Thomas Risse

Die Güte von Pseudo-Zufallszahlen ist die Qualität ihrer Generatoren

Auszug. Die Sicherheit kryptographischer Verfahren basiert häufig auf der Güte der verwendeten Pseudo-Zufallszahlen, PRNs, und damit auf der Qualität der eingesetzten Zufallszahlen-Generatoren, PRNGs. Wir stellen hier einige PRNGs vor, diskutieren Sätze von Güte-Kriterien für Zufallszahlen, wie sie von NIST und BSI für die Zertifizierung verwendet werden, und wenden die BSI-Kriterien auf Zufallszahlen an, wie sie die Generatoren in MATLAB oder SAGE erzeugen. Überraschenderweise erfüllen die Zufallszahlen alle Kriterien, unabhängig davon, ob sie von alten 'legacy' PRNGs oder von modernen 'kryptographisch sicheren' PRNGs erzeugt wurden.

Einführung

Die Sicherheit vieler kryptographischer Verfahren basiert auf der Güte der verwendeten Zufallszahlen. Beispiele liefern die Verwendung in *block ciphers*, als RSA-moduli, in *keystream generation*, für Verfahren des *zero knowledge authentication* oder in PQC [19]. Zumeist handelt es sich um algorithmisch erzeugte Pseudo-Zufallszahlen, PRNs, und grundsätzlich geht es um 0-1-Folgen [9], [4], [21].
Damit ist Qualität der eingesetzten Zufallszahlen-Generatoren, PRNGs, Ausschlag-gebend, wie einige wenige aktuelle Beispiele belegen:

• ConsensusSecurityVulnerabilityAlert@sans.org 13/08/15: Flaw in Android random number generator leaves Bitcoin wallets open to theft. Users of such apps are encouraged to migrate away from insecure wallets through any feasible mechanism as soon as possible. `http://bitcoin.org/en/alert/2013-08-11-android`

• sans.org 13/08/31: 'Security Analysis of Pseudo-Random Number Generators with Input `/dev/random` is not Robust.' [5]

• c't 22/2013 S.44, wired: RSA Tells Its Developer Customers: Stop Using NSA-Linked Random Bit Generator

`http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-90-ARev1BandC`

Im nächsten Abschnitt stellen wir einige gängige PRNGs vor, wie sie auch Computer-Algebra-Systeme wie MATLAB `http://www.mathworks.de` oder SAGE `http://www.SAGEmath.org` bereitstellen. Im folgenden Abschnitt beschreiben und diskutieren wir Kriterien für die Güte, d.h. Zufälligkeit von Zufallszahlen. Zertifizierende Stellen wie das NIST [21] in den USA oder das BSI [2] in Deutschland haben Test Suites publiziert. Im letzten Ab-

schnitt wenden wir die Güte-Kriterien des BSI auf Zufallszahlen an, die verschiedene PRNGs von MATLAB bzw. SAGE erzeugen. Man wird nachdenklich, wenn die Güte-Kriterien nicht unterscheiden (können), ob Zufallszahlen von alten 'legacy' PRNGs oder von aktuellen 'kryptographisch sicheren' PRNGs erzeugt wurden.

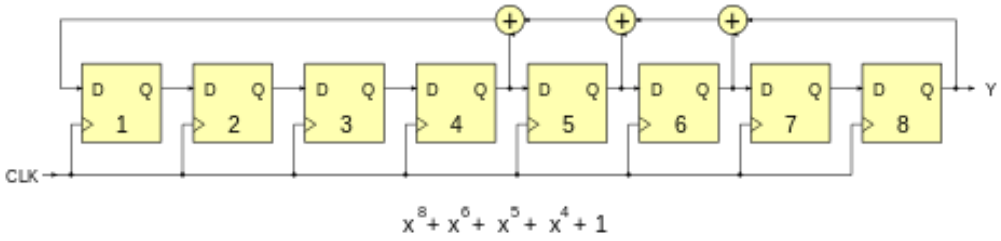
Einige Pseudo-Zufallszahlen-Generatoren

Pseudo oder *deterministische Zufallszahlen-Generatoren* erzeugen Zufallszahlen algorithmisch. Typisch ist eine Initialisierung eines internen Zustandes (seed) und die Berechnung der nächsten Zufallszahl ausgehend von der(den) letzten Zufallszahl(en).

- John von Neumann's middle-square method, 1946:
  wähle eine natürliche Zahl (seed), quadriere sie, entnimm dem Quadrat die mittleren Ziffern als Zufallszahl und verwende diese Zahl als seed für die nächste Iteration.

- Linear congruential generators [11]:
  Wähle seed x_o *und berechne* $x_{k+1} = a\,x_k + c \bmod m$
  für $k = 0, 1, \ldots$ *und geeignete Parameter* a, c *und* m

- Linear Feedback Shift Registers, LFSRs:
  Wähle einen Anfangszustand $\mathbf{x}$ *und laß die hardware, e.g.*



  arbeiten: im Zustand $\mathbf{x}$ *erzeugt das LFSR das nächste bit* y*, indem es* $y = p(\mathbf{x}) = x^8 + x^6 + x^5 + x^4 + 1$ *in* $\mathbb{GF}_2$ *berechnet.*

- Mersenne[1] twister algorithm [14]:
  Wähle $N = 624$ *seeds,* Y_1*, ...* Y_N*, berechne* Y_i *for* $i > N$ *durch*
  $h := Y_{i-N} - Y_{i-N} \bmod 2^{31} + Y_{i-N+1} \bmod 2^{31}$
  $Y_i := Y_{i-227} \oplus \lfloor h/2 \rfloor \oplus ((h \bmod 2) \cdot 9908b0df_{hex})$
  Nachbearbeitung garantiert Gleichverteilung:

[1] M. Mersenne (1588-1648) www-history.mcs.st-andrews.ac.uk/Biographies/Mersenne.html
untersuchte als Erster natürliche Zahlen der Form $2^n - 1$: einige sind prim! s.a. GIMPS

$x := Y_i \oplus \lfloor Y_i/2^{11} \rfloor$; $y := x \oplus ((x \cdot 2^7) \wedge 9d2c5680_{hex})$;
$z := y \oplus ((y \cdot 2^{15}) \wedge efc60000_{hex})$; $Z_i := z \oplus \lfloor z/2^{18} \rfloor$
mit Periodenlänge $2^{19937} - 1 \approx 4.3{\cdot}10^{6001}$ und Verbesserungen [17].

- Blum-Blum-Shub generator [1]:
  Wähle $n = pq$ für geeignete Primzahlen p und q, wähle seed y with $\gcd(y, n) = 1$, *berechne* $y_0 = y^2 \bmod n$ *und für* $i = 0, 1, ...$ *berechne* $y_{i+1} = y_i^2 \bmod n$.

Falls gewünscht, erzeugt man etwa per $b_i = y_i \bmod 2$ aus der Folge nicht negativer ganzer Zahlen (y_i) die 0-1-Folge (b_i).

Güte-Kriterien

Typischerweise sind Gütetests statistische Tests, die meist per χ^2-Test vergleichen, ob sich zu bewertende Zufallszahlen von den Zufallszahlen, die ein idealer Zufallszahlengenerator erzeugt, unterscheiden.
Es gibt eine Reihe einschlägiger 'akademischer' test suites, z.B. [16], *Diehard Battery of Tests*[2]oder *DieHarder: a random number test suite including an alternative GPL implementation of Diehard tests in C*[3] sowie test suites, die Zertifizierungsstellen wie NIST/FIPS [21] in den USA oder BSI [2] in Deuschland veröffentlichen. Im Folgenden setzen wir uns mit der aus acht Einzeltests bestehenden test suite des BSI auseinander.

1. Monobit Test

FIPS, HAC [16], NIST [21], BSI [8],[9] pp44 testen, ob 0 und 1 gleichverteilt sind.

Sei $n = 20000$, $T_1 = \sum_{i=1}^{n} b_i$. Die bit-Folge $\left(b_i\right)_{i=1}^{n}$ besteht den *Monobit Test* falls $9654 < T_1 < 10346$.

Die Test-Statistik T_1 ist binomial und näherungsweise $N(np, npq)$-verteilt. Dieses zweiseitige Kriterium entspricht einer Irrtumswahrscheinlichkeit von 10^{-6}, die das BSI zumeist – aber nicht immer – verwendet.

2. Poker Test

FIPS, HAC [16] 2bit, NIST [21], BSI [8],[9] pp46 testen, ob die aus je vier aufeinander folgenden Zufallszahlen gebildeten nibbles gleichverteilt sind.

[2] `http://www.stat.fsu.edu/pub/diehard/`
[3] `http://www.phy.duke.edu/~rgb/General/dieharder.php`

Sei $n = 20000$. Je vier bits bilden ein nibble. Sei $h_i := |\{j : 8b_{4j-3} + 4b_{4j-2} + 2b_{4j-1} + b_{4j} = i\}|$ und $T_2 = \frac{16}{5000}\sum_{i=o}^{15} h_i^2 - 5000$. Die bit-Folge $(b_i)_{i=1}^n$ besteht den *Poker Test*, falls $1.03 < T_2 < 57.4$.

Wegen $T_2 = \frac{16}{n_4}\sum_{i=o}^{15} h_i^2 - n_4 = \sum_{i=o}^{15}\frac{(h_i - n_4/16)^2}{n_4/16} \geq 0$ mit $n_4 = \frac{n}{4}$ ist T_2 χ^2-verteilt mit $df = 15$. Wir erwarten hier ein einseitiges Kriterium!

3. Run Tests

FIPS, HAC [16], NIST [21] #runs, BSI [8],[9] pp47 testen, ob die Anzahl der 0-runs and 1-runs so wie bei einem idealen RNG verteilt sind.

Sei $n = 20000$, sei k_ℓ die Anzahl der runs der Länge ℓ. Die bit-Folge $(b_i)_{i=1}^n$ besteht den *Run Tests*, falls $k_1 \in [2267, 2733]$, $k_2 \in [1079, 1421]$, $k_3 \in [502, 748]$, $k_4 \in [233, 402]$, $k_5 \in [90, 223]$ und $k_{\geq 6} \in [90, 2^2_3 3]$.[4]

0-runs oder 1-runs der Länge ℓ treten mit $p_{run} = \frac{1}{2^{\ell+2}}$ an jeder der $n - \ell - 1$ inneren Stellen auf und mit $p'_{run} = 2p_{run}$ an den beiden Rändern, so daß $E(K_\ell) = \frac{n-\ell-1+2+2}{2^{\ell+2}} = \frac{n-\ell+3}{2^{\ell+2}}$. Für $b \in \{0, 1\}$ ist dann $T_{3_{\ell_b}} = \sum_{i=1}^{\ell}\frac{(k_i^{(b)} - E(K_i))^2}{E(K_i)}$ näherungsweise χ^2-verteilt. Wieder gibt es eine Reihe von Fragwürdigkeiten: z.B. [16] unterstellt $df = 2\ell - 2$, [15] behauptet $df = 2\ell$ und BSI [8],[9] geht von $df = 2\ell - 1$ aus.

4. Longrun Test

FIPS, NIST [21] longest/block, BSI [8],[9] p49 testen, ob nicht etwa 0-runs oder 1-runs mit Längen größer 33 vorkommen.

Sei $n = 20000$. Die bit-Folge $(b_i)_{i=1}^n$ besteht den *Longrun Test*, falls $k_\ell = 0$ für alle $\ell \geq 34$.

Übrigens gilt $P(k_\ell = 0) = \frac{F^{(\ell)}_{n+2}}{2^n}$ mit den Fibonacci ℓ-Schritt Zahlen [24] $F^{(\ell)}_k = \sum_{i=1}^{\ell} F^{(\ell)}_{k-i}$ mit $F^{(\ell)}_k = 0$ für $k \leq 0$ und $F^{(\ell)}_1 = F^{(\ell)}_2 = 1$, so daß etwa SAGE [23] $P(k_\ell = 0)$ leicht exakt berechnet.

5. Autocorrelation Test

HAC [16], BSI [8],[9] pp49 testen die Autokorrelation der 0-1-Folge.

Sei $n = 20000$ und $T_{5_\tau} = \sum_{j=1}^{n/4} b_j \oplus b_{j+\tau}$ for $\tau \in \{1, 2, ..., \frac{n}{4}\}$. Die bit-Folge $(b_i)_{i=1}^n$ besteht den *Autocorrelation Test*, falls $|T_{5_\tau} - \frac{n}{8}| < 174$ für alle τ.

[4] BSI verwendet 223 in älteren und 233 in neueren Dokumenten.

Warum ist hier nur die erste Hälfte der (b_i) relevant? Das Kriterium ist ansonsten konsistent mit dem Umstand, daß T_{5_τ} näherungsweise $N(\frac{n}{4}\,\frac{1}{2}, \frac{n}{4}\,\frac{1}{2}\,\frac{1}{2})$-verteilt ist.

6. Uniform Distribution Test

HAC 2bit, NIST $^{\text{template}}_{\text{matching}}$, BSI [8],[9] pp50 testen, ob bit-Muster der Länge k mit der Häufigkeit 2^{-k} vorkommen.

Erzeuge $\mathbf{w}_j \in \{0,1\}^k$ aus $\big(b_i\big)_{i=1}^{nk}$; $T_{6_\mathbf{x}} := \frac{|\{j:\mathbf{w}_j=\mathbf{x}\}|}{n}$ ist die relative Häufigkeit von $\mathbf{x}$. Die bit-Folge $\big(b_i\big)_{i=1}^{nk}$ besteht den *Uniform distribution Test* für Parameter k, n und α, falls $\left|T_{6_\mathbf{x}} - 2^{-k}\right| < \alpha$ für alle $\mathbf{x} \in \{0,1\}^k$.

Das BSI spezifiziert hier eine ganze Klasse von Gleichverteilungstests, die im Übrigen monobit tests verallgemeinern. Überraschenderweise nutzt das BSI diesen Umstand garnicht, wenn es in *Test Procedure B* [8],[9] p55 vorgibt: T_6 *has to be run with* $k = 1$, $n = 10^5$ *and* $\alpha = 0.025$. und auf p51 ausdrücklich: $(b_i)_{i=1}^n$ *passes if* $|T_{6_o} - \frac{1}{2}| < \alpha$.

7. Multinomial/Homogeneity Test

BSI [8],[9] pp51 testet, ob das Auftreten einer 0 oder einer 1 von den vorangegangen Zufallszahlen (un-) abhängig ist.

Erzeuge $\mathbf{w}_{i,j} \in \{0,1,...,s-1\}$ für $^{i=1,..,h}_{j=1,..,n}$ aus $\big(b_k\big)_k$, d.h. führe h unabhängige Wiederholungen des j-ten Experimentes durch. Sei $f_i(t) = |\{j : w_{ij} = t\}|$ und $p_t = \frac{1}{hn}\sum_{i=1}^h f_i(t)$. Die bit-Folge $\big(b_i\big)_i$ besteht den *Multinomial Test* für Parameter h, s, n und α, falls $T_7 \le \chi^2(\alpha,(h-1)(s-1))$ wobei $T_7 = \sum_{i=1}^h \sum_{t=o}^{s-1} \frac{(f_i(t)-n\,p_t)^2}{n\,p_t}$.

T_7 ist näherungsweise χ^2-verteilt mit $df = (h-1)(s-1)$. Wie T_6 spezifiziert auch T_7 eine ganze Klasse von Tests. BSI [8],[9] pp55 *Test Procedure B* verlangt nun drei spezifische Tests:

1. step 2 bits
   Extrahiere $\mathrm{TF}_r = \{(b_{2j+1}, b_{2j+2}) : b_{2j+1} = r\}$ *mit* $|\mathrm{TF}_0| = |\mathrm{TF}_1| = n_1 = 10^5$ *von* (b_i). *Bestimme die empirischen Übergangswahrscheinlichkeiten* $v_r(i) := |\{j : (b_{2j+1}, i) \in \mathrm{TF}_r\}|/n_1$. *Die 0-1-Folge besteht den Test* T_7, *falls* $|v_0(1) + v_1(0) - 1| < \alpha_1 = 0.02$.

2. step 3 bits – vermutlich $h = 2$ (oder $h = 4$?) und $s = 2$
   Extrahiere $\mathrm{TF}_{\mathtt{rs}} = \{(b_{3j+1},..,b_{3j+3}) : (b_{3j+1}, b_{3j+2}) = (\mathtt{rs})\}$ *mit*

$|\mathrm{TF}_{\mathtt{oo}}| = |\mathrm{TF}_{\mathtt{o1}}| = |\mathrm{TF}_{\mathtt{1o}}| = |\mathrm{TF}_{\mathtt{11}}| = n_2 = 10^5$ *von* (b_i). *Bestimme die empirischen Übergangswahrscheinlichkeiten* $v_{\mathtt{rs}}(\mathtt{i}) := |\{j : (b_{3j+1}, .., b_{3j+3}) = (\mathtt{rsi})\}|/n_2$.
'for each $\mathtt{s} \in \{0,1\}$ *then compare* $v_{\mathtt{0s}}$ *and* $v_{\mathtt{1s}}$ *with test* T_7 *at* $\alpha_2 = 0.0001$*'*.

3. step 4 bits – vermutlich $h = 3$ (oder $h{=}8$?) und $s = 2$
   Extrahiere $\mathrm{TF}_{\mathtt{rst}} = \{(b_{4j+1}, ..., b_{4j+4}) : (b_{4j+1}, .., b_{4j+3}) = (\mathtt{rst})\}$ *mit* $|\mathrm{TF}_{\mathtt{ooo}}| = |\mathrm{TF}_{\mathtt{oo1}}| = \ldots = |\mathrm{TF}_{\mathtt{111}}| = n_3 = 10^5$ *von* (b_i). *Bestimme die empirischen Übergangswahrscheinlichkeiten* $v_{\mathtt{rst}}(i) := |\{j : (b_{4j+1}, .., b_{4j+4}) = (\mathtt{rsti})\}|/n_3$.
   'for each $(\mathtt{s},\mathtt{t}) \in \{0,1\}^2$ *then compare* $v_{\mathtt{0st}}$ *and* $v_{\mathtt{1st}}$ *with test* T_7 *at* $\alpha_3 = 0.0001$*'*

Hier stellt das BSI offensichtlich unterschiedliche Anforderungen an die (Un-) Abhängigkeit des aktuellen bits von dem Vorgänger, von den beiden bzw. von den drei Vorgängern.

8. Entropy Test

HAC [16], NIST $^{\text{approx. entropy}}$, BSI [8],[9] pp52 $^{\text{in Übereinstimmung mit}}$ [3],[4] testen die Entropie der 0-1-Folge.

Erzeuge $\mathbf{w}_n \in \{0,1\}^L$ aus $\big(b_i\big)_{i=1}^{(Q+K)L}$. Sei A_n der Abstand von w_n zu einer identischen Vorgänger-Teilfolge, d.h. $A_n = \begin{cases} n & \text{falls es kein } i \geq 1 \text{ gibt mit } \mathbf{w}_n = \mathbf{w}_{n-i} \\ \min\{i \geq 1 : \mathbf{w}_n = \mathbf{w}_{n-i}\} & \text{sonst} \end{cases}$ Sei $T_8 = \frac{1}{K}\sum_{n=Q+1}^{Q+K} g(A_n)$ mit $g(i) = \frac{1}{\log 2}\sum_{k=1}^{i-1}\frac{1}{k} \approx \frac{\log i + \gamma + \frac{1}{2i} + \frac{1}{12i^2}}{\log 2} + \mathcal{O}(\frac{1}{i^4})$ und der Euler Konstanten $\gamma \approx 0.577216$. Die bit-Folge $\big(b_i\big)_{i=1}^{(Q+K)L}$ besteht den *Entropy Test*, falls T_8 näherungsweise $N(\mu, \sigma^2)$-verteilt ist mit 'tabellierten' $\mu = \mu(L,K)$ und $\sigma = \sigma(L,K)$.

Wie schon T_6 und T_7 so gibt auch T_8 nur den Rahmen und keine Parameter für den Anpassungstest vor. BSI [8],[9] pp55 *Test Procedure B* spezifiziert jedoch: $\big(b_i\big)_{i=1}^{n}$ besteht T_8 mit $L = 8$, $Q = 10 \cdot 2^L = 2560$, $K = 1000 \cdot 2^L = 256000$, $\mu = L$, und $\sigma = c(L,K)\sqrt{\mathrm{Var}(g(A_n))/K}$, falls $T_8 > 7.976$.
Wir halten fest: T_8 basiert auf Maurer's *universal test* [15] sowie seiner Korrektur und Verallgemeinerung durch Coron&Naccache [3] und Coron [4]. Insofern irritiert, daß der BSI-Test ein-seitig und die drei originalen Tests [15],[3],[4] zwei-seitig sind.

BSI Kriterien auf MATLABs und SAGEs Zufallszahlen anwenden

Wir wenden die BSI test suite auf 0-1-Folgen an, die von den diversen PRNGs erzeugt wurden, die die verschiedene MATLAB-Versionen 7.0, 7.3, 7.11, 7.13 und 8.1 bzw. SAGE 5.4.1 vorhalten: *jeder* PRNG, also auch schwache 'legacy' PRNGs besteht die BSI test suite (Test Code [20]).

BSI-Tests	MATLAB					SAGE[5]
	7.0	7.3	7.11	7.13	8.1	5.4.1
Monobit Test	√	√	√	√	√	√
Poker Test	√	√	√	√	√	√
Run Tests	√	√	√	√	√	√
Longrun Test	√	√	√	√	√	√
Autocorrelation Test	√	√	√	√	√	√
Uniform Distribution Test	√	√	√	√	√	√
Homogeneity Test[6]	√	√	√	√	√	√
Entropy Test	√	√	√	√	√	√

Das Ergebnis ist insofern bemerkenswert, als auch schwache 'legacy' PRNGs die BSI test suite bestehen.

Zusammenfassung

In jeder der getesteten Versionen, hält MATLAB 'legacy' und jeweils aktuelle PRNGs vor.

\+ Man kann PRNGs sehr einfach verwenden. Optional kann man spezielle Generatoren auswählen.

\+ Wie üblich, werden die PRNGs randomisiert, indem seed anhand der System-Zeit bestimmt wird.

– Von Version 7.0 bis zu Version 8.1 haben die MATLAB-Entwickler die PRNGs sowie die Art und Weise, diese auszuwählen und zu benutzen, dauernd und nicht immer konsistent verändert, vgl. Code [20].

Im Vergleich zu MATLAB ist SAGE wirklich Objekt-orientiert und bietet Methoden zur zufälligen Erzeugung von Objekten vieler verschiedener Klassen. Wir haben hier nur den PRNG zur Erzeugung von zufälligen ganzen Zahlen, speziell eben 0-1-Folgen getestet und lassen damit das riesige Potential von SAGE, etwa zufällige Graphen o.ä. zu erzeugen, außer Acht.

[5] per `ZZ.random_element(x=0, y=2, distribution="uniform")`

[5] mit einiger Interpretation der ziemlich problematischen Vorgaben in [9] pp50/51 und pp55

Die mangelhafte Unterscheidungskraft der BSI test suite provoziert grundsätzlichen Fragen:

? Auf welchen Kriterien beruht der Einschluß/Ausschluß von Tests? Über die acht BSI-Tests hinaus sieht nämlich etwa das NIST [21] weitere Tests vor, die auf dem Rang binärer Matrizen, auf der diskreten Fourier-Transformation, auf template matching, auf der linearen Komplexität, auf random walks etc.

? Warum benützt das BSI nicht in allen Tests durchgängig χ^2-Tests?

? Nach welchen Kriterien bestimmt das BSI die unterschiedlichen (!) Irrtumswahrscheinlichkeiten?

? Was verhindet, daß ein PRNG immer wieder dieselben (Pseudo-) Zufallszahlen erzeugt, die die BSI-Tests einmal bestanden haben?

Zudem fehlt den BSI-Dokumenten [8] und [9] in [2] eine errata-Sammlung, wie sie etwa [7] für das NIST [21] darstellt.
Wünschenwert ist ein konsistentes, Fehler-freies framework für Zufälligkeit, aus dem man für vorgegebene 'globale' Irrtumswahrscheinlichkeit eine 'vollständige' Test-Suite ableiten kann. Leider kann m.E. ein solches framework aber – höchstwahrscheinlich oder fast sicher – nicht existieren.

Literaturverzeichnis

[1] Lenore Blum, Manuel Blum, Michael Shub: A Simple Unpredictable Pseudo-Random Number Generator; SIAM Journal on Computing, Vol 15, Nr. 2, 364-383, May 1986

[2] BSI: Anwendungshinweise und Interpretationen (zum Schema), AIS; `www.bsi.bund.de/DE/Themen/ZertifizierungundAnerkennung/ZertifizierungnachCCundITSEC/AnwendungshinweiseundInterpretationen/AIS/aiscc_node.html`

[3] Jean-Sebastien Coron, David Naccache: An Accurate Evaluation of Maurers Universal Test; Proc. of SAC'98; Springer LNCS 1998, `http://www.jscoron.fr/publications/universal.pdf`

[4] Jean-Sebastien Coron: On the Security of Random Sources; in H. Imai, Y. Zheng, Eds.: Public-Key Cryptography; LNCS vol. 1560, 29-42, Springer 1999, `http://www.jscoron.fr/publications/entropy.pdf`

[5] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, Daniel Wichs: Security Analysis of Pseudo-Random Number Generators with Input: `/dev/random` is not Robust; ACM Conference on Computer and Communication Security, CCS, November 2013, `http://eprint.iacr.org/2013/338.pdf`

[6] P. L'Ecuyer, R. Simard, E.J. Chen, W.D. Kelton: An Objected-Oriented Random-Number Package with Many Long Streams and Substreams; Operations Research, 50(6):1073-1075, 2002

[7] Charmaine Kenny: Random Number Generators – An Evaluation and Comparison of Random.org and Some Commonly Used Generators; Trinity College Dublin, April 2005, `http://www.random.org/analysis/Analysis2005.pdf`

[8] Wolfgang Killmann, Werner Schindler: Functionality Classes and Evaluation Methodology for Random Number Generators; s. [2] `AIS20_Functionality_classes_for_random_number_generators.pdf`

[9] Wolfgang Killmann, Werner Schindler: Functionality Classes and Evaluation Methodology for Random Number Generators; 2011, s. [2] AIS31_Functionality_classes_for_random_number_generators.pdf

[10] Wolfgang Killmann, Werner Schindler: Functionality Classes and Evaluation Methodology for True (Physical) Random Number Generators; version 3.1, 2001, s. [2] AIS_31_Functionality_classes_evaluation_methodology_for_true_RNG_e.pdf

[11] Derrick H. Lehmer: Mathematical methods in large-scale computing units; Ann. Computing Lab., Harvard Univ. 26 (1951), 141-146

[12] George Marsaglia: Random numbers fall mainly in the planes; Proceedings of the National Academy of Sciences, 61 (1968), 25-28.

[13] George Marsaglia and A. Zaman: A new class of random number generators, Annals of Applied Probability, 3 (1991), 462-480

[14] M. Matsumoto, T. Nishimura: Mersenne Twister – A 623-dimensionally equidistributed uniform pseudorandom number generator; ACM Trans. Modeling and Computer Simulation Vol. 8, No. 1, January 1998, 3-30 http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/mt.pdf

[15] Ueli M. Maurer: A Universal Statistical Test for Random Bit Generators; Journal of Cryptology, vol. 5, no. 2, 1992, 89-105 ftp://ftp.inf.ethz.ch/pub/crypto/publications/Maurer92a.pdf

[16] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: Handbook of Applied Cryptography; CRC Press, October 1996 http://cacr.uwaterloo.ca/hac/

[17] François Panneton, Pierre L'Ecuyer, Makoto Matsumoto: Improved Long-Period Generators Based on Linear Recurrences Modulo 2; ACM Transactions on Mathematical Software, Vol. 32, No. 1, 2006, 1-16 ir.lib.hiroshima-u.ac.jp/metadb/up/81936204/ACMTraMath_32_1.pdf

[18] S.K. Park, K.W. Miller: Random number generators – Good ones are hard to find; Communications of the ACM, 31 (1988), 1192-1201 http://www.cems.uwe.ac.uk/~irjohnso/coursenotes/ufeen8-15-m/p1192-parkmiller.pdf

[19] Thomas Risse: How SAGE helps to implement Goppa Codes and the McEliece Public Key Crypto System; Ubiquitous Computing and Communication Journal, UbiCC, ISSN 1992-8424, Special Issue on 5th International Conference on Information Technology (ICIT'11) http://www.weblearn.hs-bremen.de/risse/papers/UbiCC11/

[20] Thomas Risse: Güte von Zufallszahlen – Qualität von Zufallszahlen-Generatoren; 11. workshop Mathematik für Ingenieure; Hochschule Bochum 30.9.2013 http://www.weblearn.hs-bremen.de/risse/papers/MathEng11

[21] Andrew Rukhin et al: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications; National Institute of Standards and Technology, NIST April 2010 http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

[22] Werner Schindler: Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators; BSI, version 2.0, 1999 AIS_20_Functionality_Classes_Evaluation_Methodology_DRNG_e.pdf

[23] William A. Stein et al.: SAGE Mathematics Software – System for Algebraic and Geometric Experimentation; www.SAGEmath.org

[24] Eric W. Weisstein: Run; MathWorld – A Wolfram Web Resource http://mathworld.wolfram.com/Run.html

Prof. Dr. rer. nat. Thomas Risse
Fakultät Elektrotechnik & Informatik
Hochschule Bremen, University of Applied Sciences
Flughafenallee 10
D-28199 Bremen
E-Mail: risse@hs-bremen.de