi luate în considerare.

Mașina virtuală folosită pentru testarea temelor de casă pe vmchecker este descrisă în secțiunea Mașini virtuale din pagina de resurse.

Resurse

Arhiva ce conține fișierele de la care puteți începe implementarea este aici.