costruzione di numeri primi

```
function NextPrime(n)
    (n<=2)    && return 2
    (n%2==0)  && return NextPrime(n+1)
    PrimeP(n) && return n
    NextPrime(n+2)
end
```

$O(\log(n))$

```
function SolvayStrassen(n)
    n==2 && return true
    n>2 && (n%2==zero(n)) && return false
    W=(n+3)÷2
    for a in 2:W
        if (powermod(a,(n-1)÷2,n)!=(n+Jacobi(a,n))%n)
            print("Witness=",a,"\n")
            return false
        end
    end
    true
end
```

$(n+x) \bmod n$: x se $x \geq 0$; $n-x$ se $x < 0$

$-1, 0, 1$

$n-1, 0, 1$

deterministico

$O(\frac{n}{i})$ test

```
function SPprime(n,a)
    t,s = n-1,0
    while (t%2==0)
        s += 1
        t >>= 1
    end
    b=powermod(a,t,n)
    ((b==1) | (b==n-1)) && return true
    for j in 1:s-1
        b=powermod(b,2,n)
        b==n-1 && return true
    end
    false
end
```

```
function Miller(n)
    W = Integer(min(round(2*log(n)^2),
    for a in 2:W
        SPprime(n,a) || return false
    end
    true
end
```

$\frac{n}{4}$

volta GRH

Test: Agrawal, Kayal, Saxena (AKS)

polinomiale e deterministico.

```
function AKS(n)

    "Approximates (as integer) the k-th root of n"
    function int_root(n,k)
        x,y=1<<Integer(ceil(msb(n)/k)),0
        while(y<x)
            (y>0) && (x=y)
            y=((k-1)*x+(n÷x^(k-1)))÷k
        end
        x
    end

    "Is n an i-th power?"
    function ispower(n,i)
        int_root(n,i)^i==n
    end

    "Is n a perfect power for some i?"
    function ispower(n)
        for i=2:Integer(ceil(log2(n)))
            ispower(n,i) && return true
        end
        return false
    end

    "Find an integer r such that Or(n)>log2(n)^2"
    function findOrder(n)
        lval=Integer(floor(log2(n))^2)+1
        uval=Integer(floor(max(3,log2(n)^5)))
        for r=lval:uval
            (1∉[powermod(n,i,r) for i in 1:lval]) && return r
        end
    end

    "Compute (A*B  mod (X-1)^r) mod n where r=lenght(B)"
    function *(A::Vector{T},B::Vector{T}) where T<:Integer
        r=length(B)
        Rpol=fill(0,r)
        for i in 0:r-1
            Rpol+=A[i+1]*circshift(B,i)
            Rpol.%=n
        end
        Rpol
    end

    "Compute (X+a)^n mod (X-1)^r) mod n"
    function Ξ(a::Integer,n::Integer,r::Integer)
        Xpol=fill(0,r+1)
        Xpol[[1,2]]=[a,1]
        Rpol=fill(0,r+1)
        Rpol[1]=1
        for i in msb(n):-1:0
            Rpol=Rpol*Rpol
            (((n>>i) & 1)==1) && ( Rpol=Rpol*Xpol )
        end
        Rpol
    end
    # Step 1
```

radice intera k-esima cioè intero i più grande tale che $i^k \le n$

step 1

N.B in generale se $n = a^k$ allora $k \le \log_2(n)$

$(b_0 + b_1 x + \dots + b_{r-1} x^{r-1}) \cdot x \bmod (x^r - 1) =$

$= b_0 x + \dots + b_{r-2} x^{r-1} + b_{r-1} x^r =$

$= b_{r-1} + b_0 x + \dots + b_{r-2} x^{r-1}$

$(a_0 \dots a_{r-1}) \to a_0 + a_1 x + \dots a_{r-1} x^{r-1}$

$(b_0 \dots b_{r-1}) \to b_0 + b_1 x + \dots b_{r-1} x^{r-1}$

$a_0 (b_0 \dots b_{r-1}) + a_1 (b_{r-1} b_0 \dots b_{r-2}) + \dots$

$(x+a)^n \bmod (x-1)^r$

$X_{pol} = (X+a)$

$(x+a)^n$

passaggi $O(\log_2(n))$

```
# Step 1
ispower(n) && return false
# Step 2
r=findOrder(n)
# Step 3
for a in 2:r
   gcd(a,n)!=1 && gcd(a,n)!=n && return false
end
# Step 4
n≤r && return true
# Step 5
for a in 1:Integer(ceil(√(r))*ceil(log2(n)))
   Xpol=Ξ(a,n,r)
   Xpol[1]-=a
   Xpol[(n%r)+1]-=1
   Xpol = Xpol .% n
   Xpol==0*Xpol || return false
end
# Step 6
return true
end
```

n potenza di qualche intero $k < n$? — sì → N non primo; no → N primo.

Trovare r tale che $o_r(n) > \log(n)^2$

si dimostra che $\log^2(n) < r < \log^5(n)$

$r = O(\log^5(n))$

$\exists$ un fattore di n < r

$(x^{n\%r} - 1)$ → le iterazioni sono polinomiali nel numero di cifre di n

si basa essenzialmente nel fatto che

n è primo $\iff \forall a$ con $0 \le a < n-1$

$$(x-a)^n = x^n - a$$

n è primo $\iff (x-a)^n \bmod (x^r - 1) = (x-a)^n$

ove l'operazione mod $(x^r - 1)$ significa "dividere per $(x^r - 1)$ e prendere il resto" ma è equivalente nel polinomio che si scrive a porre

$x^r = 1$, $x^{r+1} = x$ ____ $x^{2r} = 1$ ____

$(x^5 - x^3 + x^2 - 1) \bmod (x^3 - 1)$ — $x^3 = 1$, $x^4 = x$, $x^5 = x^2$

$= x^2 - 1 + x^2 - 1 = 2x^2 - 2$

Trovare i primi è un problema "facile"

Fattorizzare $n=pq$ è "difficile" (almeno su macchine di Turing classiche quando p, q sono grandi"

Supponiamo di avere 2 sequenze (x_i, y_i) di numeri $1 \le t \le n$ con n ciuffo da fattorizzare

$GCD(n, x_i - y_i)$ ← 1 non ci copriamo un fc
n ($x_i = y_i$)
$\neq 1, n$ ⇒ ci fornisce un fattore di n.

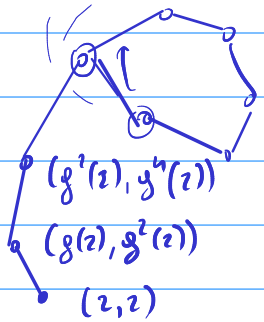
c'è una "collisione" se $GCD(n, x_i - y_i) \neq \{1, n\}$

Prob. di una collisione $\approx 1\sqrt{p}$ dove p è il più piccolo fattore di n

$n \to$ numero da fattorizzare

```
X=2
y=2
d=1
while (d=1)
    x = g(x)
    y = g(g(y))
    d = gcd(|x-y|, n)
end
d=n && return fail
d
```

$g(x) = x^2 + 1$



1
$p \neq n$
se $|x-y| = 0$

Alzare la probabilità di successo.

Stima di complessità $O(n^{\frac{1}{2}})$ Esponenziale.

OAEP / optimal asymmetric encryption padding.

RSA → 3 problemi /

1) Deterministico → data una chiave ogni messaggio ha esattamente una codifica.
(codifiche differenti ⇒ msg differenti)

2) IND-CPA → è possibile da parte di un attaccante costruire messaggi con struttura particolare

3) proprietà omomorfiche possono essere sfruttate per alcuni attacchi in presenza di oracoli/schemi di firma

(e la firma non garantisce che ogni messaggio con firma valida sia stato visto dal firmatario).

(m_1, s_1) (m_2, s_2) → $(m_1 m_2, s_1 s_2 \bmod n)$

Per quanto riguarda la firma → soluzione è l'uso di funzioni hash.

Non si firma il messaggio ma si firma una funzione calcolata sullo stesso per cui è difficile trovare preimmagini.

m $\quad s = h(m)^d \bmod n \rightarrow (m, s)$

verifica $\quad s^e = h(m) \bmod n$

$m_1 \rightarrow s_1 = h(m_1)^d \bmod n \longrightarrow (m_1, s_1)$

$m_2 \rightarrow s_2 = h(m_2)^d \bmod n \longrightarrow (m_2, s_2)$

$m_1 \cdot m_2 \qquad s_1 \cdot s_2 = [h(m_1) \cdot h(m_2)]^d \bmod n$

se $h(m_1)\,h(m_2) \neq h(m_1 m_2) \Rightarrow$

$s_1 s_2$ non è una firma valida per $m_1 m_2$.

Chi falsifica la firma deve trovare m_3 tale che

$$h(m_3) = h(m_1) \cdot h(m_2)$$

Si cercano funzioni hash h per cui questo è difficile.

MORALE: Non si firma direttamente un messaggio ma si firma una sua immagine tramite una opportuna funzione hash crittografica

h resistente alle
- preimmagini (dato x è difficile trovare m tale che $h(m)=x$)
- collisioni → è difficile trovare due messaggi con lo stesso hash → in particolare

preimmagini
A) dato m è difficile trovare m' con $h(m)=h(m')$

complessità
B) è difficile trovare m, m' con $h(m)=h(m')$

Si dice che c'è una collisione fra 2 messaggi m, m' se $h(m)=h(m') \Rightarrow$ una firma $s = h(m)^d \bmod n$ vale sia per m che per m'.

In A ci si aspetta che se $h: \{0,1\}^* \longrightarrow \{0,1\}^n$

serve testare $2^n + 1$ messaggi prima di trovare una collisione

h ha 2^n output differenti

principio della piccionaia

$2^n + 1$ input → almeno 2 output sono uguali

→ Dato m vogliamo che la prob. di trovare m' con $h(m') = h(m)$ sia $\simeq \frac{1}{2^n}$

In B → cerchiamo 2 messaggi con $h(m) = h(m')$ senza fissare a priori m

↓

paradosso dei compleanni

Il meglio che si può fare è far sì che la prob di trovarli entrambi sia $\simeq \frac{1}{2^{n/2}} \frac{1}{2}$

La probabilità che in un gruppo di 2n persone ce ne siano almeno 2 che hanno il medesimo compleanno è $> \frac{1}{2}$

Ragioniamo sulla probabilità che non ci siano collisioni (= compleanni il medesimo giorno) in funzione del numero di persone.

$$p_1 = \frac{365}{365} \qquad p_2 = \frac{365}{365} \cdot \frac{364}{365} \qquad p_3 = \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365}$$

$$p_n = p_{n-1} \cdot \frac{365 - n + 1}{365} = \frac{365! / (365-n)!}{365^n} =$$

$$= \frac{1}{365^n} \frac{365!}{(365-n)!} = \frac{n!}{365^n} \binom{365}{n}$$

↳ questo numero $\to 0$ al crescere di n.

$p_{23} \approx 0{,}492$ in particolare $\boxed{1 - p_{23} > \frac{1}{2}}$

ovvero la probabilità di collisioni è maggiore di quella che collisioni non ci siano

per approssimare il numero di persone richieste

prob che non ci siano collisioni è

$$p_*(n) = \prod_{k=1}^{n-1} \left(1 - \frac{k}{365}\right) = * \qquad n > 1$$

$$\boxed{1 - \frac{k}{365} \approx e^{-k/365}} \quad \text{perché} \quad e^x = 1 + x + \frac{x^2}{2!} + \dots$$

e prendiamo l'espansione del primo ordine (tenuto conto che siamo "vicini a zero")

$$* \approx \prod_{k=1}^{n-1} e^{-k/365} = e^{\frac{-1}{365}\sum_{k=1}^{n-1} k} = e^{\frac{(n-1)n}{-365 \cdot 2}}$$

vogliamo $e^{-\frac{(n-1)n}{365 \cdot 2}} < \frac{1}{2}$

$$-\frac{(n-1)n}{2} < (-\log 2) \cdot 365$$

$$n(n-1) > (2 \log 2) \cdot 365$$

$$n^2 > 2 \log 2 \qquad n > \sqrt{(2 \log 2) 365}$$

□

```
using Plots
plotlyjs()
```

```
function iscollision(x)
    length(unique(x))<length(x)
end

function samplebirtday(n,i,tr=5000)
    j=0
    for _ in 1:tr
        iscollision(rand(1:n,i)) && (j+=1)
    end
    j/tr
end
```

Campionamento casuale

```
function actualest(n,i)
    1-reduce(*,[1-j/n for j in 0:i-1])
end
```

Stima $1-\prod_{i=0}^{n}\left(1-\frac{j}{n}\right)$

```
function expest(n,i)
    C=√(-2*log(1-1/2))
    v=1-C*exp(-i*(i+1)/(2n))
    v>0 ? v : 0
end
```

Stima esponenziale

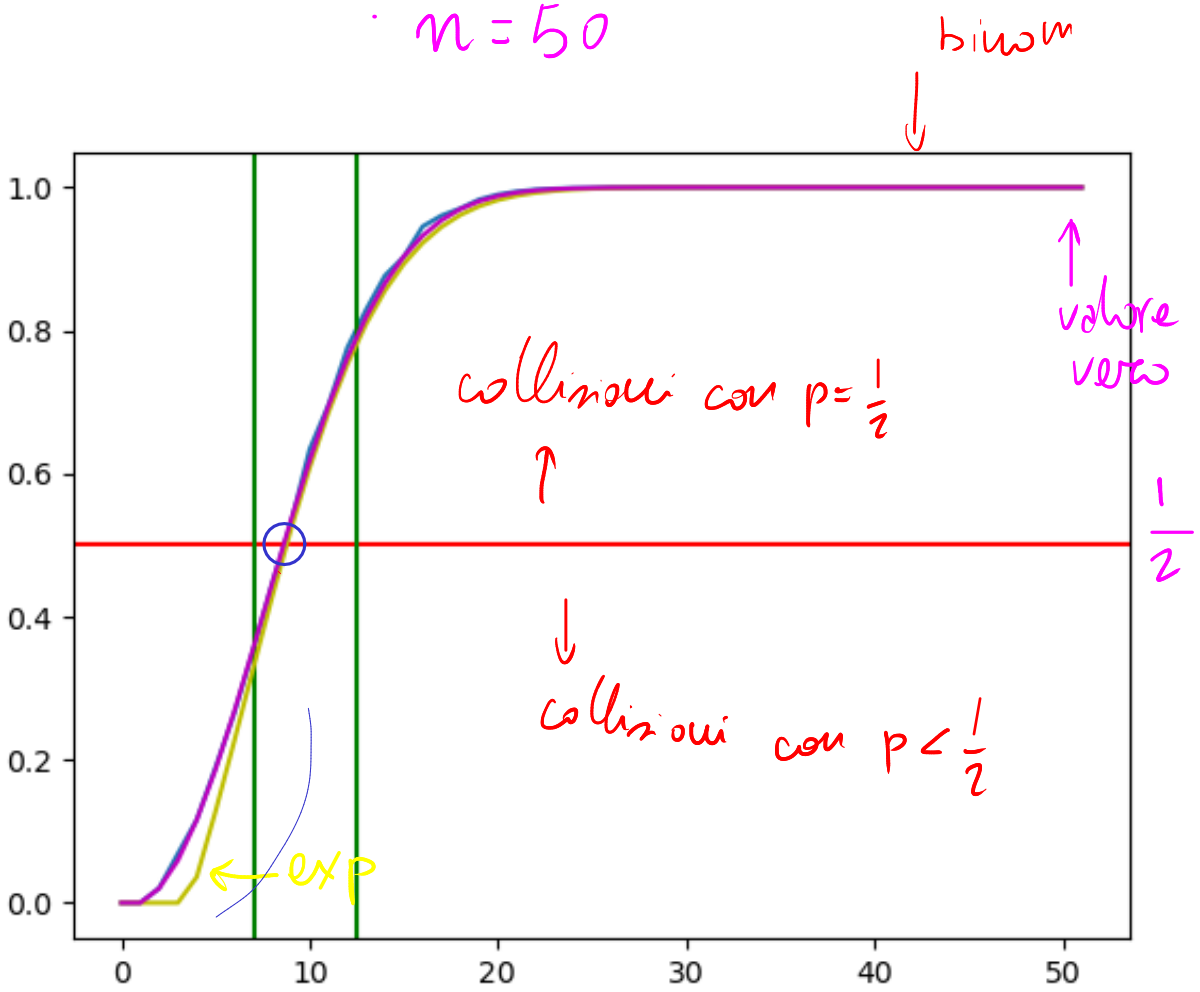
$1-Ce^{\frac{-i(i+1)}{2n}}$

```
function getprobs(n,tr=5000)
    R=[ 0 0 0 ]
    for i in 1:n
        R=[R ; [samplebirtday(n,i,tr) actualest(n,i) expest(n,i)] ]
    end
    R
end

function plotprobs(n,tr=5000)
    cv=ceil(√(2log(2)*n))
    m=getprobs(n,tr)
    plot(xrange=(0,n),yrange=(0,1))
    hline!([1/2],label="")
    vline!([cv,cv+2],label="")
    plot!(m[:,1],label="Sample")
    plot!(m[:,2],label="Actual")
    plot!(m[:,3],label="Exp")
end
```

Matrice e Grafico

$n = 50$



OAEP

r = nonce

$$\mathcal{E}(x) = f\left(x \oplus G(r) \,\|\, r \oplus H(x \oplus G(r))\right)$$

codifica RSA

x: messaggio

$r \oplus H(\ldots)$: concatenazione.

G, H opportune funzioni Hash resistenti alle collisioni

$H(w)$ ha "tanti bit in output quanti la lunghezza di r"

$G(w)$ è "tutti bit in output quanto la lunghezza di x"

DATO x MESSAGGIO

r NONCE.

Costruiamo $x \oplus G(r)$

$H(x \oplus G(r))$

Trasmettiamo $f(\underbrace{x \oplus G(r)}, r \oplus \underbrace{H(x \oplus G(r))})$

$f(\alpha \| \beta)$

OSS

DATO $(\alpha \| \beta)$ si può ricavare x

infatti $r = \beta \oplus H(\alpha)$

$x = \alpha \oplus G(r) =$

$= \alpha \oplus G(\beta \oplus H(\alpha))$

la codifica viene garantita da f.

1) la codifica non è più deterministica (dipende dal nonce r)

2) ALICE (o chi comunque sceglie x)

NON HA ALCUN CONTROLLO su r

r è differente da chi codifica

ma nemmeno BOB ha controllo su

$G(r)$ ed $H(x \oplus G(r))$