Zufallsgenerator
= Achillesferse

1100000111101110111111111000011
01100101011001001111111110110
01101010001100100001110010000
10110100001111011110101110000
01011100010101100111000101111
11110000011010001010011101111
10011110101011001000000001000
11010101000100111000110011010
01100100110101 1

Bootvorgang, ID | Mail, Message-ID | Web-Browser | Spiel

danach: Kryptoschlüssel

Zufall

Unvorhersagbarkeit

Gegenteil von berechenbar

computationally infeasable to predict

Zufallsgeneratoren prüfen

statistical test suite (description)

csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

Hypothesis H_0: the sequence being tested is random

Zufallsgeneratoren prüfen

Frequency (Monobit) Test
Frequency Test within a Block
Runs Test
Test for the Longest Run of Ones in a Block
Binary Matrix Rank Test
Discrete Fourier Transform (Spectral) Test
Non-overlapping Template Matching Test
Overlapping Template Matching Test
Maurer's ,,Universal Statistical" Test
Linear Complexity Test
Serial Test
Approximate Entropy Test
Cumulative Sums (Cusum) Test
Random Excursions Test
Random Excursions Variant Test

Testergebnisse und Wirklichkeit

	Test OK	Test lehnt ab
PRNG OK (TRNG)	√	α
PRNG schlecht	β	√

Ziel α

Reduce probability α of false alarm (type I error)

Level of significance (of the test)

Crypto: $\alpha = 0.01$

Reject true random number generator

Only with probability 1%

Ziel β

Reduce probability α of false acceptance (type II error)

Fatal for Crypto:

Accept flawed random number generator

β difficult to determine, but …

Bounded by function of α and *n*

Hilfsmittel

- Gamma function

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx$$

$$\Gamma(n) = (n-1)!$$

$$\Gamma(a,x) = \int_x^\infty t^{a-1} e^{-t} dt$$

GSL-Library (GNU scientific library)

double gsl_sf_gamma_inc_Q (double a, double x)

$$\gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt$$

Hilfsmittel

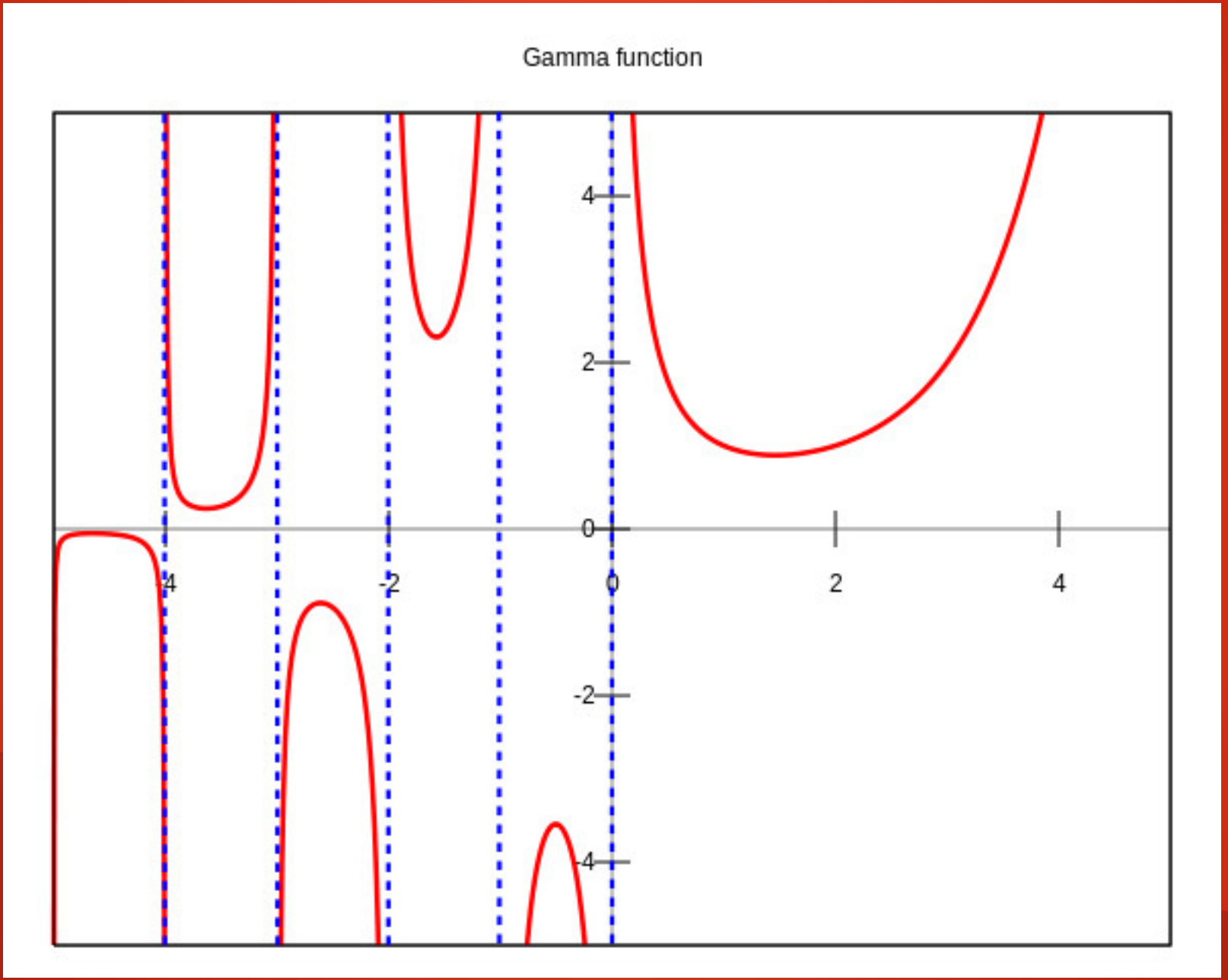
- (upper incomplete) Gamma function

$$\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$$

$$\Gamma(n) = (n-1)!$$

$$\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}dt$$

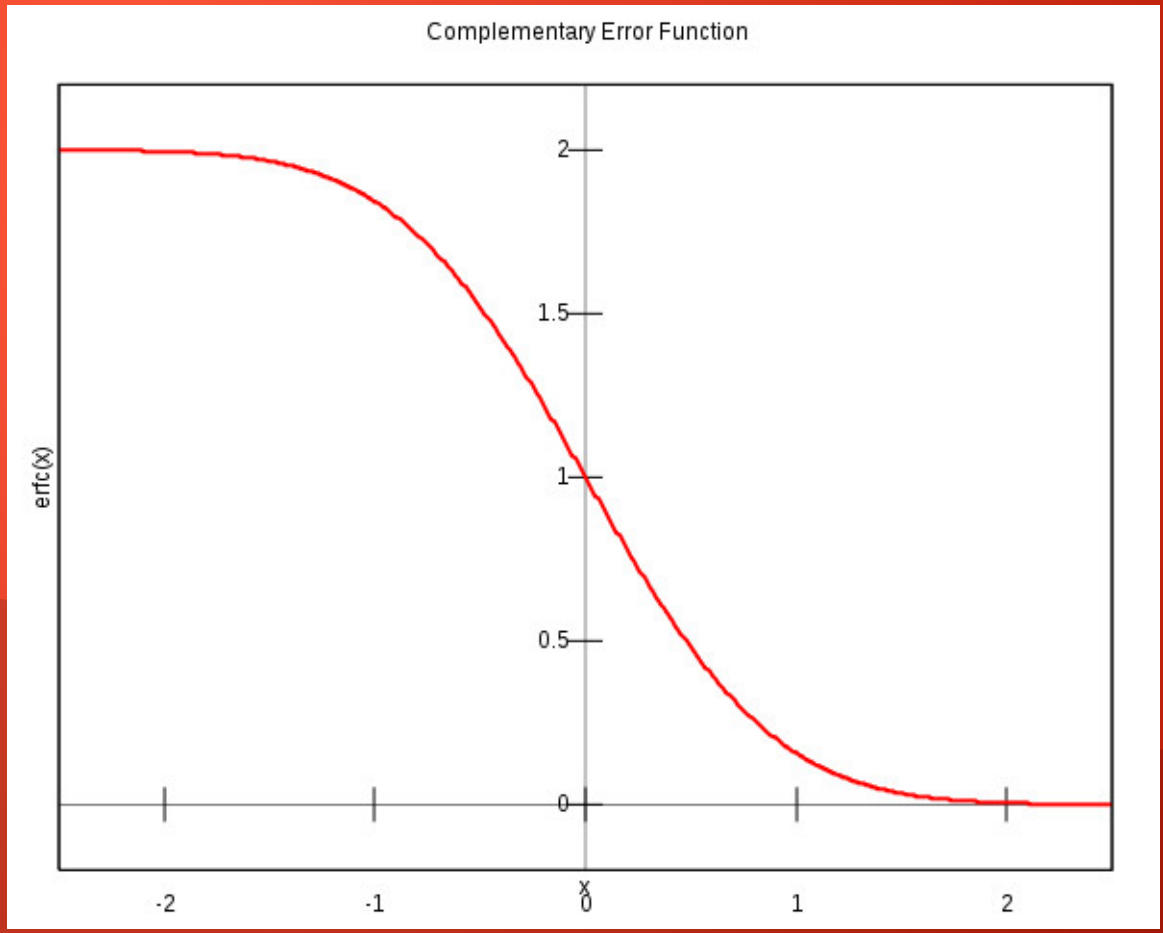
```
? gamma(5)
%2 = 24.000000000000000000000000000000000
? incgam(1,0)
%3 = 1.000000000000000000000000000000000000
? incgamc(1,0)
%4 = 0.E-38
```


Gamma function
4
2
0
-2
-4
-4
-2
0
2
4

Hilfsmittel

- Error function
  erfc(x)
- in C
- in Python

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2}\, dt$$



Normalverteilung:

Mittelwert,

Standardabweichung

prüft, ob Abweichung zu groß

χ^2-Test

- Aufteilung in Kategorien
- Stelle fest, ob Auftreten der Fälle zufällig
- Freiheitsgrade *k* : wieviele Fälle frei wählbar

$$d_i = \frac{(c_i - e_i)^2}{e_i}$$

$$\chi^2 = \sum_i d_i$$

$$P = \Gamma(k/2, \chi^2/2)$$

Monobit-Test

- f(bit)=2*bit-1, n = Anzahl der Bits
- Summe
- Mittelwert 0
- Abweichung von 0 mit Errorfunktion erfc()

$$s = \frac{|S|}{\sqrt{n}}$$

$$P = \operatorname{erfc}\left(\frac{s}{\sqrt{2}}\right)$$

Frequency in Block

- N Blöcke der Größe M
- N = n / M
- M > 0.01 * n > 20, N<100
- Prozentsatz von 1en in i-tem Block

$$\chi^2 = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2} \right)^2$$

$$P = \Gamma(N/2, \chi^2/2)$$

Runs

- 01111……10 ist ein 1-run
- wie oft aufeinanderfolgende Bits nicht identisch

$$V_n = \sum_{i=0}^{n-2} (\epsilon_i \neq \epsilon_{i+1})$$

$$P = \operatorname{erfc}\left(\frac{|V_n - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right)$$

Variante: blockweise, Blöcke der Länge 8, 128, 10000

Matrix Rank

- 32 x 32 Bits als Matrix
- Rang der Matrix mit prozentualer Erwartung
  - Voller Rang in 28% aller Fälle
  - Rang 31 in 58% der Fälle
  - Rang <31 in 13% der Fälle

- Drei Klassen → χ2-Test

$$P = e^{-\chi^2/2}$$

Fourier-Transformation

- f(b)=2*b-1

$$a_j = \sum_{k=0}^{n-1} f(\epsilon_k) \cdot \exp(2\pi i \cdot k \cdot j/n)$$

$$\exp(2\pi i \cdot k \cdot j/n) = \cos(2\pi k \cdot j/n) + i \sin(2\pi k \cdot j/n)$$

$$|a| = \sqrt{(c^2 + d^2)}.$$

Template Matching

- feste Schablone von Bits, z.B. 10111
- über die Folge gleiten lassen,
  wie oft ein Treffer?
- Überlappend / nicht-überlappend

Maurer Universal Entropy

- Wie gut lässt sich die Folge mit Komprimierungsverfahren (Ziv) behandeln?
- Wieviel Information enthält die Folge?

Linear Complexity

- Finde LFSR-Rekursionsformel für Block der Länge M
- Länge des LFSR, Erwartungswert

$$\mu = \frac{M}{2} + \frac{9 + (-1)^{M+1}}{36} - \frac{\frac{M}{3} + \frac{2}{9}}{2^M}$$

$$T_i = (-1)^M \cdot (L_i - \mu) + \frac{2}{9}$$

class k	condition	π_k
0	$T_i \leq -2.5$	0.010417
1	$-2.5 < T_i \leq -1.5$	0.031250
2	$-1.5 < T_i \leq -0.5$	0.125000
3	$-0.5 < T_i \leq +0.5$	0.500000
4	$+0.5 < T_i \leq +1.5$	0.250000
5	$+1.5 < T_i \leq +2.5$	0.062500
6	$+2.5 < T_i$	0.020833

Random Excursions

- f(b) = 2*b -1

- Addiere die f()-Werte, drei Tests
  - Wie groß ist der größte Abstand von 0?
  - Wie oft wird 0 durchquert?
  - Wie oft kommt Zustand s vor {-9,-8,...,-1,1,2,...,9}

Beispiel:
DUAL-EC DBRG

- bis 2006: keine Standards für Zufallsgeneratoren
- in 2006: Vorschlag mit 4 Algorithmen
- Shumow/Ferguson (Microsoft), CRYPTO 2007, mögliche NSA-Hintertür im DUAL-EC-Verfahren
- standardisierte Parameterwahl nicht erläutert

DUAL-EC (vereinfacht)

(1) a=2, b=3, p=13 feste Werte

(2) wähle einen Startwert x

(3) berechne $y = (a \cdot x) \bmod p$

(4) berechne $z = (b \cdot y) \bmod p$

(5) Zufallsbits: unterste 4 Bits von z

(6) setze x=y und gehe zu (3)

Zufallsfolge mit x=10 : 0, 3, 6, 4, 3, 1, 5, 2, 7, ...

DUAL-EC (vereinfacht)

(1) a=2, b=3, p=13 feste Werte

(2) wähle einen Startwert x x=10

(3) berechne y = (a · x) mod p $2 \cdot 10 = 20 \bmod 13 = 7$

(4) berechne z = (b · y) mod p $7 \cdot 3 = 21 \bmod 13 = 8$

(5) gib die untersten 3 Bits von z aus $8 = (1000)_2$

(6) setze x=y und gehe zu (3)

Beispiel: Startwert x=10

man muss nur raten ob hier eine 0 oder eine 1 steht

Zufallszahlen:

0, 3, 6, 4, 3, 1, 5, 2, 7, ...

$0 = (0000)_2$

DUAL-EC-Trick der NSA

1111101111011100 01110100000010110010010....10100111011110100111101 01

65536 Möglichkeiten

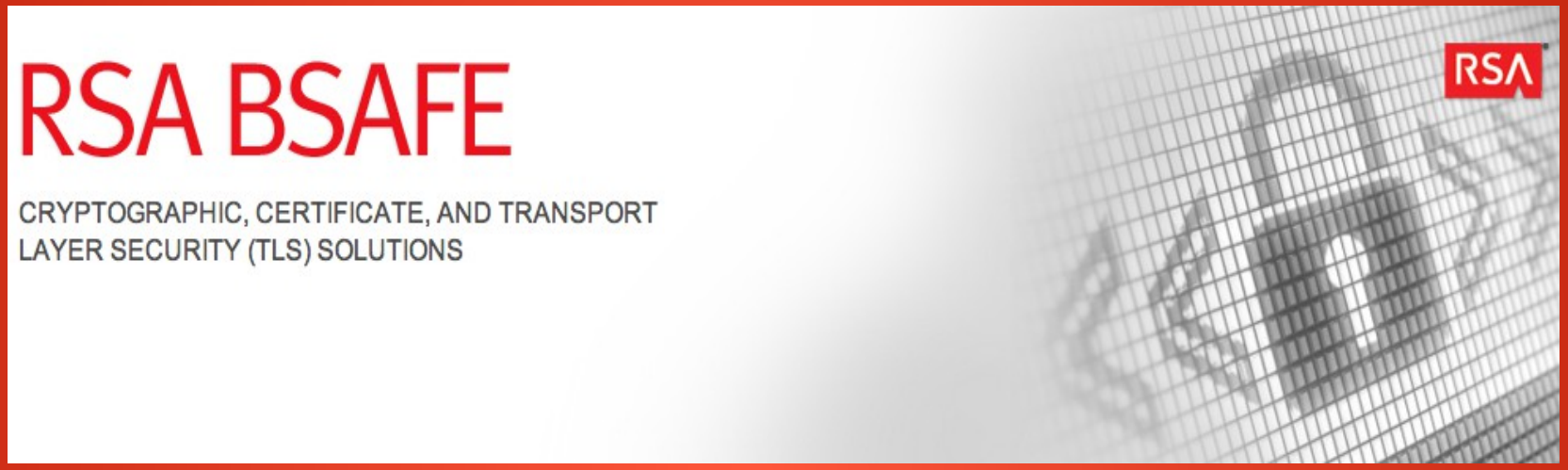
16 geheime Bits
im Zustand des
Zufallszahlenerzeugers

240 ausgegebene
Zufallsbits

(relativ) viele ausgegebenen Zufallsbits → Anfängerfehler

besser: viele Bits im geheimen Zustand → weniger Performance

Eine 10 Millionen $ Frage



- **NSA HQ:** würden Sie DUAL-EC als Voreinstellung in BSAFE einbauen?
- **RSA Inc:** Nein!
- **NSA HQ:** und wenn wir Ihnen 10 Millionen $ geben?
- **RSA Inc:** Okay.

www.reuters.com am 21.12.2013

BACKDOORED 2004-2013

Empfohlene PRNGs: CTR-DRBG

Output:

1. *K*: The new value for *Key*.
2. *V*: The new value for *V*.

CTR_DRBG_Update Process:

1. *temp* = *Null*.
2. While (**len** (*temp*) < *seedlen*) do
   - 2.1 $V = (V + 1) \bmod 2^{outlen}$.
   - 2.2 *output_block* = **Block_Encrypt** (*Key*, *V*).
   - 2.3 *temp* = *temp* || *ouput_block*.

Empfohlene PRNGs: Hash-DRBG

Hashgen Process:

1. $m = \left\lceil \frac{requested\_no\_of\_bits}{outlen} \right\rceil$.
2. $data = V$.
3. W = the *Null* string.
4. For $i = 1$ to m

   4.1 $w_i = \mathbf{Hash}(data)$.

   4.2 $W = W \| w_i$.

   4.3 $data = (data + 1) \bmod 2^{seedlen}$.
5. $returned\_bits$ = Leftmost ($requested\_no\_of\_bits$) bits of W.
6. Return $returned\_bits$.

Empfohlene PRNGs: HMAC-DRBG

1. $K = \mathbf{HMAC}\ (K, V \| 0x00 \| provided\_data)$.
2. $V = \mathbf{HMAC}\ (K, V)$.
3. If ($provided\_data = Null$), then return K and V.
4. $K = \mathbf{HMAC}\ (K, V \| 0x01 \| provided\_data)$.
5. $V = \mathbf{HMAC}\ (K, V)$.
6. Return K and V.