Pràctica 2 - Criptografia

Presentació

Aquesta segona pràctica consta de dos exercicis. En el primer es treballa un criptosistema de flux, l'*Shrinking generator* i el segon exercici implementa una part del criptosistema IDEA.

Descripció de la PAC/pràctica a realitzar

Exercici 1 (5 punts)

En aquest primer exercici implementarem una funció que permeti obtenir un generador pseudoaleatori *Shrinking* que utilitzarem per xifrar i desxifrar missatges.

L'*Shrinking generator* és un generador pseudoaleatori format per dos LFSRs. Un primer LFSR, que denotarem LFSR1, s'utilitza per controlar la sortida del segon LFSR, l'LFSR2. A cada impuls de rellotge es mouen els dos LFSR. Si l'LFSR1 dóna com a sortida un 1, aleshores la sortida del generador és directament el bit de sortida de l'LFSR2. Ara bé, si l'LFSR1 té com a sortida un 0, aleshores la sortida de l'LFSR2 es descarta, es tornen a moure els dos LFSRs i es repeteix l'operació. En podeu trobar una descripció al capítol 6 del "Handbook of Applied Cryptography", concretament a la pàgina 211.

Per simplificar la implementació, utilitzarem la funció del paquet matemàtic SAGE que implementa un LFSR:

```
sage.crypto.lfsr.lfsr_sequence(key, fill, n)
```

Per a desenvolupar el generador cal que implementeu les següents funcions que us permetran validar la correcció de la vostra implementació amb el joc de proves que us proporcionem:

1. Funció que implementa un LFSR. **(1 punt)**

   La funció prendrà com a variables d'entrada tres valors: `polinomi`, `estat_inicial` i `bits_de_sortida`; i retornarà la seqüència de sortida.

   - A la variable `polinomi` s'escriurà el polinomi de connexions de l'LFSR;
   - La variable `estat_inicial` contindrà l'estat inicial de l'LFSR;
   - La variable `bits_de_sortida` contindrà el nombre de bits de la seqüència de sortida.
   - La funció retornarà la seqüència de sortida.
     - Exemple:

- `polinomi`: `[l,l,o,o]` *(equival a x^4+x^3+1 on el coeficient corresponent al terme independent no s'especifica en el vector)*
- `estat_inicial`: `[l,o,o,o]`
- `bits_de_sortida`: `20`
- `sortida`: `[1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1]`

2. Funció que implementa el generador pseudoaleatori *Shrinking*. **(2 punts**)

   Aquesta funció implementarà el generador *Shrinking* tal i com l'hem definit anteriorment. Així, la funció prendrà com a variables d'entrada tres paràmetres: `parametres_pol_1`, `parametres_pol_2` i `bits_de_sortida`; i retornarà la seqüència de sortida.

   - Les variables `parametres_pol_x` contindran el polinomi de connexions i l'estat inicial (en aquest ordre) de cada un dels polinomis que formen el generador *Shrinking*.
   - La variable `bits_de_sortida` contindrà el nombre total de bits de sortida del generador.
   - La funció retornarà la seqüència de sortida.
     - Exemple:
       - `parametres_pol_1: [[l,o,o,l],[l,o,o,o]]`
       - `parametres_pol_2: [[l,o,o,l,o],[l,o,o,o,o]]`
       - `bits_de_sortida: 20`
       - `sortida: [1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1]`

3. Funció que xifra i desxifra informació utilitzant el generador *Shrinking*. **(2 punts)**

   Aquesta funció implementarà un algorisme de xifrat en flux on el generador pseudoaleatori serà l'*Shrinking*. La funció prendrà com a variables d'entrada quatre paràmetres: `parametres_pol_1`, `parametres_pol_2`, `missatge` i `mode`; i retornarà la seqüència de sortida, que serà el text xifrat en cas que s'estigui xifrant i el text en clar si es desxifra.

   - Les variables `parametres_pol_x` contindran el polinomi de connexions i l'estat inicial (en aquest ordre) de cada un dels polinomis que formen el generador *Shrinking*.
   - La variable `missatge` contindrà el missatge a tractar, el text en clar en cas que es vulgui xifrar o el text xifrat en cas que es vulgui desxifrar. El text en clar podrà contenir un missatge de text de mida arbitrària. El text xifrat podrà

contenir un missatge de mida arbitrària expressat com a seqüència de zeros i uns.

- La variable `mode` contindrà els valors 'e' o 'd' en funció de si es vol xifrar o desxifrar (respectivament).
- La funció retornarà el missatge obtingut.
  - Exemple:
    - `parametres_pol_1: [[l,o,o,l],[l,o,o,o]]`
    - `parametres_pol_2: [[l,o,o,l,o],[l,o,o,o,o]]`
    - `missatge: 'PLAINTEXT'`
    - `mode: 'e'`
    - `sortida: '11111110001101100101001010110010010000011110100111000100010000010010'`

Exercici 2 (5 punts)

En aquest segon exercici treballarem amb el criptosistema IDEA, un criptosistema de bloc que està inclòs en els possibles criptosistemes a emprar en el protocol TLS 1.1.

Tal i com es descriu en la pàgina 21 del mòdul 4 “Xifres de clau compartida: xifres de bloc”, el criptosistema IDEA és un criptosistema de bloc que xifra blocs de 64 bits utilitzant una clau de 128 bits. Aquest criptosistema realitza 8 iteracions principals més una transformació de sortida i utilitza una funció d'expansió de la clau que permet transformar la clau de 128 bits en 52 claus de 16 bits necessàries en les diferents iteracions.

En aquest exercici implementarem la funció de generació de claus de xifrat i una única iteració del procés de xifrat tal i com es demana en cada un dels següents apartats:

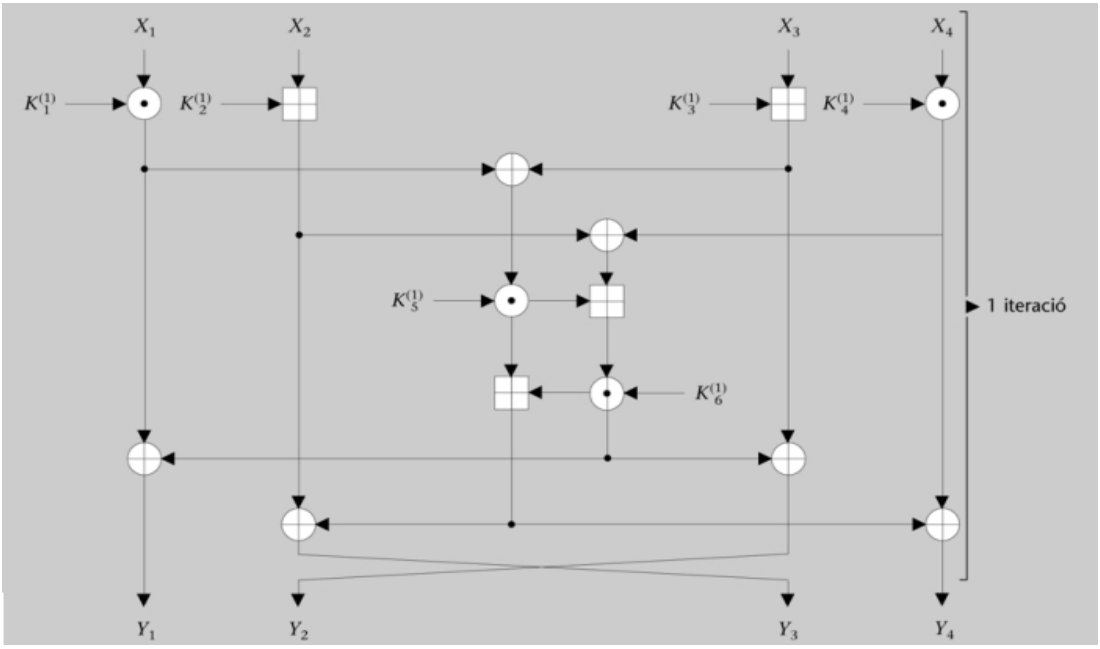
1. Funció que implementa la generació de claus de xifrat de l'IDEA. **(1,5 punts)**

   La funció prendrà com a variable d'entrada el valor: `key`; i en retornarà la clau expandida.

   - A la variable `key` s'escriurà la representació hexadecimal de la clau de 128 bits de l'IDEA;
   - La funció retornarà les 52 claus necessàries per les diferents iteracions concatenades en una única cadena hexadecimal.
     - Exemple:
       - `key: 'CF72654304050607080 90A0B00000E0F'`
       - `sortida: 'CF72654304050607080 90A0B00000E0F86080A0C0E101214160`

```
0001C1F9EE4CA181C2024282C0000383F3DC9950C10144850580
000707E7B932A1820283038400000E0FCF726543040506070809
0A0B0F9EE4CA86080A0C0E1012141600001C150C1014181C2024
2'
```

2. Funció que implementa una iteració de l'IDEA tal i com es descriu en el següent gràfic (en el mòdul didàctic trobareu la llegenda de cada un dels símbols). **(3,5 punts**)



La funció prendrà com a variables d'entrada dos paràmetres: `bloc_en_clar` i `clau_iteracio`; i retornarà quatre blocs de 16 bits.

- La variable `bloc_en_clar` contindrà els 64 bits corresponents a la concatenació dels valors X_1, X_2, X_3, X_4 , representats en hexadecimal.
- La variable `clau_iteracio` contindrà les sis claus de 16 bits corresponents a la concatenació dels valors $K_1^{(1)}, K_2^{(1)}, K_3^{(1)}, K_4^{(1)}, K_5^{(1)}, K_6^{(1)}$, representades en hexadecimal.
- La funció retornarà la seqüència de sortida que conté la concatenació dels quatre blocs de text xifrat Y_1, Y_2, Y_3, Y_4 , també en hexadecimal.
  - Exemple:
    - `bloc_en_clar: '0123456789ABCDEF'`
    - `clau_iteracio: 'CF726543040506070809 0A0B'`
    - `sortida: 'C3C383D88FF213E8'`

Format i data de lliurament

La data màxima de lliurament de la pràctica és el 28/04/2014 (a les 24 hores).

Juntament amb l'enunciat de la pràctica trobareu l'esquelet de la mateixa en format SAGE notebook worksheet (extensió .sws). Aquest mateix fitxer és el que heu de lliurar un cop hi codifiqueu totes les funcions.

En aquest esquelet també hi trobareu inclosos els jocs de proves dels diferents apartats. Tal i com s'esmenta en el fitxer sws, **no es pot modificar cap part del fitxer corresponent al joc de proves**. Això vol dir que heu de respectar el nom de les funcions i variables que s'han definit en aquest esquelet.

El lliurament de la pràctica constarà de d'**un únic fitxer** SAGE worksheet (extensió sws) on heu inclòs la vostra implementació.