Stream
Block \ cipher

Stream cipher (Lorentz)

ALGORITMO CHE DATO in input K chiave genera una sequenza di bit di lunghezza arbitraria, tale che sia "impossibile" data ogni sottosequenza della sequenza considerata prevedere i bits successivi.

→ SIMULA UNA SEQUENZA CASUALE.

1 1 ~~0~~ 1 0 1 0 0 ~~0~~ 0 0 1 ~~0~~ 1

ma non è casuale.

→ DATA LA SEQUENZA S LA CODIFICA DI UN messaggio m

m è semplicemente m ⊕ S

1) La sequenza costruita da una chiave K di lunghezza t non è in generale periodica.
2) La sequenza non deve avere proprietà statistiche "evidenti"

se ci sono proprietà statistiche è sempre possibile "indovinare" i bit successivi usando le stesse

$1 \to \frac{1}{3}$ $0 \to \frac{2}{3}$ $1 \to \frac{1}{2}$ $0 \to \frac{1}{2}$

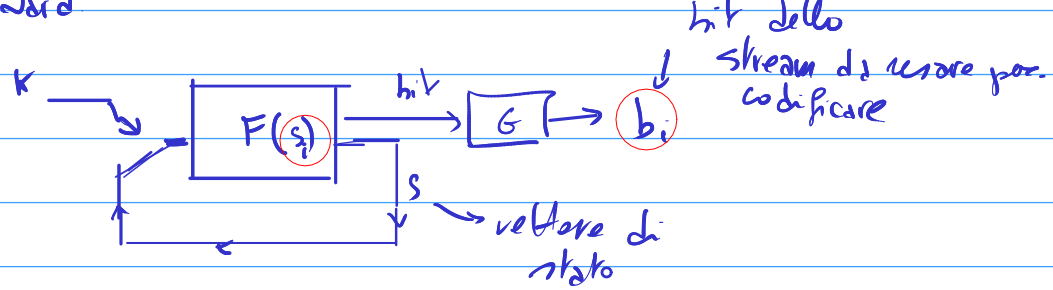
00
01
10
11

In generale si richiede che la sequenza sia pseudo-casuale → ∀ lunghezza ℓ, ogni sottosequenza di ℓ bits possibile (2^ℓ) compare

con la medesima frequenza

→ N.B.: [Stream cipher e generatori di numeri pseudo-casuali sono costruzioni collegate ma differenti.]

Metodo standard



RICOSTRUIRE s = stato iniziale consente di rigenerare tutti i bit della sequenza in output a partire dal momento in cui s è stato scoperto.

NON SERVE SCOPRIRE $K = S_0$ = stato iniziale

In generale si vuole che
- F non sia invertibile *
- Dall'output b_i non si abbiano informazioni usabili su s_i

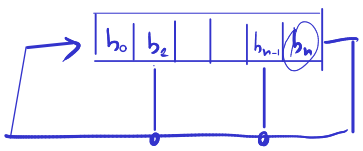
* deve essere "difficile" calcolare la preimmagine di $s_{i+1} = F(s_i)$.

→ Si vuole anche che la generazione delle sequenze sia veloce.

Idea base LFSR linear feedback shift register.

$b'_i = b_{(i+1)\%n+1}$

oss. uno shift ha periodo $n+1$



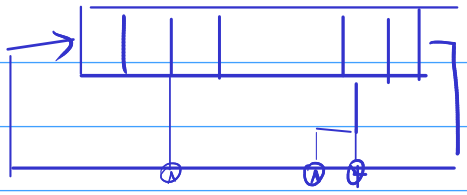
$b'_i = b_{i-1}$ se $i \geq 1$

$b_0 = b_n \oplus b_{j_1} \oplus \dots \oplus b_{j_k}$

LFSR perché si applicano solo degli XOR sul bit finale con altri bits del registro

N.B: solo 1 bit viene ricalcolato

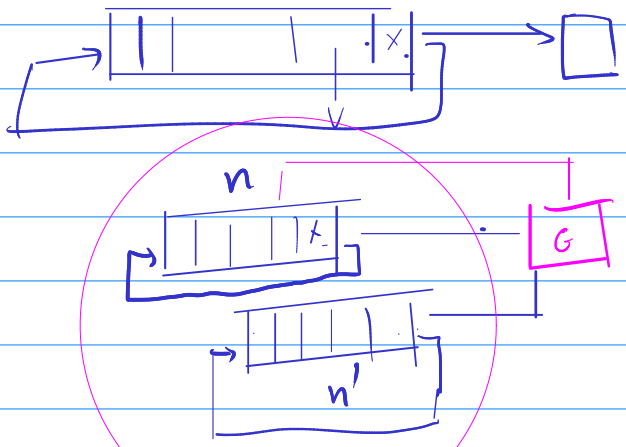
NLFSR → si fanno XOR ed AND



↓

esercizio: possiamo calcolare qualsiasi funzione booleana sullo stato del registro

Come costruire F dunque: NLFSR → inizializzarli con la chiave



Come mescolare un mazzo di carte

(1 2 ... n) e volete calcolare una sua permutazione casuale

```
for i = 1:n
    j = random(1:n)
    swap!(s[i], s[j])
end
```

→ n^n

```
for i = 1:n
    j = random(i:n)
    swap!(s[i], s[j])
end
```

→ $n!$

su n elementi ci sono n! possibili permutazioni

```
using Plots

function NaiveShuffle(x)
    r=copy(x)
    l=length(r)
    for i in 1:l
        j=rand(1:l)
        r[[i,j]]=r[[j,i]]
    end
    r
end

function FisherShuffle(x)
    r=copy(x)
    l=length(r)
    for i in 1:l
        j=rand(i:l)
        r[[i,j]]=r[[j,i]]
    end
    r
end

function SampleShuffle(n,tr=1000000)
    RN=Dict()
    RF=Dict()
    for _ in 1:tr
        r=collect(1:n)
        s0=NaiveShuffle(r)
        s1=FisherShuffle(r)
        (s0∈keys(RN)) ? (RN[s0]+=1) : RN[s0]=1
        (s1∈keys(RF)) ? (RF[s1]+=1) : RF[s1]=1
    end
    Dict([ k=>[RN[k] RF[k]]./tr for k in keys(RN)∩keys(RF) ])
end

function plotShuffle(n,tr=1000000)
    R=SampleShuffle(n,tr)
    vals=reduce(vcat,values(R))
    scatter(vals,label=["Naive" "Fisher"])
end


X=plotShuffle(4)
```