

# Blowfish, IDEA, (pseudo)juhuarvud

Erika Matsak, PhD

# Bruce Schneier

- Bruce Schneier (sündinud 15. 01. 1963)  
Ameerika krüptograaf, arvutite turvalisuse spetsialist.
- Mitmete raamatute autor, mis on pühendatud arvutite turvalisusele ja krüptograafiale
- Counterpane Internet Security asutaja ning juhtiv spetsialist
- Blowfish algoritmi esimene publikatsioon: Description of a “*New Variable-Length Key, 64-bit Block Cipher (Blowfish)*”, First Fast Software Encryption workshop in Cambridge, UK (proceedings published by Springer-Verlag, Lecture Notes in Computer Science #809, 1994), ISBN:3-540-58108-1

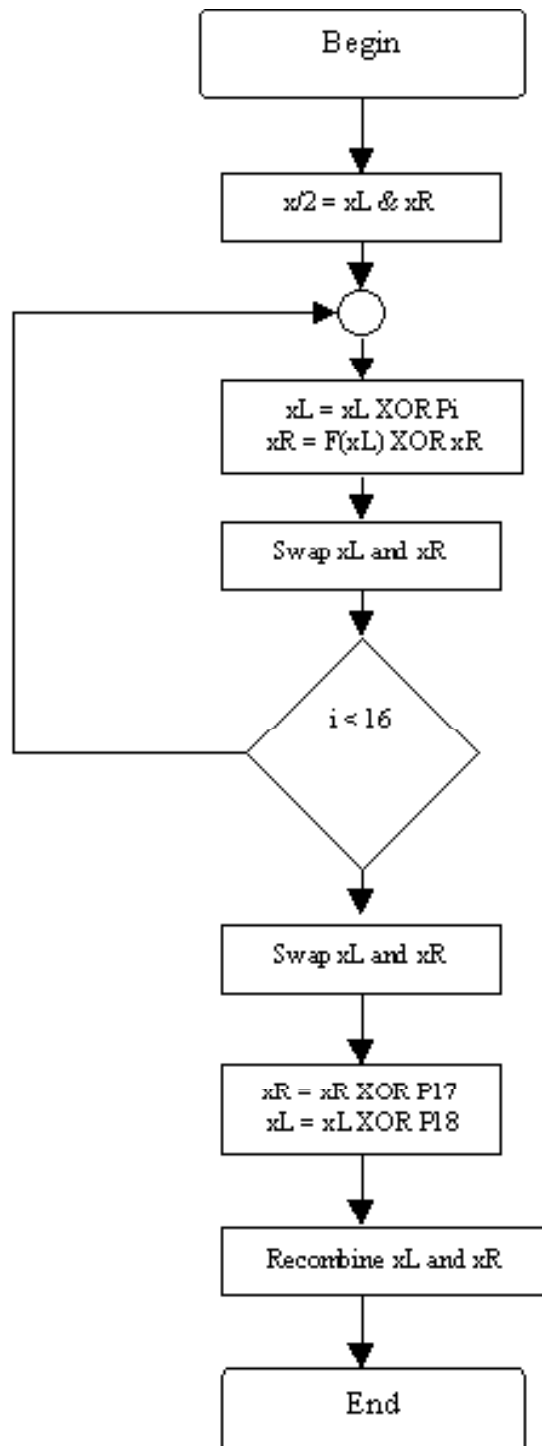


## Mõned raamatud:

“Beyond fear: thinking sensibly about security in an uncertain world”

“Secrets and lies: digital security in a networked world”

# Algoritm Blowfish



- Loodud 1993 aastal Bruce Schneier'i poolt
- Sümmeetriline plokk-šiffer
- Kasutab XOR operatsioone, liitmist ja permutatsioone.
- Vabalt levitav.
- Võtme pikkus varieeruv: 32 kuni 448 bitt, ehk 1-14 sõna (kui sõna pikkus 32 bitti)

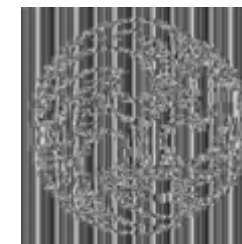
Ploki suurus – 64 bitti

Aluseks on Feisteli võrk (16 raundi)

XOR teostatakse 32-bitiselt

Algoritm sisaldab kaks osa: alamvõtmete genereerimine ning andmete šifreerimine

Pildi krüpteerimine:



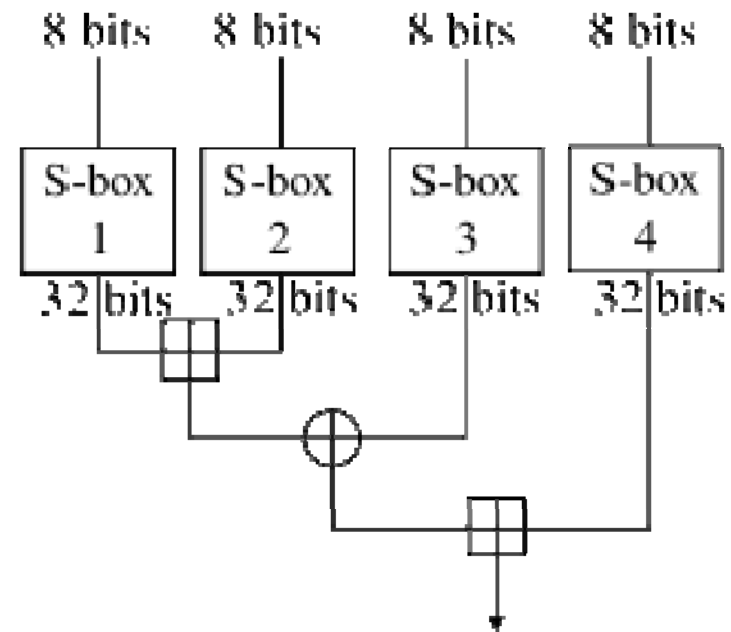
# Algoritm Blowfish. Funktioon $f$

## Funktsiooni $f$ arvutamine:

Jagada  $X_L$  neljaks 8-bitiseks plokkiks: a, b, c ja d.

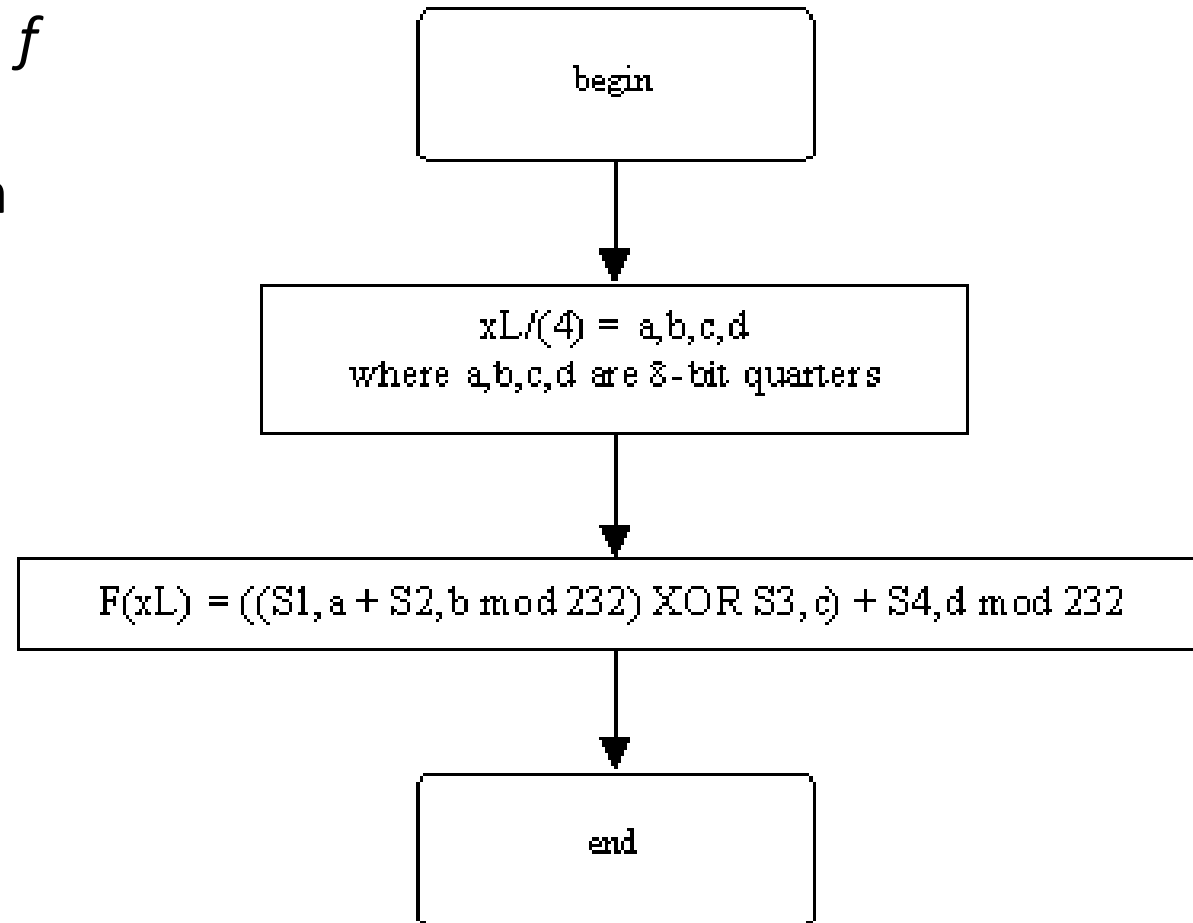
Seejärel arvutada:

$$((S1[a] + S2[b] \bmod 2^{32}) \oplus S3[c]) + S4[d] \bmod 2^{32};$$



# Algoritm Blowfish, funktsioon $f$

Funktsiooni  $f$   
arvutamise  
plokk skeem



# Algoritm Blowfish. Alamvõtmed.

- Kasutab väga palju alamvõtmeid, mis peavad olema genereeritud enne šifreerimist, genereeritakse selle sama Blowfish-i algoritmiga
  1. Jagada võti 32 bitisteks plokkideks  $k_1 \dots k_n$ .
    - Teostada XOR  $P_1$  ja  $k_1$  vahel,  $P_2$  ja  $k_2$  ja  $P_n \dots k_n$  vahel, ehk  $a_i = P_i \oplus k_i$
    - Kui esialgne võti on liiga lühike, siis moodustakse sellest piisavalt pikk konkatensatsioon, näiteks KEYKEYKEYKEY..... jne

P maatriks HEX kujul, mis on genereeritud arvust PI (ehk  $\pi$ ), kus iga arv on kahendkoodis 32 bitti, ehk kokku on  $32 \times 18 = 576$  bitti, millega saab katta iga võtme pikkust

0x <b>243f6a88</b>	0x <b>85a308d3</b>	0x <b>13198a2e</b>	0x <b>03707344</b>
0xa <b>4093822</b>	0x <b>299f31d0</b>	0x <b>082efa98</b>	0x <b>ec4e6c89</b>
0x <b>452821e6</b>	0x <b>38d01377</b>	0x <b>be5466cf</b>	0x <b>34e90c6c</b>
0xc <b>0ac29b7</b>	0xc <b>97c50dd</b>	0x <b>3f84d5b5</b>	0xb <b>5470917</b>
0x <b>9216d5d9</b>	0x <b>8979fb1b</b>		

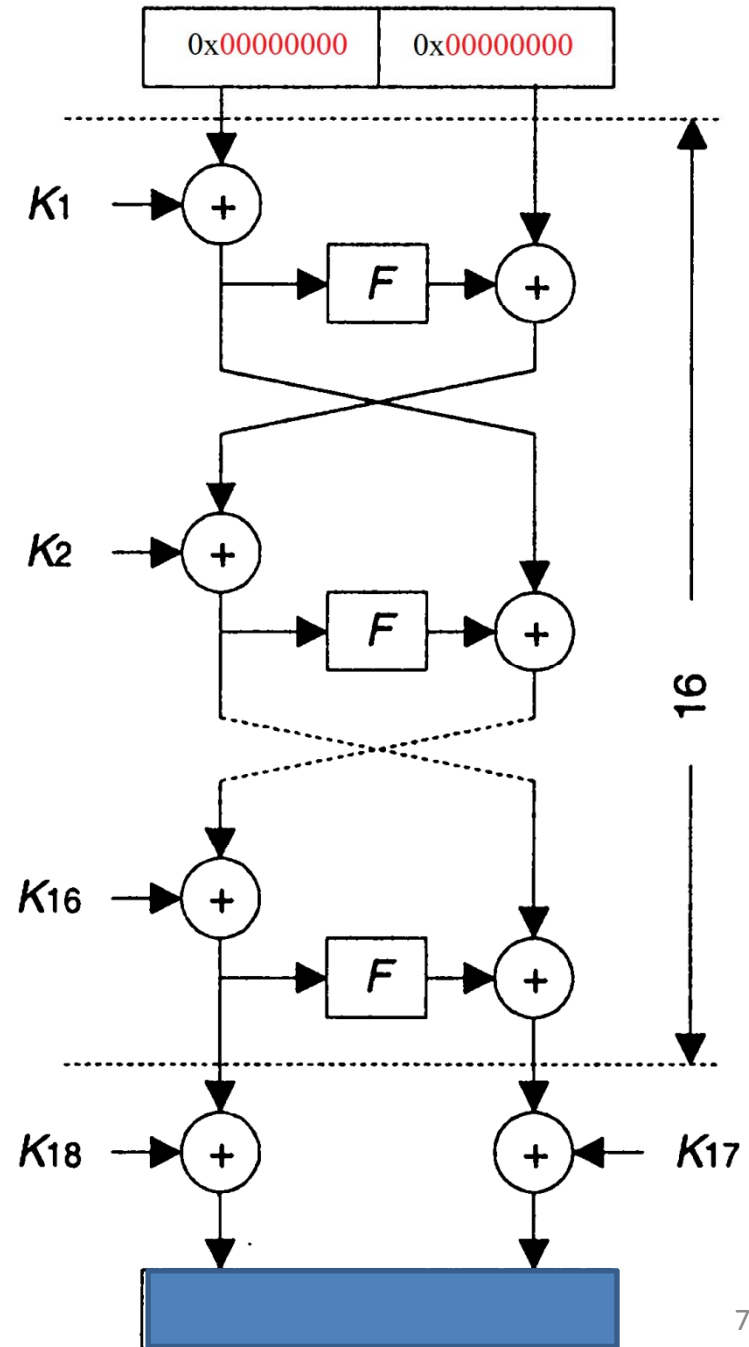
# Algoritm Blowfish. Alamvõtmed.

Kasutades esimesel etapil genereeritud alamvõtmeid  $k_1 \dots k_{18}$  šifreeritakse rida (algoritmiga Blowfish), mis koosneb ainult nullidest.

Läbitakse 16 raundi

Seejärel šifreeritakse  $K_1$  ja  $K_2$  ning tulemuseks on  $K_3$  ja  $K_4$ .

Seejärel asendatakse S-boxide vastavate kohtade sisu (näiteks 0 ja 1 kohad S1-boxis)  
Nii jätkatakse, kuni kõik K-d ja S-id on asendatud.



# Algoritm Blowfish. Alamvõtmed

## Lähemalt:

- Sisestatud võti teisendatakse kahendkoodiks
- Näiteks, kui võtmeks on “my password 8”, siis sellele vastab: 01101101 01111001 00100000 01110000  
01100001 01110011 01110011 01110111 01101111  
01110010 01100100 00100000 00111000
- Iga 32 bitti (märgitud erineva värviga) kasutakse alamvõtmete genereerimiseks, näiteks esimesed 32 bitti (sinine osa) pannakse XOR operaatoriga kokku P1-ga (0x**243f6a88**, ehk 00100100 00111111 01101010 10001000).

# Algoritm Blowfish. Alamvõtmed.

- Alamvõtmete järgmine osa on seotud 16 Blowfish raundiga. Alustatakse sisendist, mille pikkus on 32 bitti ning kõik bitid on nullid. See jagatakse 4-ks 8-bitiseks osaks. Iga taoline osa tähistab kümnendkoodis S-box-i sisendit.

## Näide.

Kui tegu on arvuga:

10011011 10101111 11100101 10011000,

siis kümnendkoodis on tegu järgmise arvude jadaga:

155 175 229 152.

Järelikult esimesest S-boxist tuleb võtta element nr 155, teisest S-Boxist element 175 jne. Saadud arvudega arvutatakse funktsiooni  $f$ :

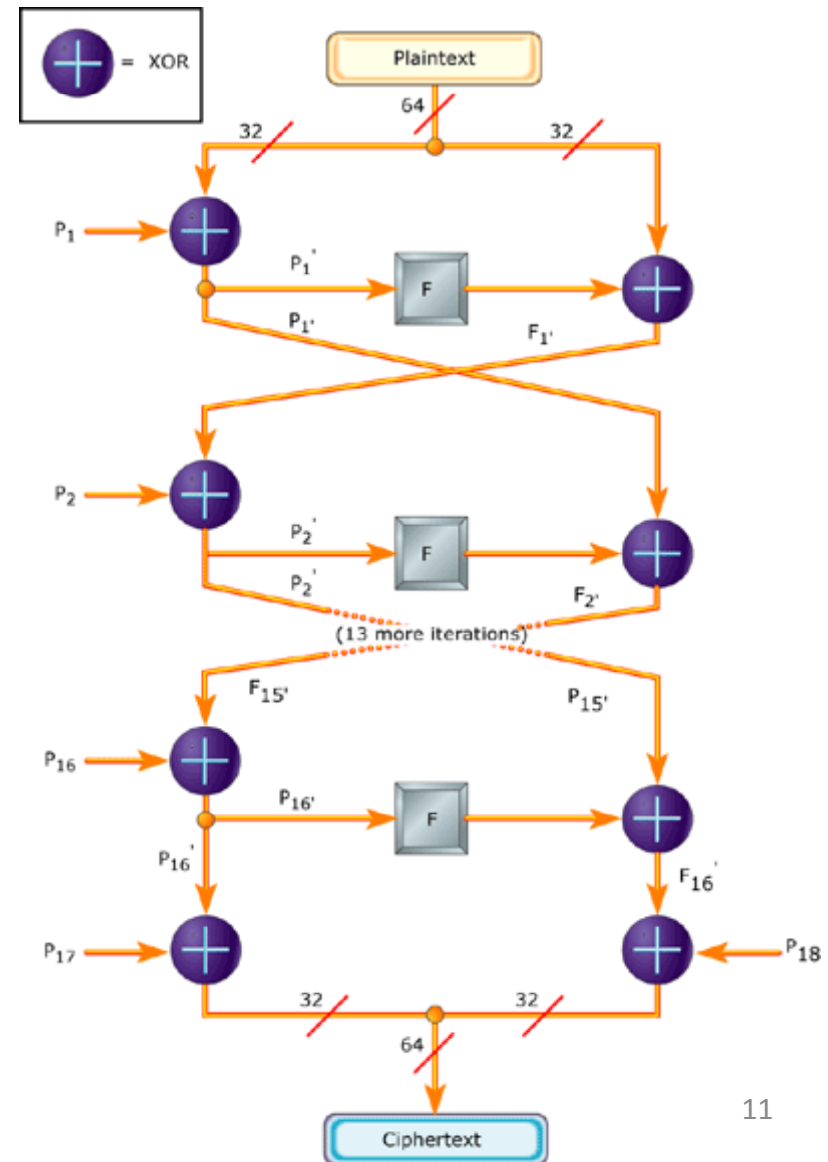
$$((S1[a] + S2[b] \bmod 2^{32}) \oplus S3[c]) + S4[d] \bmod 2^{32};$$

# Algorithm Blowfish, 16 raundi

```
for (int i = 0; i < 16; i++) {  
    Xl = Xl ^ this.ctx.P[i];  
    Xr = F(Xl) ^ Xr;  
    tmp = Xl;  
    Xl = Xr;  
    Xr = tmp;  
}
```

# Algoritm Blowfish

- Kui alamvõtmed on valmis, siis krüpteeritakse teksti sama algoritmiga.
- Avateksti krüpteerimiseks kasutatakse uut S-boxi
- Seejärel jagatakse avatekst 64 bitisteks plokkideks, jaotatakse pooleks: vasakuks ja paremaks pooleks ning läbitakse kõik raundid.



# Algoritm IDEA (International Data Encryption Algorithm)

- Sümmeetriline plokkšiffer. Loodud Xuejia Lai ning James L. Massey poolt
- Algoritmi esimene versioon on välja töötatud 1990aastal ja aasta pärast täiendatud. Patent kehtib kuni 2010–2011. Kasutamine mitte ärilistel eesmärkidel on tasuta



**James L. Massey,**  
**11. veebruar 1934**

Võtme pikkus 128 bitti  
Krüpteeritud plokki suurus 64 bitti

Eesmärgiks oli luua võimalikult lihtsa realisatsiooniga algoritm, mis omaks kõrget “kindlust” turvalise aspektist



**Xuejia Lai**  
**4. juuni 1954**

# Algoritm IDEA (International Data Encryption Algorithm)

## Algoritmi karakteristikud:

- Ploki suurus on piisavalt suur (64bitti). Kasutatakse ahelarežiimi (CBC)
- Võtme pikkus on oluline. Lühikese võtme puhul ei ole kõikide võtmete arv piisav, et murdmine oleks keeruline. Antud juhul on kõikide võimalike võtmete arv  $2^{128}$
- Krüpteeritud tekst on seotud võtmega “keerulisel” viisil (Segamine)
- Peidab efektiivselt krüpteeritud teksti statistilised omadused. Ehk iga krüpteeritud bitt sõltub paljudest avateksti bittidest.

# Algoritm IDEA (International Data Encryption Algorithm)

- Iga ploki (64 bitti) jagatakse neljaks osaks, iga osa suurus 16 bitti. Kõik algebralised operatsioonid teostatakse 16 biti peal.
- Šifreerimiseks ja dešifreerimiseks kasutatakse sama algoritmi.
- Operatsioonid: XOR, liitmine mod  $2^{16}$  alusel, korrutamine  $2^{16+1}$  alusel. Distributiivsus ja assotsiatiivsus nende operatsioonide vahel ei kehti. See raskendab krüptoanalüüsi ja murdmist. Operatsioonid annavad võimaluse loobuda S-Boxidest ning asendustabelitest.

$$a * (b + c) \not\leftrightarrow (a * b) + (a * c)$$

$$a + (b \oplus c) \not\leftrightarrow (a + b) \oplus c$$

# Algoritm IDEA. Võtmete genereerimine

- 128 bitisest võtmest genereeritakse iga 8 raundi jaoks 6 alamvõtit, iga alamvõtme pikkus 16 bitti. Kokku on vaja  $52=8 \times 6 + 4$  erinevat alamvõtit.
- Kõigepealt 128 biti pikkust võtit jagatakse kaheksaks osaks. Nimetatud osadest saadakse 8 esimest alamvõtit.

$$(K_1^{(1)} K_2^{(1)} K_3^{(1)} K_4^{(1)} K_5^{(1)} K_6^{(1)} K_1^{(2)} K_2^{(2)})$$

6tk esimese raundi jaoks ja 2tk teise raundi jaoks

Seejärel teostakse esialgses 128 bitises võtmes registri nihutamine 25 positsiooni võrra. Seejärel jagatakse 128 bitti uuesti kaheks osaks, millest saadakse järgmised 8 alamvõtit

$$(K_3^{(2)} K_4^{(2)} K_5^{(2)} K_6^{(2)} K_1^{(3)} K_2^{(3)} K_3^{(3)} K_4^{(3)})$$

Jätkatakse, kuni on valmis  $8 \times 6 + 4 = 52$  alamvõtit.

# Algoritm IDEA. Võtmete genereerimine

<b>Raundi number</b>	<b>Alamvõtmed</b>
1	$K_1^{(1)} K_2^{(1)} K_3^{(1)} K_4^{(1)} K_5^{(1)} K_6^{(1)}$
2	$K_1^{(2)} K_2^{(2)} K_3^{(2)} K_4^{(2)} K_5^{(2)} K_6^{(2)}$
3	$K_1^{(3)} K_2^{(3)} K_3^{(3)} K_4^{(3)} K_5^{(3)} K_6^{(3)}$
4	$K_1^{(4)} K_2^{(4)} K_3^{(4)} K_4^{(4)} K_5^{(4)} K_6^{(4)}$
5	$K_1^{(5)} K_2^{(5)} K_3^{(5)} K_4^{(5)} K_5^{(5)} K_6^{(5)}$
6	$K_1^{(6)} K_2^{(6)} K_3^{(6)} K_4^{(6)} K_5^{(6)} K_6^{(6)}$
7	$K_1^{(7)} K_2^{(7)} K_3^{(7)} K_4^{(7)} K_5^{(7)} K_6^{(7)}$
8	$K_1^{(8)} K_2^{(8)} K_3^{(8)} K_4^{(8)} K_5^{(8)} K_6^{(8)}$
<b>Väljundteisendus</b>	$K_1^{(9)} K_2^{(9)} K_3^{(9)} K_4^{(9)}$

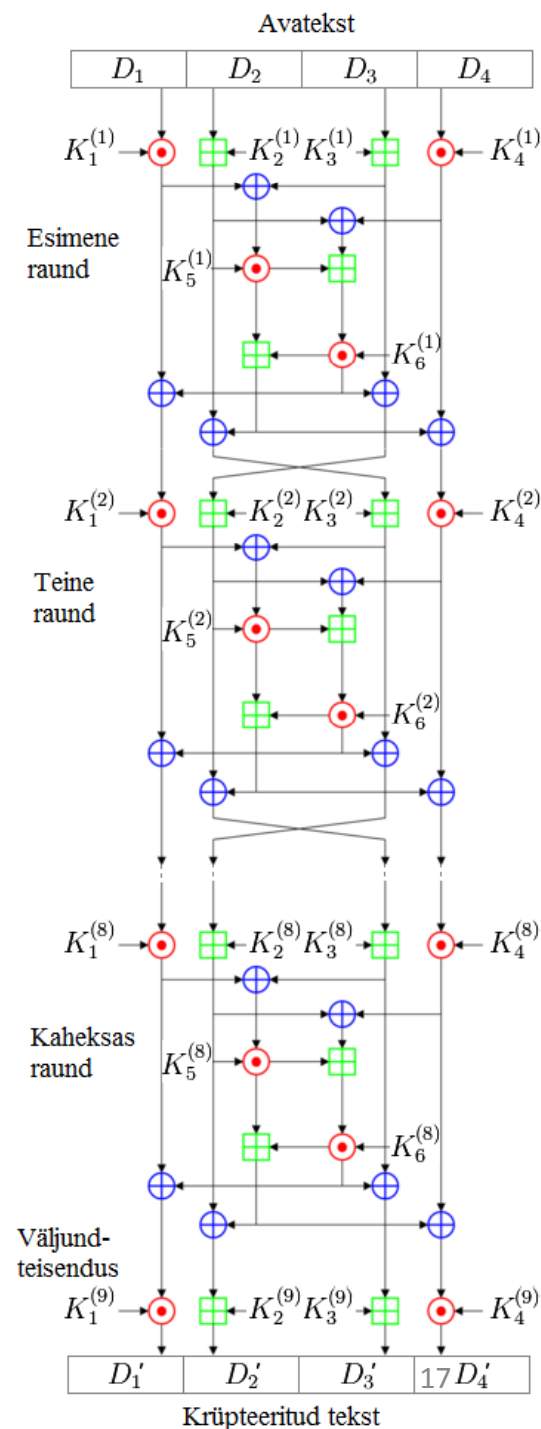
# Algoritm IDEA

$\oplus$  XOR    $\odot$  korrutamine    $\boxplus$  liitmine

Avateksti iga 64 bitine plokk jagatakse neljaks osaks:  $D_1, D_2, D_3, D_4$

Šifreerimise protsess koosneb 8 ühesugusest raundist. Igal raundil kasutakse omi alamvõtmeid.

Kasutatakse tehteid XOR, korrutamist ja liitmist. Kui on läbitud 8 raundi, siis teostatakse väljundteisendus. Saadud krüptogrammi osad pannakse kokku konkatenatsiooniga.



# Algoritm IDEA. Matemaatiline kirjeldus

- Sisend:  $(D_1^{(0)}, D_2^{(0)}, D_3^{(0)}, D_4^{(0)})$
- Iga raundi jaoks arvutatakse:

$$\begin{array}{ll}
 A^{(i)} = D_1^{(i-1)} * K_1^{(i)} & D^{(i)} = D_4^{(i-1)} * K_4^{(i)} \\
 B^{(i)} = D_2^{(i-1)} + K_2^{(i)} & \longrightarrow E^{(i)} = A^{(i)} \oplus C^{(i)} \\
 C^{(i)} = D_3^{(i-1)} + K_3^{(i)} & F^{(i)} = B^{(i)} \oplus D^{(i)}
 \end{array}$$

$$D_1^{(i)} = A^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$

$$D_2^{(i)} = C^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$

$$D_3^{(i)} = B^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$

$$D_4^{(i)} = D^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$

# Pseudo-juhuarvud

- John von Neumann «*igaüks, kel on nõrkus juhuslike arvude saamise aritmeetiliste meetodite vastu – on kahtlemata patune*».
- Mitte ükski determineeritud algoritm ei saa genereerida täiesti juhuslike arve. Saab vaid lähendada mõningaid juhuarvude omadusi.

Võimalikud probleemid:

- Genereeritavad arvud, ei ole üks-teisest sõltumatud
- Mõned arvud on “vähem” juhuslikud, kui teised
- Mitteühtlane ühemõõtmeline jaotus (samas peaks aga iga arvu esinemise sagedus olema enam-vähem ühesugune)
- Pööratavus



John von Neumann  
(28. 12. 1903 – 8. 02.  
1957)

# Pseudo-juhuslikud jadad

- Kas jada 0100100010 on (pseudo) juhuslik?
- Intuitsioon 1: Jada on pseudojuhuslik, kui tema järgmise biti ennustamise tõenäosus ei erine kuigi palju  $1/2$  -st.
- Intuitsioon 2: Jada on pseudojuhuslik, kui ta on eristamatu  $n$ -ö päris-juhuslikust jadast.

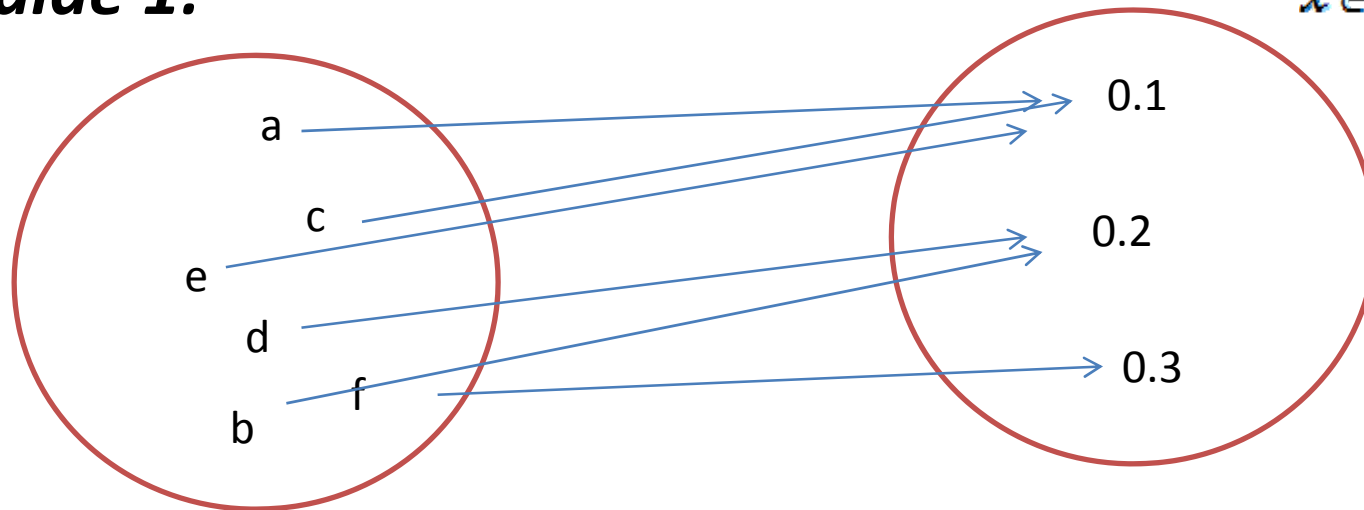
[http://home.cyber.ee/ahtbu/pseudo\\_slides.pdf](http://home.cyber.ee/ahtbu/pseudo_slides.pdf)

# Tõenäosusjaotused

- Tõenäosusjaotuseks  $D$  lõplikul hulgal  $\mathcal{X}$  nimetatakse funktsiooni  $D: \mathcal{X} \rightarrow [0,1]$  mis igale elemendile  $x \in \mathcal{X}$  seab vastavusse tema esinemise tõenäosuse  $D(x)$ .

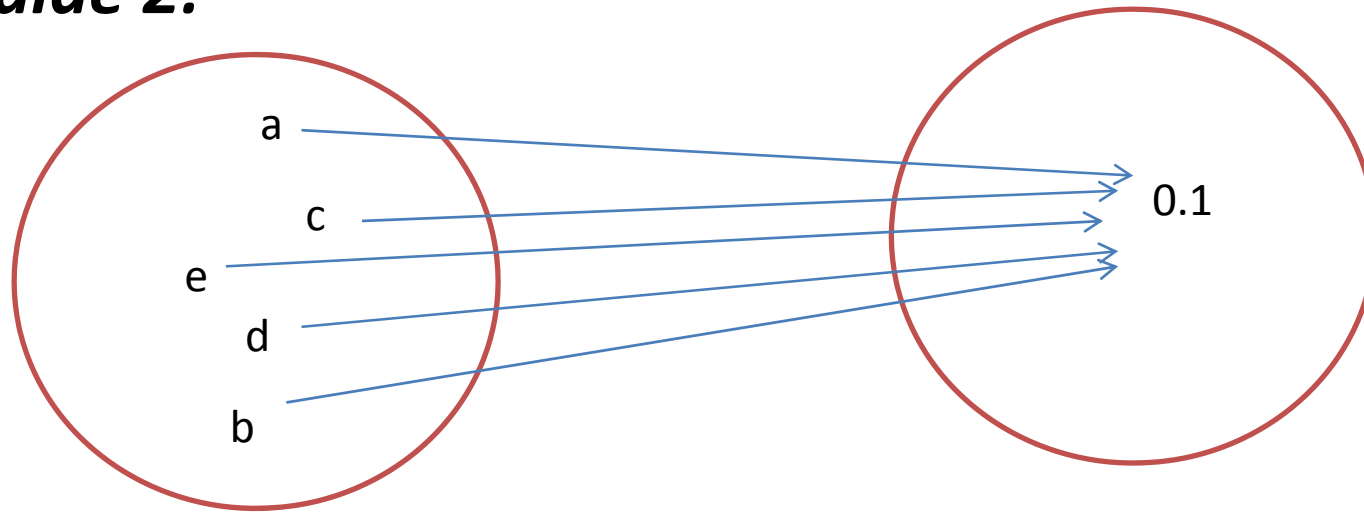
$$\sum_{x \in \mathcal{X}} D(x) = 1$$

**Näide 1:**

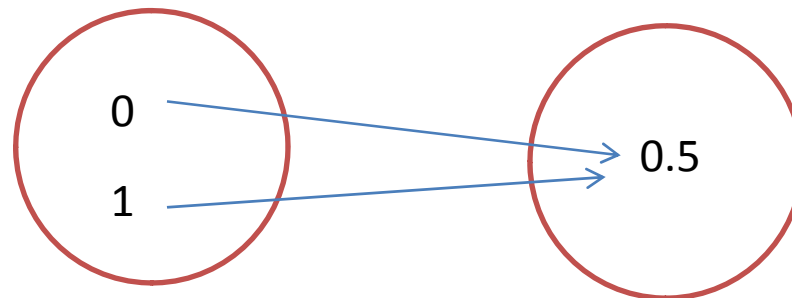


# Tõenäosusjaotused

**Näide 2:**



**Näide 3:**



# Pseudo-juhuarvud.

## Lineaarne kongruentsivalem

### *(Linear congruential formula)*

$$X_{k+1} = (aX_k + c) \bmod m$$

- kasutatakse lihtsamates situatsioonides, algoritm ei ole krüpteerimise mõttes tugev
  - Näide: Java programmeerimiskeel: RANDOM

$a, c, m$  täisarvulised konstandid, tulemuseks on arvude jada vahemikus  $0 \leq X_n < m$ .

Juhuarv sõltub  $X_0$  -st, kuigi paljud omadused genereeritud jadas sõltuvad koefitsientidest  $a, c, m$  mitte esimest elemendist  $X_0$

# Pseudo-juhuarvud.

## Lineaarne kongruentsivalem (*Linear congruential formula*)

- Koefitsientide valik on väga oluline. Siin peab  $m$  olema võimalikult suur (tavaliselt suurim täisarv, mida arvuti suudab genereerida, ehk  $\approx 2^{31}$ )
- Vähesed koefitsiendid annavad jada, mis on lähedane juhuslikule ning on realiseeritav 32 bitisel protsessoril.
- Üks tuntud hea valik on  $a = 7^5$ ,  $c = 0$ ,  $m = 2^{31} - 1$ . Kui aga vastane teab koefitsiente ja kas või üht arvu, siis on kogu jada tema jaoks arvutatav.
- Kui vastane ei tea koefitsiente, aga teab kolme järjest paikneva arvu, siis on võimalik lahendada võrrandeid:

$$X_1 = (a X_0 + c) \bmod m$$

$$X_2 = (a X_1 + c) \bmod m$$

$$X_3 = (a X_2 + c) \bmod m$$

ning leida  $a$ ,  $c$ ,  $m$ .

- Võimalik lahendus: lisada igale genereeritavale arvule  $n$ -ö jooksev kellaaeg sekundites

# Pseudo-juhuarvud, *LFSR (Linear feedback shift register)*

- Koosneb kahest osast: registri nihutamisest ning tagasiside funktsioonist.
  - Kõik bitid nihutatakse registris ühe positsiooni võrra paremale
  - Uue biti arvutamine toimub eelnevalt välja valitud positsioonidel paiknevatest bittidest funktsiooni abil
  - Funktsioon on loomu poolest XOR kõikidest bittidest, vastavate koefitsientidega

$$S_L = (c_1 * S_{L-1}) \oplus (c_2 * S_{L-2}) \oplus \dots \oplus (c_L * S_0)$$

$$S_{L+1} = (c_1 * S_L) \oplus (c_2 * S_{L-1}) \oplus \dots \oplus (c_L * S_1)$$

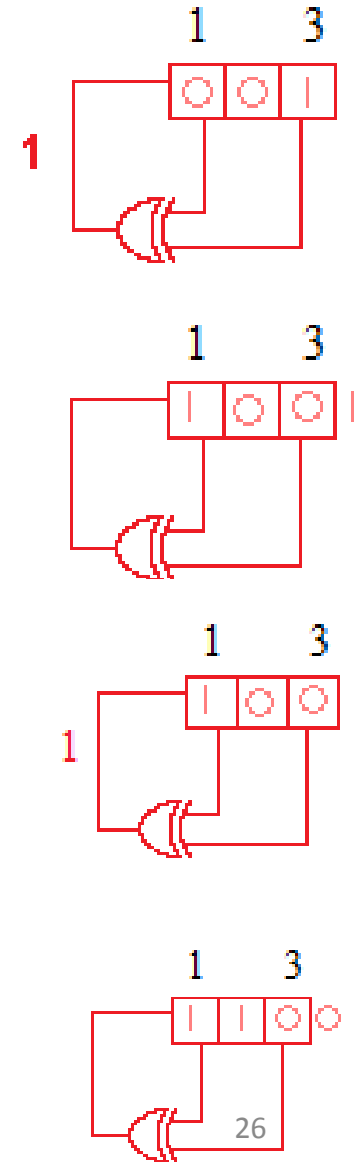
$$S_j = (c_1 * S_{j-1}) \oplus (c_2 * S_{j-2}) \oplus \dots \oplus (c_L * S_{j-L})$$

# Pseudo-juhuarvud, *LFSR (Linear feedback shift register)*

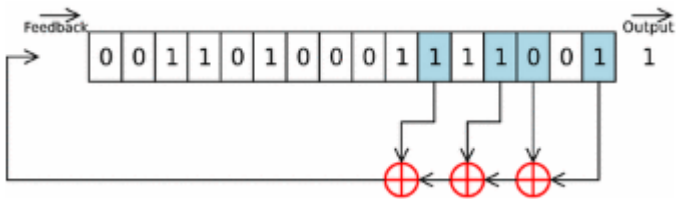
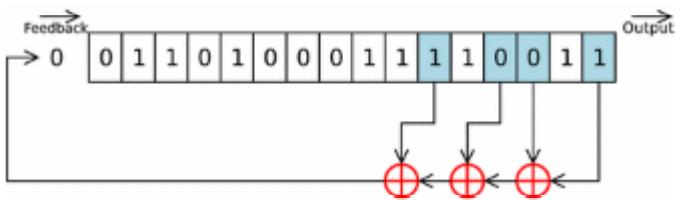
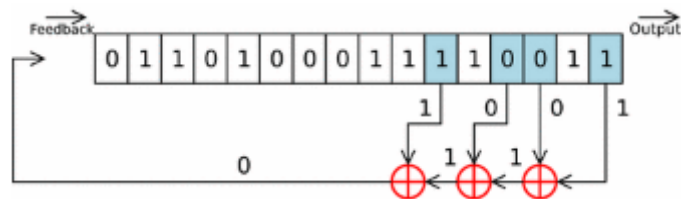
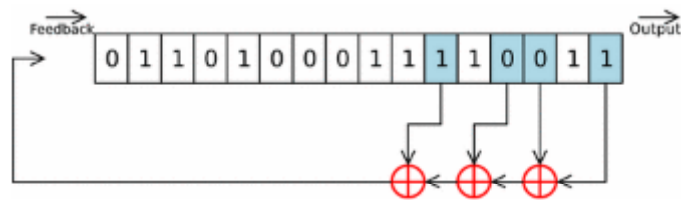
## Näide:

- Olgu assotsieeritud polünoom (hulkliige) :  $x^3+x+1$  ning jada: [0,0,1]
- Sel juhul funktsioon näeb välja järgmiselt:  
 $S_j = S_{j-1} \oplus S_{j-3}$ , kus  $c_1=1$ ,  $c_2=0$ ,  $c_3=1$

Samm	Olek
0	[0,0,1]
1	[1,0,0]
2	[1,1,0]
3	[1,1,1]
4	[0,1,1]
5	[1,0,1]
6	[0,1,0]
7	[0,0,1]



# Pseudo-juhuarvud, *LFSR (Linear feedback shift register)*



$$x^{16} + x^{14} + x^{13} + x^{11} + 1.$$

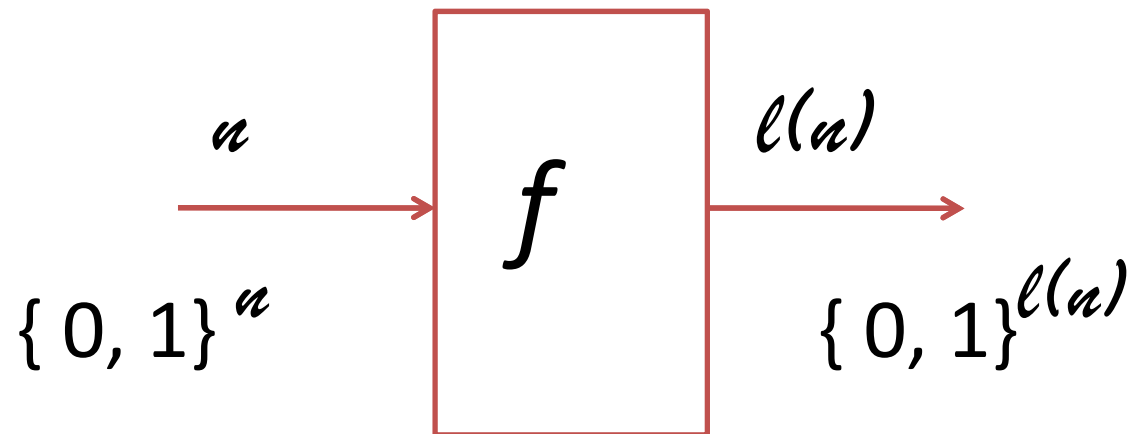
Period:  $2^n - 1$

Ehk antud juhul 65535

# Pseudo-juhuarvude generaatorid

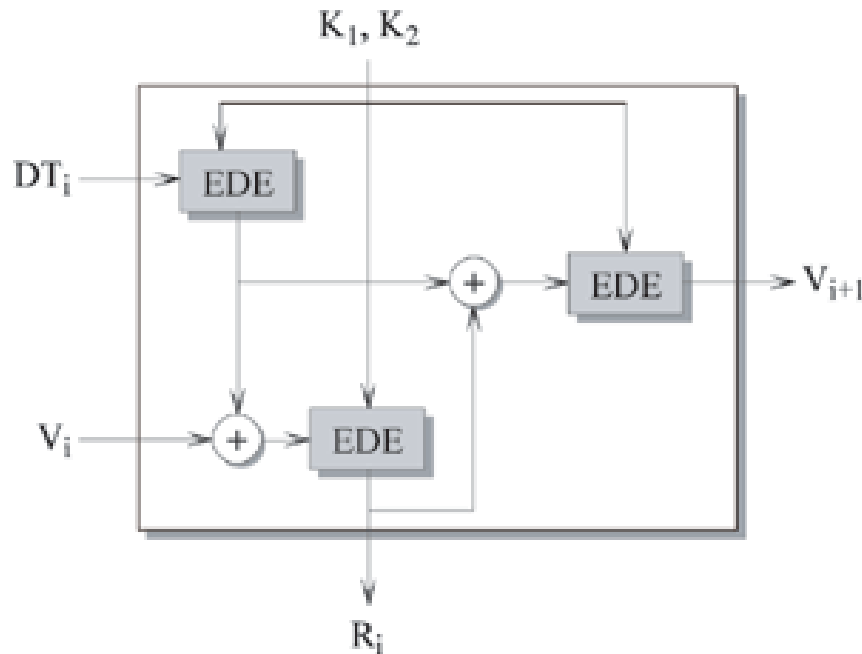
- Juhuarvude generaatori sisendi pikkuse  $n$  ja väljundi pikkuse  $l(n)$  kohta eeldatakse ainult seda,  $l(n) > n$ .
- Erijuhul on generaatori väljund vaid ühe biti võrra pikem sisendist, ehk pikkusega  $n+1$ . Praktikas on aga enamasti vaja generaatoreid, mis “venitavad” rohkem.

[http://home.cyber.ee/ahtbu/pseudo\\_slides.pdf](http://home.cyber.ee/ahtbu/pseudo_slides.pdf)

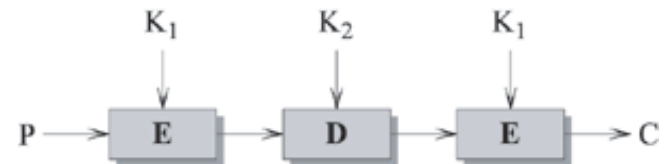


# Pseudo-juhuarvude generaator. ANSI X9.17

- $Dt_i$  – kuupäeva ning aja väärtus enne  $i$  raundi käivitamist
- $V_i$  – algväärtus raundil  $i$
- $R_i$  – pseudo-juhuarv, mis on moodustatud  $i$  raundil
- $K_1, K_2$  – võtmed, mis kasutatakse igal raundil (iga võti 56 bitt)



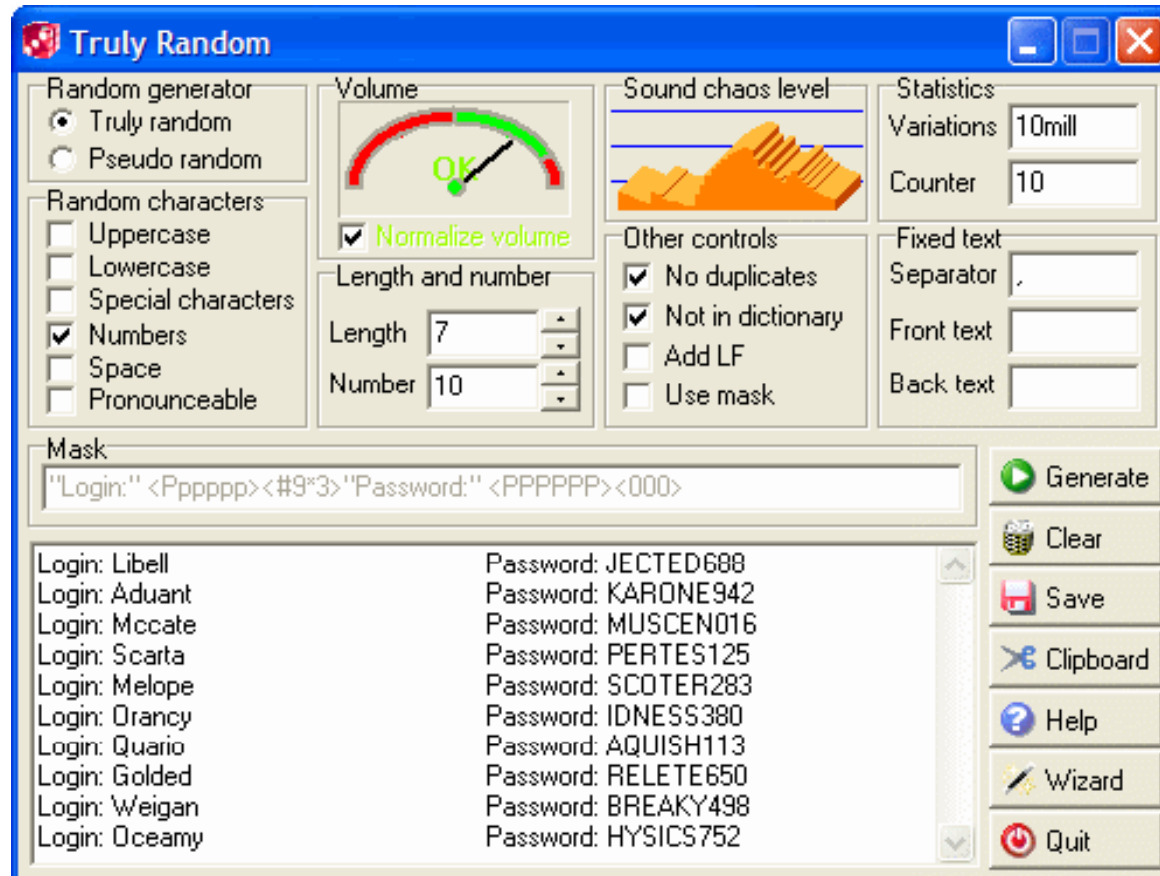
EDE on kolmekordne DES  
algoritm



$$R_i = EDE_{K_1, K_2} [ EDE_{K_1, K_2} [ Dt_i ] V_i ]$$

$$V_{i+1} = EDE_{K_1, K_2} [ EDE_{K_1, K_2} [ Dt_i ] R_i ]$$

# Pseudo-juhuarvude generaatorid



<http://www.lisisoft.com/download.php?id=86895>

# Pseudo-juhuarvude generaatorid

