Laborator de criptografie – Cifrarea cu cheie secretă

1 Scopul lucrării

Scopul lucrării este familiarizarea studenților în privința conceptelor criptografiei cu cheie secretă. La finalizarea lucrării, studenții vor avea o experiență nemijlocită cu algoritmii de cifrare, modurile de cifrare, completări pentru textul în clar[1] și vectorul de inițializare (engl. initial vector, IV). În plus, studenții vor fi capabili să folosească unelte și să scrie programe pentru cifrarea/descifrarea mesajelor. Acest laborator acoperă următoarele subiecte:

- Cifrarea cu cheie secretă
- Cifruri de substituție și analiza frecvenței
- Moduri de cifrare și completări pentru textul în clar (engl. paddings)
- Programarea folosind biblioteca criptografică

2 Desfășurarea lucrării

3 Sarcini de efectuat

3.1 Sarcina 1: Analiza frecvenței împotriva unui cifru de substituție monoalfabetic

Se știe foarte bine că un cifru de substituție monoalfabetic (cunoscut și ca cifru monoalfabetic) nu este sigur, deoarece poate fi supus analizei frecvenței. În această lucrare vi se dă un text cifrat cu un cifru monoalfabetic; mai precis, fiecare literă din textul în clar este înlocuită de o alta, iar înlocuirea nu variază (adică, o literă este totdeauna înlocuită de aceeași literă în cifrare). Sarcina noastră este să aflăm textul original folosind analiza frecvenței.

În cele ce urmează vom descrie cum am cifrat articolul original și ce simplificări am făcut. Instructorii pot alege un articol aleator, în loc să-l folosească pe cel dat.

- Pasul 1: să facem câteva simplificări în articolul original. Convertim toate literele la minuscule și eliminăm toate semnele de punctuație și numerele. Păstrăm spațiile dintre cuvinte, astfel încât putem observa limitele cuvintelor din textul cifrat. La cifrarea reală folosind un cifru monoalfabetic, spațiile ar fi eliminate. Păstrăm spațiile pentru a simplifica sarcina. Am făcut acest lucru folosind următoarele comenzi:

```
$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
$ tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
```

- Pasul 2: generăm cheia de criptare, adică tabela de substituție. Vom permuta alfabetul de la a la z folosind Python și vom folosi alfabetul permutat pe post de cheie. Observați următorul program.

[1] engl. paddings

```
$ python
>>> import random
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> list = random.sample(s, len(s))
>>> ''.join(list)
'sxtrwinqbedpvgkfmalhyuojzc'
```

- Pasul 3: folosim comanda `tr` pentru a cifra. Cifrăm doar litere, lăsând caracterele de tip "whitespace" neschimbate.

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' \
< plaintext.txt > ciphertext.txt
```

Am creat text cifrat folosind o cheie de cifrare diferită (nu cea de mai sus). Sarcina noastră acum este să folosim analiza frecvenței literelor pentru a ne da seama care este cheia de criptare și textul în clar original.

Ghid. Folosind analiza frecvenței, putem determina textul în clar pentru câteva caractere destul de ușor. Acele caractere le putem înlocui cu corespondentele ghicite de text clar, pentru a afla mai multe indicii. E mai bine să folosim litere mari pentru textul în clar, astfel încât să ne putem da seama care este textul cifrat și care este descifrat. Putem folosi comanda `tr` pentru a face acest lucru. Spre exemplu, în cele ce urmează, putem înlocui literele `a`, `e`, și `t` din `in.txt` cu literele X, și, respectiv, E; rezultatele sunt salvate în `out.txt`.

```
$ tr 'aet' 'XGE' < in.txt > out.txt
```

Există multe resurse online pe care le putem folosi. Dăm patru dintre acestea:

- `https://www.dcode.fr/frequency-analysis` și `https://www.boxentriq.com/code-breaking/frequency-analysis`: acest situri de web pot produce statistici din textul cifrat, inclusiv frecvențele literelor luate individual, frecvențele bigramelor (secvențe de 2 litere) și ale trigramelor (secvențe de 3 litere), ș.a.m.d.
- `https://en.wikipedia.org/wiki/Frequency_analysis`: această pagină a Wikipedia oferă frecvențele pentru un text englezesc în clar, tipic.
- `https://en.wikipedia.org/wiki/Bigram`: frecvențele bigramelor.
- `https://en.wikipedia.org/wiki/Trigram`: frecvențele trigramelor.

3.2 Sarcina 2: Criptarea cu diferite cifruri și în diverse moduri

În cadrul acestei sarcini vor încerca diverși algoritmi de cifrare, în diferite moduri. Se poate folosi comanda `openssl enc` pentru a cripta/decripta un fișier. Pentru a citi manualele folosiți `man openssl` și `man enc`.

```
% openssl enc ciphertype -e  -in plain.txt -out cipher.bin \
             -K  00112233445566778889aabbccddeeff \
             -iv 0102030405060708
```

Înlocuiți `ciphertype` cu un anumit tip de cifru, cum sunt `-aes-128-cbc`, `-aes-128-cfb`, `-bf-cbc` etc. Încercați cel puțin trei tipuri de cifruri diferite fiecare în trei moduri diferite. Semnificația opțiunilor din linia de comandă și toate cifrurile suportate se pot afla folosind comanda `"man enc"`. Mai jos sunt prezentate câteva opțiuni uzuale ale comenzii `openssl enc`:

```
-in <file>     fișier de intrare
-out <file>    fișier de ieșire
-e             criptează
-d             decriptează
-K/-iv         urmează cheia/iv în hex
-[pP]          tipărește iv/cheia (apoi ieși dacă argumentul este -P)
```

3.3 Sarcina 3: Modul de criptare – ECB vs. CBC

Fișierul `pic_original.bmp` conține o imagine simplă. Dorim să criptăm această imagine, astfel încât cei care nu au cheile de cifrare să nu poată ști ce conține imaginea. Cifrați fișierul folosind modurile ECB (Electronic Code Book) și CBC (Cipher Block Chaining) și apoi executați următoarele:

1. Tratați imaginea cifrată ca imagine și folosiți software de vizualizare pentru a o afișa. Deoarece, în cazul fișierelor `.bmp`, primii 54 octeți conțin informație de header despre imagine, trebuie să îi setăm corespunzător, astfel ca fișierul să poată fi tratat ca fișier `.bmp` legitim. Pentru aceasta putem folosi editorul hex `bless` pentru a modifica direct fișiere binare. Putem folosi și comenzile următoare pentru a obține antetul din `p1.bmp`, datele din `p2.bmp` (de la deplasamentul 55 până la sfârșitul fișierului) și apoi combina antetul și datele într-un fișier nou.

   ```
   $ head -c 54 p1.bmp > header
   $ tail -c +55 p2.bmp > body
   $ cat header body > new.bmp
   ```

2. Afișați imaginea cifrată folosind orice program de vizualizare ()am instalat programul de vizualizare numit `eog` pe mașina virtuală).

3. Puteți deriva orice informație utilă referitoare la original din figura cifrată? Justificați-vă observațiile.

   Alegeți o imagine după dorință, repetați experimentul de mai sus și raportați ce ați observat.

3.4 Sarcina 4 : Caractere de completare pentru textul în clar

Pentru cifrurile bloc, atunci când dimensiunea textului în clar nu este multiplu al dimensiunii blocului cifrat, pot fi necesare caractere de completare. În mod normal toate cifrurile bloc folosesc completarea PKCS#5, cunoscută ca și completare bloc standard (standard block padding). Vom efectua următoarele experimente pentru a înțelege cum funcționează completarea:

1. Folosim modurile ECB, CBC, CFB și OFB pentru a cifra fișierul (cu orice cifru). *Raportați ce moduri au completări și care nu au. Pentru cele care nu au nevoie de completări explicați de ce.*

2. Creăm trei fișiere, care să conțină 5, 10 și, respectiv, 16 octeți. Putem folosi comanda "`echo -n`" pentru a crea astfel de fișiere. Exemplul următor arată cum se creează un fișier numit `f1.txt` de lungime 5 (fără opțiunea `-n` lungimea ar fi 6 deoarece `echo` adaugă un caracter de linie nouă (`newline`):

```
$ echo -n "12345" > f1.txt
```

Apoi folosim comanda "`openssl enc -aes-128-cbc -e`" pentru a cifra aceste trei fișiere folosind AES pe 128-biți în modul CBC. *Precizați lungimea fișierelor cifrate*. Am dori să vedem ce se adaugă la completare în timpul cifrării. Pentru a realiza acest scop, vom descifra aceste fișiere folosind comanda "`openssl enc -aes-128-cbc -d`". Din păcate, descifrarea va înlătura automat orice completare, astfel că nu putem vedea umplerea. Comanda are și o opțiune numită "`-nopad`", care dezactivează umplerea, adică, la descifrare, caracterele de completare nu vor fi înlăturate. De aceea, examinând datele descifrate, putem afla ce s-a folosit la completare. Vă rugăm să folosiți aceasta tehnică pentru a afla umplerile pentru cele trei fișiere. Trebuie remarcat că datele de completare pot să nu fie tipăribile, așa că poate fi nevoie să folosiți o unealtă de afișare în hexazecimal pentru a afișa conținutul. În exemplul următor se arată cum se poate afișa un fișier în format hexazecimal:

```
$ hexdump -C p1.txt
00000000 31 32 33 34 35 36 37 38 39 49 4a 4b 4c 0a |123456789IJKL.|
$ xxd p1.txt
00000000: 3132 3334 3536 3738 3949 4a4b 4c0a 123456789IJKL.
```

3.5 Sarcina 5: Propagarea erorilor – Text cifrat alterat

Pentru a înțelege proprietățile diverselor moduri de cifrare, efectuați următorul exercițiu:

1. Creați un fișier text de lungime cel puțin 1000 octeți.
2. Cifrați fișierul folosind cifrul AES-128.
3. Din nefericire, un bit din cel de-al 55-lea octet din fișierul cifrat fost alterat. Simulați alterarea folosind editorul hex `bless`.
4. Descifrați fișierul alterat (fișierul cifrat) folosind cheia și IV[2] corecte.

Răspundeți la următoarele întrebări:

1. Câtă informație puteți recupera din descifrarea fișierului alterat, dacă modul de cifrarea a fost ECB, CBC, CFB, respectiv OFB? Dați un răspuns înainte de efectuarea acestei sarcini și apoi aflați dacă răspunsul dat a fost corect sau greșit după execuția acestei sarcini.
2. Explicați de ce.

3.5.1 Sarcina 6. Vectorul inițial (IV) și erori comune

Majoritatea modurilor de cifrare necesită un vector inițial (IV). Proprietățile unui IV depind de schema criptografică utilizată. Dacă nu alegem cu grijă vectorii inițiali, datele cifrate de noi pot fi nesigure, chiar dacă folosim un algoritm și un mod sigur. Obiectivul este să înțelegeți problemele care apar la alegerea necorespunzătoare a IV. Efectuați următoarele experimente:

[2] vectorul de inițializare, engl. Initialization Vector

- **Sarcina 6.1.Unicitatea IV** O cerință de bază pentru IV este *unicitatea*, ceea ce înseamnă că nu poate fi refolosit cu aceeași cheie. Pentru a înțelege de ce, cifrați același text folosind (1) doi IV diferiți și (2) același IV. *Vă rugăm să vă descrieți observațiile și, pe baza lor, să explicați de ce IV trebuie să fie unici*.

- **Sarcina 6.2. Eroare comună: folosirea aceluiași IV.** Cineva poate spune că, dacă textul în clar nu se repetă, folosirea aceluiași IV este sigură. Să examinăm modul Output Feedback (OFB). Să presupunem că atacatorul obține textul în clar (`P1`) și textul cifrat (`C1`). Poate el/ea descifra alte mesaje cifrate dacă IV este tot timpul același? Vi se oferă următoarea informație și vă rugăm să încercați să determinați conținutul real al `P2` pe baza lui `C2`, `P1` și `C1`.

  ```
  Plaintext (P1): This is a known message!
  Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
  Plaintext (P2): (unknown to you)
  Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
  ```

  Dacă înlocuim în acest experiment OFB cu CFB (Cipher Feedback), cât din `P2` poate fi revelat? Doar trebuie să răspundeți la întrebare, fără să demonstrați.

  Atacul folosit în acest experiment se numește `atac cu text în clar cunoscut`, care reprezintă un atac model pentru criptanaliști (situație în care atacatorul are acces la textul în clar și corespondentul său cifrat). Dacă aceasta poate duce la revelarea de alte informații secrete, schema de cifrare nu este considerată ca fiind sigură.

- **Sarcina 6.3. Eroare comună: folosirea unui IV predictibil**. Din sarcinile anterioare, știm că IV nu se pot repeta. O altă cerință importantă pentru UV este impredictibilitatea pentru multe dintre schemele de criptare, adică IV trebuie să fie generați aleator. În cadrul acestei sarcini vom vedea ce se întâmplă dacă IV sunt predictibili.

  Să presupunem că Bob a trimis un mesaj cifrat, iar Eve știe că conținutul mesajului este fie `Yes` fie `No`. Eve poate vedea textul cifrat și IV folosit la cifrarea mesajului, dar cum algoritmul de criptare AES este destul de tare, Eve nu are idee ce conține de fapt. Cu toate acestea, cum Bob folosește IV-uri predictibile, Eve știe exact ce IV va folosi Bob pentru următorul mesaj. Cele ce urmează rezumă ce știu Bob și Eve:

  ```
  Metoda de criptare: 128-bit AES in modul CBC.
  Cheia (in hex): 00112233445566778899aabbccddeeff (știut doar de Bob)
  textul cifrat (C1): bef65565572ccee2a9f9553154ed9498 (știut de amândoi)
  IV folosit pentru P1 (știut de amândoi)
  (în ascii): 1234567890123456
  (în hex) : 31323334353637383930313233343536
  Următorul IV (știut de amândoi)
  (în ascii): 1234567890123457
  (în hex) : 31323334353637383930313233343537
  ```

Un cifru bun nu trebuie doar să tolereze atacul cu text clar cunoscut descris anterior, dar și să tolereze un *atac cu text în clar ales*, care este un model de atac pentru criptanaliză în care atacatorul poate obține textul cifrat pentru un text în clar arbitrar. Cum AES este un cifru tare care poate tolera atacul cu text în clar ales, Bob își poate permite să cifreze orice text în clar pe care i-l dă Eve; el folosește un IV diferit pentru fiecare text în clar, dar, din păcate, IV pe care le generează nu sunt aleatoare ci predictibile.

Sarcina voastră este să construiți un mesaj `P2` și să cereți lui Bob să-l cifreze și să vă dea textul cifrat. Obiectivul vostru este să folosiți această ocazie pentru a determina dacă conținutul real al lui `P1` este `Yes` sau `No`.

Există criptanaliză mai avansată asupra IV, care este în afara scopului acestei lucrări. Puteți citi articolul postat la `https://defuse.ca/cbcmodeiv.htm`. Deoarece cerințele pentru IV depind de schemele criptografice, e greu de ținut minte ce proprietăți trebuie păstrate la alegerea unui IV. Totuși, vom fi în siguranță dacă folosim un IV nou la fiecare cifrare, iar IV nou trebuie generat folosind un generator de numere pseudoaleatoare bun, care să fie impredictibile pentru adversari.

3.6 Sarcina 7: Programarea folosind biblioteca criptografică

Până acum am învățat cum să folosim uneltele furnizate de `openssl` pentru criptarea și decriptarea mesajelor. În cadrul acestei sarcini vom învăța cum să folosim biblioteca criptografică a `openssl` la cifrarea/descifrarea mesajelor în programe.

OpenSSL furnizează o API numită EVP, care constituie o interfață de nivel înalt pentru funcții criptografice. Deși OpenSSL are și interfețe directe cu fiecare algoritm de cifrare, biblioteca EVP furnizează o interfață comună pentru diferiții algoritmi de cifrare. Pentru a cere EVP să folosească un anume algoritm, pur și simple trebuie să transmitem alegerea făcută interfeței EVP. Cod exemplu se găsește la `https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption`. Familiarizați-vă cu acest program și apoi efectuați exercițiul următor.

Se dă un text în clar și unul cifrat. Știți următoarele:

- S-a folosit `aes-128-cbc` pentru a genera textul cifrat din textul clar
- Un alt indiciu este că cheia folosită la cifrare este un cuvânt englezesc mai scurt de 16 caractere; cuvântul poate fi găsit într-un dicționar englezesc uzual. Cum cuvântul este mai scurt de 16 caractere (adică 128 biți), se adaugă caractere #, cu valoarea hexazecimală 0x23, la sfârșitul cuvântului pentru a forma o cheie de 128 biți.

Ținta dvs. este să scrieți un program care să determine această cheie. Puteți descărca o listă de cuvinte englezești de pe Internet. Există și o legătură spre o asemenea listă la:
`http://www.cis.syr.edu/~wedu/seed/Labs/Crypto/Crypto_Encryption/files/words.txt`
Textul în clar, textul cifrat și IV sunt:

```
Textul în clar (total 21 caractere): This is a top secret.
Textul cifrat (format hexazecimal):  764aa26b55a4da654df6b19e4bce00f4
                                     ed05e09346fb0e762583cb7da2ac93a2
IV (format hexazecimal): aabbccddeeff00998877665544332211
```

Trebuie să țineți seama de următoarele observații:

Observația 1: Dacă optați să stocați textul în clar într-un fișier și să-l folosiți ca intrare în program, atunci trebuie să vă asigurați că lungimea fișierului este 21. Unele editoare pot adăuga un caracter special la sfârșitul fișierului. Dacă e așa, atunci puteți folosi unealta `bless` pentru a elimina caracterul special. Cea mai ușoară modalitate de a stoca mesajul într-un fișier este prin folosirea comenzii `echo`:

```
$ echo -n "This is a top secret." > file
```

Observația 2: Pentru această sarcină trebuie să scrieți propriul program pentru a utiliza biblioteca criptografică. Sarcina nu va fi considerata ca realizată dacă o efectuați sarcina folosind comenzile `openssl`. Cod exemplu se poate găsi la `https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption`

Observația 3: La compilarea codului cu `gcc`, nu uitați să folosiți opțiunea `-lcrypto`, deoarece codul dvs. are nevoie de biblioteca criptografică. Un exemplu:

```
$ gcc -o myenc myenc.c -lcrypto
```

Notă pentru instructori. Se pot utiliza următoarele comenzi pentru a genera text în clar și cifrat folosind o cheie diferită (cuvântul `example` trebuie înlocuit cu un alt cuvânt secret și trebuie adăugate semne # pentru a face lungimea șirului să fie 16):

```
$ echo -n "This is a top secret." > plaintext.txt
$ echo -n "example#########" > key
$ xxd -p key
6578616d706c65232323232323232323
$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext.bin \
-K 6578616d706c65232323232323232323 \
-iv 0102030405060708090a0b0c0d0e0f \
$ xxd -p ciphertext.bin
e5accdb667e8e569b1b34f423508c15422631198454e104ceb658f5918800c22
```

4 Trimiterea rezultatelor

Trebuie să trimiteți un raport detaliat în care să descrieți ce ați făcut și ce ați observat; de asemenea trebuie să explicați observațiile surprinzătoare sau interesante. În cadrul raportului trebuie să răspundeți la toate întrebările puse în această lucrare de laborator.