Comparative Analysis of Random Generators

Johannes vom Dorp, Joachim von zur Gathen, Daniel Loebenberger, Jan Lühr, and Simon Schneider

Dedicated to Peter Paule on his 60th birthday
Cogito ergo summo

1 Introduction

Randomness is an indispensable tool in computer algebra. Even for the basic and apparently simple task of factoring univariate polynomials over finite fields the only known efficient (= polynomial-time) algorithms are probabilistic, and finding a deterministic solution is the central theoretical problem in that area. For many, but not all, tasks of computational linear algebra the most efficient algorithms today use pre- and post-multiplication by random matrices, as introduced in Borodin et al. (1982) and refined in many ways since then; it is now a staple tool in that field.

Even greater is the importance in cryptography, say for generating all kinds of secret keys. Deterministic or predictable keys would allow an adversary to reproduce them and break the cryptosystem. Since the random keys are only known to the legitimate user, a brute-force attack would require an exhaustive search of a key space that is prohibitively large, thus preventing a feasible or practical search.

Now a fundamental problem is that we treat our computers as deterministic entities that, by their nature, cannot generate randomness. This is not literally true

J. vom Dorp
Fraunhofer FKIE, Bonn, Germany
e-mail: johannes.vom.dorp@fkie.fraunhofer.de

J. von zur Gathen (✉) · J. Lühr · S. Schneider
Bonn-Aachen International Center for Information Technology, Universität Bonn, Bonn, Germany
e-mail: gathen@bit.uni-bonn.de; luehr@cs.uni-bonn.de; schneider@cs.uni-bonn.de

D. Loebenberger
OTH Amberg-Weiden and Fraunhofer AISEC, Weiden, Germany
e-mail: d.loebenberger@oth-aw.de; daniel.loebenberger@aisec.fraunhofer.de

V. Pillwein, C. Schneider (eds.), *Algorithmic Combinatorics: Enumerative Combinatorics, Special Functions and Computer Algebra*, Texts & Monographs in Symbolic Computation, https://doi.org/10.1007/978-3-030-44559-1_10

because tiny random influences may come from effects like cosmic radiation, but these are easily controlled by error-correcting measures. Furthermore, quantum computers provide randomness naturally. Even more, they can factor integers in polynomial time and break most of the classical cryptosystems, say RSA, due to the famous algorithm by Shor (1999). But it is a matter of opinion whether or when scalable quantum computing will become a reality. Some central problems are described in Dyakonov (2018) and Clarke (2019), with a professionally optimistic view in the latter article.

How can we deal with this basic impossibility to generate randomness on our computers? After all, we do want secure internet connections and much more. A common solution works in two steps:

1. Produce values that are supposed to carry a reasonable amount of randomness, using an outside source, say measuring some physical process that looks chaotic to us.
2. Extend a small amount of true randomness to an arbitrarily large amount of *pseudorandomness*.

And what does that mean? True randomness refers to *uniform randomness*. A uniformly random source with values in a finite set produces each element of the set with the same probability. A pseudorandom source, usually called a *pseudorandom generator*, produces values that cannot be distinguished efficiently from uniformly random ones. That is, no efficient (polynomial-time) machine, deterministic or probabilistic, exists which can ask for an arbitrarily long stream of values, is given either a uniformly random stream or a stream generated by the pseudorandom generator, and then decides (with non-negligible probability of correctness) which of the two is the case.

Given a generator claimed to be pseudorandom, how do you prove that no such distinguishing machine exists? Unfortunately, we cannot, and there is no proof of any "provable security" in sight. The difficulty is embodied in the question $P \neq NP$ posed by Cook (1971) and, almost half a century later, is still an open one-million-dollar *millennium problem*. But computational complexity offers a solution: reductions. We take some algorithmic problem which is considered to be hard (not solvable in (random) polynomial time) and show that the existence of an efficient distinguisher implies a solution to the problem. A well-known such problem is the factorization of large integers. Many researchers have looked at it and no solution is known (except on the as yet hypothetical quantum computers). Such a reduction is currently the best way of establishing pseudorandomness.

Probability theory suggest a different approach: measure the entropy. It expresses the "amount" of randomness that a source produces. Unfortunately, entropy cannot be measured practically (Goldreich et al. 1999; von zur Gathen and Loebenberger 2018). As a way out, sometimes the *block entropy* is measured, see below. It will show large statistical abnormalities, if present, within the output stream, but cannot indicate their absence. In our context, this is rather useless, since even cryptographically weak generators may possess high block entropy.

An intermediate step before seeding a pseudorandom generator from a source is *randomness extraction*. Some of the methods in that area only require a lower bound on the source's *min-entropy*, a more intuitive measure for randomness. By their nature, physical random generators are not amenable to mathematically rigorous proofs of such bounds. Quite justifiably, reasonable engineering standards ignore such theoretical stipulations in practice, but we give some weight to them.

For physical hardware generators, applying a series of statistical tests like the above seems to be the only approach, and we also use it for lack of alternatives. For instance, lack of sufficient entropy caused severe weakening of Debian's OpenSSL implementation, see Schneier (2008). However, experts know the dangers of this approach quite well:

> The main part of a security evaluation considers the generic design and its implementation. The central goal is to quantify (at least a lower bound for) the entropy per random bit. Unfortunately, entropy cannot be measured as voltage or temperature. Instead, entropy is a property of random variables and not of observed realizations (here: random numbers). (Killmann and Schindler 2008)

This warning is often ignored in the literature.

Pseudorandom generators come in two flavors: based on a symmetric cryptosystem like the Advanced Encryption Standard (AES), or based on number-theoretic hard problems such as factoring integers. The general wisdom is that the latter are much slower than the former. The main goal of this paper is to examine this opinion which, to our surprise, turns out to be untenable.

We study one hardware generator; by its nature, it is out of scope for theoretical comparisons. Among the software pseudorandom generators, AES and some of the number-theoretic ones perform roughly equally well, provided they are run with *fair* implementations. We use corresponding home-brew code to run them, implemented with the same care. However, if the AES generator is run on specialized *AES-friendly* hardware, it outperforms the others by a large distance. This comes as no surprise.

Our comparative analysis covers some popular pseudorandom generators and two physical sources of randomness. Of course, the choice of possible generators is vast. We thus try to select examples of the respective classes to get a representative picture of the whole situation. Our measurements were reported in Burlakov et al. (2015a), so that their absolute values are somewhat outdated. But that is not the point here: we strive for a fair comparison of the generators, and that can be expected to carry through to later hardware versions, with a grain of salt.

An example of the insatiable thirst for randomness are TLS transactions, which consume at 43.000 new transactions per second (cipher suite ECDHE-ECDSA-AES256-GCM-SHA384) on a single Intel Xeon based system (cf. NGINX 2016 product information) 1376 KB/s of randomness to generate pre-master secrets of 256 bits each—ideally, using only negligible CPU resources.

In our setup we use as a source of random seeds one particular output of the hardware generator PRG310-4, which was analyzed in Schindler and Killmann (2003). On the software side we discuss several number-theoretic generators, namely the linear congruential generator, the RSA generator, and the Blum–Blum–Shub generator, all at carefully selected truncation rates of the output. The generators come with certain security reductions. For comparison we add to our analysis pseudorandom generators based on a well-studied block cipher, in our case AES in counter mode.

The article is structured as follows: We first present previous work on generator analysis in Sect. 2 before giving a detailed overview of the generators in Sect. 3. The main contribution is the evaluation regarding throughput and entropy consumption in Sect. 4. We conclude and elaborate on future work in Sect. 5.

All algorithms except the one employing AES-NI were implemented in a textbook manner using non-optimized C-code, thus providing a fair comparison. The source code of the algorithms is available at Burlakov et al. (2015b).

2 Related Work

Concerning physical generators, Killmann and Schindler (2008) analyze noisy diodes as a random source, providing a model for its entropy. One example of a noisy diodes based generator is the commercial generator PRG310-4, which is distributed by Bergmann (2019). Concerning non-physical true random generators, Linux' VirtIO generator as used in `/dev/random` is illustrated by Gutterman et al. (2006) and explained by Lacharme et al. (2012). Combined, they provide a clear picture of its inner workings. Additionally, there is the study by Müller (2019) in which the quality of `/dev/random` and `/dev/urandom` is studied with respect to the functionality classes for random generators as given by Killmann and Schindler (2011).

Referring to pseudorandom generators, the RSA based generator is explained in Shamir (1983), Fischlin and Schnorr (2000), and Steinfeld et al. (2006). Its cryptographic security is shown in Alexi et al. (1988) and extended in Steinfeld et al. (2006). Linear congruential generators were first proposed by Lehmer (1951). Attacks were discussed in Plumstead (1982) and Håstad and Shamir (1985). They all exploited its simple linear structure and come with a specific parameterization. Not all parameterizations—such as truncating its output to a single bit—have been attacked successfully as of today. Contini and Shparlinski (2005) analyze this in depth concluding that (for some cases)

> [. . .] we do not know if the truncated linear congruential generator can still be cryptanalyzed.

Blum et al. (1986) introduced the Blum–Blum–Shub generator. Alexi et al. (1988) and Fischlin and Schnorr (2000) show that the integer modulus can be factored, given a distinguisher for the generator.

A totally different approach for the construction of pseudorandom generators are the ones based on established cryptographic primitives. NIST (2015) specifies several standards for producing cryptographically secure random numbers. Besides hash-based techniques, there is also a standard employing a block cipher in counter mode, see also NIST (2001b) for this purpose.

RFC 4086, see Eastlake et al. (2005), compares different techniques and provides a de-facto standard focussing on internet engineering. There, several entropy pool techniques and randomness generation procedures are specified. However, RFC 4086 lacks recommendations for the ciphers to be used in OFB (output feedback) and CTR (counter mode) generation. We show here that such a general recommendation would also be ill-suited since the optimal choice depends heavily on the platform used.

We are not aware of any comprehensive fair benchmarking survey for all the generators mentioned above that integrates them into the Linux operating system.

3 The Generators

In the following, each generator which was implemented or applied for the comparative analysis is briefly presented. The output of a pseudorandom generator is, by definition, not efficiently distinguishable from uniform randomness, see for example Goldreich (2001). When assuming that certain problems in algorithmic number theory (such as factoring integers) are difficult to solve, the Blum–Blum–Shub, and RSA generators with suitable truncation have this property, but the linear congruential generator does not. Also the AES-based generator does not, but assuming AES to be a secure cipher, the AES-based generator is pseudorandom as well.

3.1 Linux `/dev/random` and `/dev/urandom`

The German Federal Office for Information Security[1] sets cryptographic standards in Germany and judges `/dev/random` to be a non-physical true random number generator (i.e., an NTG.1 generator in the terminology of Killmann and Schindler 2011) for most Linux kernel versions, see BSI (2019b).

`/dev/urandom`, however, does not fulfill the requirements for the class NTG.1, since property NTG.1.1 requires:

> The RNG shall test the external input data provided by a non-physical entropy source in order to estimate the entropy and to detect non-tolerable statistical defects [...], see Killmann and Schindler (2011).

Additionally, `/dev/urandom` violates NTG 1.6 which states

> The average Shannon entropy per internal random bit exceeds 0.997.

Both are clearly not met by `/dev/urandom` due to the fact that the device is non-blocking.

[1]Bundesamt für Sicherheit in der Informationstechnik (BSI).

However, `/dev/urandom` fulfills all other requirements of the class NTG.1, i.e. the conditions NTG 1.2 up to NTG 1.5. In particular, it is a DRG.3 generator if it is properly seeded.

As already mentioned, system events are used to gather entropy on Linux Systems. These events are post-processed and made available to the devices `/dev/random` and `/dev/urandom`. This includes estimating the entropy of the event and mixing.

However, `/dev/urandom` will still supply the user with “randomness” without checking whether the entropy-pool is still sufficiently filled. In fact, the user is instead supplied with pseudorandom data in favor of speed requirements.

In the OpenBSD operating system, none of the random devices is implemented in a blocking mode. The idea is that much potentially bad randomness is still better than the parsimonious use of high-quality randomness. This is in contrast to the opinion, as for example held by the BSI, that one should require all used randomness to be of guaranteed good quality. As of now, there is still no consensus on this issue.

Since the `/dev/urandom` device has undergone a major change introduced by Ts’o (2016) in kernel version 4.8, two kernel versions were benchmarked to test the differences. Namely the original Ubuntu 16.04 kernel 4.4.0 and the more recent version 4.10.0.

3.2 *PRG310-4*

The PRG310-4 gathers entropy from a system of two noisy diodes, see Bergmann (2019), and is connected to a computer via USB. Similar variants exist for different interface types. According to Bergmann (2019), its behavior follows the stochastic model in Killmann and Schindler (2008), who argue that

> [...] the true conditional entropies should be indeed very close to 1 [...], which gives an output of slightly more than 500 kBit internal random numbers per second.

Bergmann (2019) mentions that this device satisfies all requirements for class PTG.3, which are “hybrid physical random number generator with cryptographic post-processing” in the terminology of Killmann and Schindler (2011).

3.3 *AES in Counter Mode*

Due to the fact that since 2008 there is AES-NI,[2] realizing dedicated processor instructions on Intel and AMD processors for the block cipher AES as standardized by NIST (2001a), we add to our comparison the AES counter mode generator. This

[2]For a white paper of AES-NI, see Gueron (2010).

generator is also standardized by NIST (2015) and produces a sequence of 128 bit blocks. We aim at security level of 128 bits, thus employing AES-128 as the underlying block cipher.

The security of the AES generator directly reduces to the security of AES. Indeed, any distinguisher for the pseudorandom generator gives an equally good distinguisher for AES in counter mode. Assuming the latter to be secure, one concludes that also the pseudorandom generator is secure.

However, in contrast to the number-theoretic generators described below, we do not have any reductionist argument in our hands to actually prove that the generator is secure if some presumably hard mathematical problem is intractable. We need to trust that the cipher AES is secure—and the dedicated processor instructions on our CPU work as specified.

When one looks carefully at the definition of a DRG.3 generator in the sense of Killmann and Schindler (2011), AES in counter mode is *not* DRG.3. Specifically, it violates the condition DRG.3.3 of backward secrecy, since the NIST document allows in a single request multiple outputs before the transition function is applied, while the BSI requires that the state transition function of the generator is applied after each new random number.

3.4 Linear Congruential Generators

The linear congruential generator as presented in Lehmer (1951) produces for $i \geq 1$ a sequence of values in $x_i \in \mathbb{Z}_M$, generated by applying for $a, b \in \mathbb{Z}_M$ iteratively

$$x_i = a \cdot x_{i-1} + b \text{ in } \mathbb{Z}_M$$

to a secret random seed $x_0 \in \mathbb{Z}_M$ provided by an external source. The parameters a, b, and M are also kept secret and chosen from the external source.

While the bytes of linear congruential generator outputs are generally well-distributed, with byte entropy close to maximal, the generated sequences are predictable and therefore cryptographically insecure.

Plumstead (1982) describes how to recover the secrets a, b and M from the sequence of $(x_i)_{i\geq 0}$ alone. A possible mitigation against this attack is to output only some *least significant bits* of the x_i. Håstad and Shamir (1985) describe a lattice based attack on such truncated linear congruential generator where all parameters are public. Stern (1987) shows that also in the case when the parameters are kept secret. This attack can be used to predict linear congruential generators that output at least $\frac{1}{3}$ of the bits of the x_i. Contini and Shparlinski (2005) write that there is no cryptanalytic attack known when only approximately $k = \log_2 \log_2 M$ bits are output per round.

We are neither aware of a more powerful attack on the linear congruential generator nor of a more up-to-date security argument for truncated linear congruential generators.

In our evaluation we used a prime modulus M with 2048 bits. Per round we output $k = 11$ bits, which coincides with the value from Contini and Shparlinski (2005) mentioned above. For comparison, we also run the generator with modulus $M = 2^{2048}$ and full output, that is, no truncation, which is basically the fastest number-theoretic generator we can hope for.

The full linear congruential generator is not a pseudorandom generator in the terminology of Killmann and Schindler (2011), since it does not provide forward secrecy. If the sketched truncated version of the linear congruential generator can indeed not be cryptanalyzed then it belongs to the class DRG.3.

3.5 The Blum–Blum–Shub Generator

The Blum–Blum–Shub generator was introduced in 1982 to the cryptographic community and later published in Blum et al. (1986). The generator produces a pseudorandom bit sequence from a random seed by repeatedly squaring modulo a so called *Blum integer* $N = p \cdot q$, where p and q are distinct large random primes congruent to 3 mod 4. In its basic variant, in each round the least significant bit of the intermediate result is returned. Vazirani and Vazirani (1984) proved that the Blum–Blum–Shub generator is secure if $k = \log_2 \log_2 N$ least significant bits are output per round.

Alexi et al. (1988) proved that factoring the Blum integer N can be reduced to being able to guess the least significant bit of any intermediate square with non-negligible advantage. The output of this generator is thus cryptographically secure under the assumption that factoring Blum integers is a hard problem.

In our evaluation p and q are randomly selected 1024 bit primes with $p = q = 3$ mod 4, which corresponds—as above—to the security level of 128 bits following again the BSI (2019a) guideline TR-02102-1.

If factoring Blum integers is hard then the Blum–Blum–Shub generator—properly seeded—is a DRG.3 generator in the terminology of Killmann and Schindler (2011).

3.6 The RSA Generator

The RSA generator was first presented by Shamir (1983) and is one of the pseudorandom generators that are proven to be cryptographically secure under certain number-theoretical assumptions. Analogously to the RSA cryptosystem, the generator is initialized by choosing a modulus N as the product of two large random primes, and an exponent e with $1 < e < \varphi(N) - 1$ and $\gcd(e, \varphi(N)) = 1$. Here, φ denotes Euler's totient function. Starting from a seed x_0 provided by an external source, the generator iteratively computes

$$x_{i+1} = x_i^e \mod N,$$

extracts the least significant k bits of each intermediate result x_i and concatenates them as output.

Our implementation uses a random 2048-bit Blum integer (see Sect. 3.5) as modulus N and various choices for the parameters e and k.

In Alexi et al. (1988) it is shown that the RSA generator is pseudorandom for $k = \log_2 \log_2 N = 11$, under the assumption that the RSA inversion problem is hard. For our tests, we choose $e = 3$, as for small exponents the generator is expected to work fast and because it allows us to compare the results to the runtime of the Blum–Blum–Shub generator.

Under a stronger assumption called the SSRSA assumption, Steinfeld et al. (2006) prove the security of the generator for $k \leq n \cdot (\frac{1}{2} - \frac{1}{e} - \varepsilon - o(1))$ for any $\varepsilon > 0$, giving for $e = 3$ the parameter value $k = 238$. Additionally, we test the larger exponent $e = 2^{16} + 1$, which is widely used in practice, for it is a prime and its structure allows efficient exponentiation, with $k = 921$.

If the RSA inversion problem is hard then the RSA generator—properly seeded—is a DRG.3 generator in the terminology of Killmann and Schindler (2011).

4 Evaluation

To evaluate the efficiency of the generators considered, we developed a framework that runs the software generators based on seed data from the PRG310-4. To this end, we implemented the generators in C, using the GMP library, see Granlund (2014), to accomplish large integer arithmetic. The evaluation framework sequentially runs all generators, reading from one true random source file of 512 kB and producing 512 kB each, while measuring the runtime of each generator and the byte entropy of each output.

All algorithms were run on an Acer V Nitro notebook with a Intel Core i5-4210U CPU at 1.70 GHz with 8 GB RAM. We used Ubuntu 16.04 64-bit with kernel version 4.10.0-32, as well version 4.4.0-92 as reference for the kernel random devices.

This process was repeated 750 times specifically, so that the average runtime of the generators should not deviate considerably from its expectation.

To see this, let A be a randomized algorithm. Then the runtime $t(A)$ is a random variable. Without loss of generality let the runtime be bounded in the interval $I = [0..1]$. We write $t = \mathcal{E}t(A)$ for the expected runtime of A. Consider running the randomized algorithm k times. Then the average runtime of this experiment is

$$X_k = \frac{1}{k} \cdot (t(A)_1 + \ldots + t(A)_k).$$

For its expectation we have

$$\mathcal{E}X_k = \mathcal{E}t(A) = t \in I.$$

Thus, the expectation of the average runtime of k runs is equal to the expectation of a single run. If we observe after k runs an average runtime of X_k, then we can ask:

> How large should k be so that the probability that the observed value X_k significantly differs from its expectation $\mathcal{E}X_k$ is very small?

By Hoeffding's inequality we have

$$\operatorname{prob}(|X_k - \mathcal{E}X_k| \geq \delta) \leq \epsilon$$

for a real number $\delta \in \mathbb{R}_{>0}$ and $\epsilon = 2\exp(-2k\delta^2)$. To be statistically significant, we set $\epsilon = \delta = 0.05$, as typically done in statistics. Then we require that $\operatorname{prob}(|X_k - t| \geq 0.05) \leq 0.05 = 2\exp(-2k \cdot 0.05^2)$, i.e., $k > 737$.

Thus we need at least 737 runs of the algorithm so that the probability that the observed result deviates statistically significant from the actual expected runtime is smaller than 1/20. Thus, 750 runs will do the job.

In order to reduce the impact of other operating system components during our benchmarking, we decided to split up the initialization and generation processes and measure the time for the generation only after a certain amount of data was generated. This way, the throughput of the generators had time to stabilize and we thus omit possible noise that is produced when the generator is started up. To determine the appropriate amount of data to be generated before the measurement, we measured throughput for increasing amounts of data so that we could see at which point the throughput stabilizes.

Figure 1 shows the pseudorandom software generators along with the two versions of `/dev/urandom` as reference points. In the logarithmic scale on

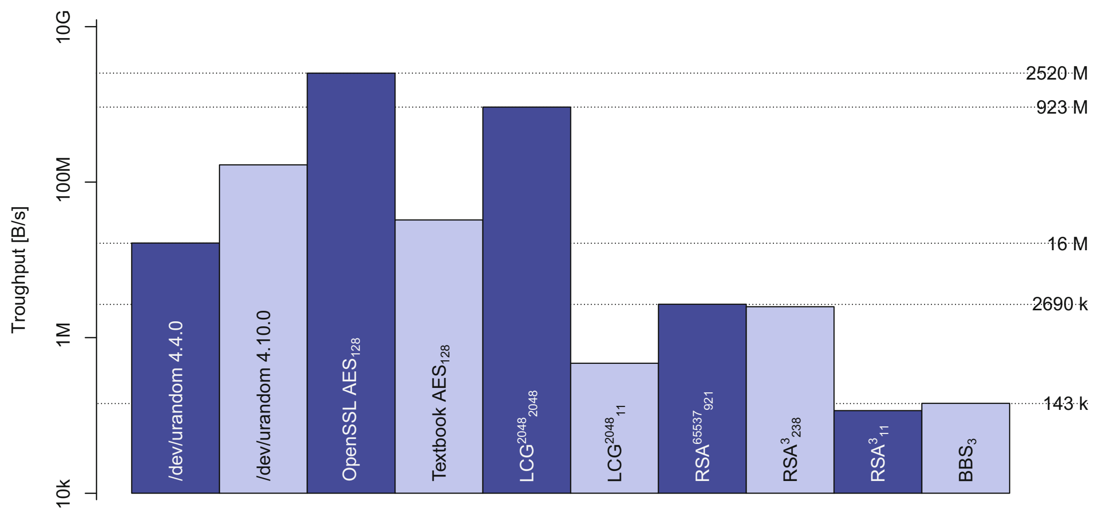


Fig. 1 Generator throughput after initialization for different output lengths

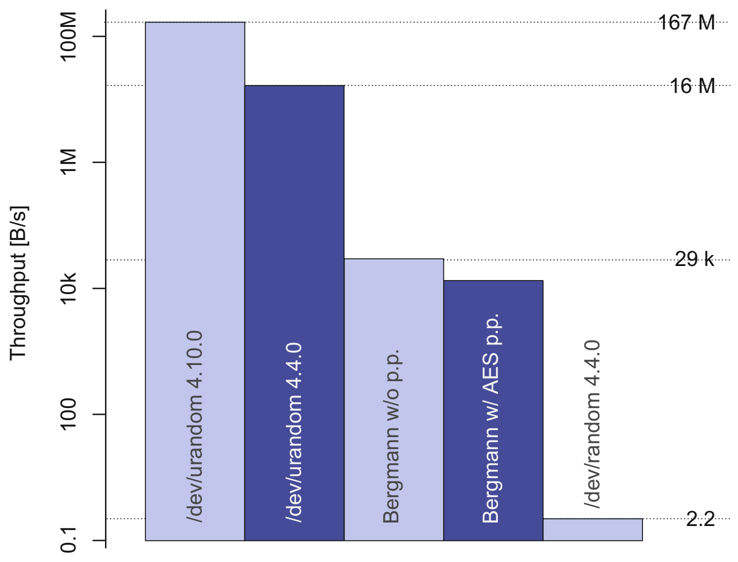


Fig. 2 Physical and non-physical true random generator throughput after initialization

the throughput axis, an AES implementation on AES-friendly hardware has a throughput of about 2.5 GB/s of pseudorandom data, while the RSA generator with 921 bit truncation and 65,537 as public exponent, i.e., the fastest number-theoretic generator assumed to be secure, provides about 2.7 MB/s of pseudorandom data. This makes the latter about 1000 times slower than the AES generator. The linear congruential generator can compete with the fast AES implementation, when not truncating the output, generating about 922 MB/s, but as explained in Sect. 3.4 without truncation the generator is not cryptographically secure. As a fairer comparison to the textbook implementations of the number theoretic generators, the textbook version of AES still generates 32.7 MB/s, beating the RSA implementation by a factor of ten.

A second benchmark was performed for the different physical generators considered, depicted in Fig. 2. Again a logarithmic scale is employed to allow having the `/dev/urandom` devices with up to 166.8 MB/s of output and the `/dev/random` device with 2.2 B/s of output in the same picture. The most surprising observation is the jump in performance regarding the `/dev/urandom` device introduced by the re-implementation described in Sect. 3.1. When only considering blocking physical devices, i.e., taking out `/dev/urandom` completely, the Bergmann generator outperforms the `/dev/random` device easily both with (13 kB/s) and without post-processing (29 kB/s).

The amount of randomness needed for seeding the software generators differs considerably. The least amount is needed by the AES based generators, which need 128 bits for the textbook and 256 bits for the OpenSSL implementation. The latter randomizes initial counter and key, whereas the former only randomizes the key. Both the RSA and the Blum–Blum–Shub generator need to generate two 1024 bit primes. The textbook method chooses uniformly random integers of the appropriate size and tests them successively for compositeness. This requires tests on expectedly $2 \cdot 1024 \cdot \ln 1024 \approx 14{,}200$ different integers by the prime number theorem, thus consuming approximately 1.8 MB of seed randomness. The primality tests themselves might consume additional randomness if a probabilistic variant is employed. There are cheaper methods, though, reducing the necessary amount of randomness to 2048 bits only. For details on this matter see Loebenberger and Nüsken (2014).

The linear congruential generator additionally randomizes the initial state and thus consumes further 2048 bits for seeding.

Taking the throughput of the physical generators into account, the amount of time needed between possible reseeding ranges from $\frac{128}{8} \cdot \frac{1}{29{,}000} = 0.00055\,\text{s}$ for the textbook AES generator seeded by the Bergmann generator to $\frac{6144}{8} \cdot \frac{1}{2.2} = 349\,\text{s}$ for the linear congruential generator seeded by `/dev/random`.

While the statistical quality of each generators output is not dependent on the reseeding, the amount of total entropy is not raised by any internal calculation, making regular reseeding sensible. Using the Bergmann generator for seeding, even the linear congruential generator can be reseeded every 0.026 s, which seems a reasonable time span especially in networking contexts.

5 Conclusion and Future Work

We implemented a number of software random generators and compared their performance to physical generators. A blocking `/dev/random` is way too slow to be of practical use as the only source of (pseudo-)randomness, except for seeding software generators. The generator PRG310-4 is roughly as fast as our Blum–Blum–Shub implementation. However, both are surpassed by the RSA generator when run with a fast parameter set, which offers the same level of security.

The most interesting result is the vast difference between blocking and non-blocking random devices. This illustrates in a nice way the still open question whether lots of potentially bad randomness surpass the parsimonious use of guaranteed high-quality randomness.

The results also suggest a profitable symbiosis of hardware-generated seeds and number-theoretic high throughput—rather the reverse of the situation in other cryptographic contexts, say, the Diffie–Hellman exchange of keys for fast AES encryption.

The speedup introduced by the AES-NI instruction-set allows to generate 151 MB/s on a laptop computer, surpassing the requirements of the NGINX cluster (1.3 MB/s) by far, implying a negligible CPU-load.

Practical use of our findings has not taken place yet. Depletion of `/dev/random` is a realistic issue—workarounds for implied problems even suggest using the non-blocking `/dev/urandom` as a physical generator, see Searle (2008). However, prohibiting the use of `/dev/urandom` for key generation is also under debate, see Bernstein (2014), and there seems to be no consensus in the near future.

As a next step, implementing and testing on kernel level using optimized implementations is recommended.

Implementing an AES based random generator in the Linux kernel appears to be reasonable, but other platforms (i.e. ARM) may favor other hardware-accelerated ciphers for better performance and less CPU load. Thus the cipher must be made configurable.

Acknowledgements This work arose from a course taught by Joachim von zur Gathen and Daniel Loebenberger at the University of Bonn. It was funded by the B-IT foundation and the state of North Rhine–Westphalia. Substantial improvements of the presentation of the article were conducted later at genua GmbH, Germany. We thank the anonymous referees for their fruitful comments.

References

Werner Alexi, Benny Chor, Oded Goldreich & Claus P. Schnorr (1988). RSA and Rabin functions: Certain Parts are As Hard As the Whole. *SIAM Journal on Computing* **17**(2), 194–209. ISSN 0097-5397. https://doi.org/10.1137/0217013.

Frank Bergmann (2019). Professionelle Zufallsgeneratoren für kryptografisch sichere Zufallszahlen. http://www.ibbergmann.org/.

D. J. Bernstein (2014). cr.yp.to: 2014.02.05: Entropy Attacks! http://blog.cr.yp.to/20140205-entropy.html. Last access 27 June 2015.

L. Blum, M. Blum & M. Shub (1986). A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing* **15**(2), 364–383. https://doi.org/10.1137/0215025.

A. Borodin, J. von zur Gathen & J. E. Hopcroft (1982). Fast parallel matrix and GCD Computations. *Information and Control* **52**, 241–256. http://dx.doi.org/10.1016/S0019-9958(82)90766-5. Extended Abstract in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, Chicago IL* (1982).

BSI (2019a). Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Technische Richtlinie BSI TR-02102-1, Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf.

BSI (2019b). Overview of Linux kernels with NTG.1-compliant random number generator /dev/random. Technical report, Bundesamt für Sicherheit in der Informationstechnik. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/LinuxRNG/NTG1_Kerneltabelle_EN.pdf.

Aleksei Burlakov, Johannes vom Dorp, Joachim von zur Gathen, Sarah Hillmann, Michael Link, Daniel Loebenberger, Jan Lühr, Simon Schneider & Sven Zemanek (2015a). Comparative analysis of pseudorandom generators. In *Proceedings of the 22nd Crypto-Day, 9–10 July 2015, Munich*. Gesellschaft für Informatik. http://fg-krypto.gi.de/fileadmin/fg-krypto/Handout-22.pdf.

Aleksei Burlakov, Johannes vom Dorp, Sarah Hillmann, Michael Link, Jan Lühr, Simon Schneider & Sven Zemanek (2015b). Comparative analysis of pseudorandom generators: Source code. BitBucket. https://bitbucket.org/sirsimonrattle/15ss-taoc.

Jim Clarke (2019). An Optimist's View of the 4 Challenges to Quantum Computing. *IEEE Spectrum* https://spectrum.ieee.org/tech-talk/computing/hardware/an-optimists-view-of-the-4-challenges-to-quantum-computing.

SCOTT CONTINI & IGOR E. SHPARLINSKI (2005). On Stern's Attack Against Secret Truncated Linear Congruential Generators. In *Information Security and Privacy—10th Australasian Conference, ACISP 2005, Brisbane, Australia, July 4–6, 2005. Proceedings*, COLIN BOYD & JUAN MANUEL GONZÁLEZ NIETO, editors, volume 3574 of *Lecture Notes in Computer Science*, 52–60. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-26547-4 (Print) 978-3-540-31684-8 (Online). ISSN 0302-9743. https://doi.org/10.1007/11506157_5.

STEPHEN A. COOK (1971). The Complexity of Theorem–Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights OH*, 151–158. ACM Press.

MIKHAIL DYAKONOV (2018). The Case Against Quantum Computing. *IEEE Spectrum*. https://spectrum.ieee.org/computing/hardware/the-case-against-quantum-computing.

D. EASTLAKE, J. SCHILLER & S. CROCKER (2005). Randomness Requirements for Security. BCP 106, RFC Editor. http://www.rfc-editor.org/rfc/rfc4086.txt.

R. FISCHLIN & C. P. SCHNORR (2000). Stronger Security Proofs for RSA and Rabin Bits. *Journal of Cryptology* **13**(2), 221–244. https://doi.org/10.1007/s001459910008. Communicated by Oded Goldreich.

JOACHIM VON ZUR GATHEN & DANIEL LOEBENBERGER (2018). Why one cannot estimate the entropy of English by sampling. *Journal of Quantitative Linguistics* **25**(1), 77–106. https://doi.org/10.1080/09296174.2017.1341724.

ODED GOLDREICH (2001). *Foundations of Cryptography*, volume I: Basic Tools. Cambridge University Press, Cambridge. ISBN 0-521-79172-3.

ODED GOLDREICH, AMIT SAHAI & SALIL VADHAN (1999). Can Statistical Zero Knowledge Be Made Non-interactive? Or On the Relationship of $\mathcal{SZK}$ and $\mathcal{NISZK}$. In *Advances in Cryptology: Proceedings of CRYPTO 1999, Santa Barbara, CA*, M. WIENER, editor, volume 1666 of *Lecture Notes in Computer Science*, 467–484. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-48405-9 (Online) 978-3-540-66347-8 (Print). ISSN 0302-9743. https://doi.org/10.1007/3-540-48405-1_30.

TORBJÖRN GRANLUND (2014). The GNU Multiple Precision Arithmetic Library. C library. https://gmplib.org/. Last access 25th June 2015.

SHAY GUERON (2010). Intel® Advanced Encryption Standard (AES) New Instructions Set: White paper. Technical report, Intel Corporation.

ZVI GUTTERMAN, BENNY PINKAS & TZACHY REINMAN (2006). Analysis of the Linux Random Number Generator. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 371–385. IEEE Computer Society. ISBN 0-7695-2574-1. https://doi.org/10.1109/SP.2006.5. Also available at http://eprint.iacr.org/2006/086.

JOHAN HÅSTAD & ADI SHAMIR (1985). The Cryptographic Security of Truncated Linearly Related Variables. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, 356–362. ACM, New York, NY, USA. ISBN 0-89791-151-2. https://doi.org/10.1145/22145.22184.

WOLFGANG KILLMANN & WERNER SCHINDLER (2008). A Design for a Physical RNG with Robust Entropy Estimators. In *CHES 2008*, ELISABETH OSWALD & PANKAJ ROHATGI, editors, volume 5154 of *LNCS*, 146–163. ISBN 978-3-540-85052-6. https://doi.org/10.1007/978-3-540-85053-3_10.

WOLFGANG KILLMANN & WERNER SCHINDLER (2011). A proposal for: Functionality classes for random number generators. Anwendungshinweise und Interpretationen zum Schema AIS 20/AIS 31, Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf?__blob=publicationFile. Version 2.0.

PATRICK LACHARME, ANDREA RÖCK, VINCENT STRUBEL & MARION VIDEAU (2012). The Linux Pseudorandom Number Generator Revisited. https://eprint.iacr.org/2012/251.pdf. Last access 27 June 2015.

D. H. LEHMER (1951). Mathematical methods in large-scale computing units. In *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery, 13–16 September*

1949, volume 26 of *Annals of the Computation Laboratory of Harvard University*, 141–146. Harvard University Press, Cambridge, Massachusetts. https://archive.org/details/proceedings_of_a_second_symposium_on_large-scale_.

DANIEL LOEBENBERGER & MICHAEL NÜSKEN (2014). Notions for RSA integers. *International Journal of Applied Cryptography* **3**(2), 116–138. ISSN 1753-0571 (online), 1753-0563 (print). http://dx.doi.org/10.1504/IJACT.2014.062723.

STEPHAN MÜLLER (2019). Documentation and Analysis of the Linux Random Number Generator. Technical report, atsec information security GmbH for the Federal Office for Information Security. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN.pdf.

NGINX (2016). Sizing Guide for Deploying NGINX Plus on Bare Metal Servers. Technical report, NGINX Inc.
https://cdn-1.wp.nginx.com/wp-content/files/nginx-pdfs/Sizing-Guide-for-Deploying-NGINX-on-Bare-Metal-Servers.pdf. Last access 16 August 2017.

NIST (2001a). *Federal Information Processing Standards Publication 197—Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. Federal Information Processings StandardsPublication 197.

NIST (2001b). NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation.

NIST (2015). NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

JOAN B. PLUMSTEAD (1982). Inferring a Sequence Generated by a Linear Congruence. *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, Chicago IL* 153–159. https://doi.org/10.1109/SFCS.1982.73. Published as Joan Boyar, *Inferring Sequences Produced by Pseudo-Random Number Generators*, Journal of the ACM **36** (1), 1989, pages 129–141, https://doi.org/10.1145/58562.59305.

WERNER SCHINDLER & WOLFGANG KILLMANN (2003). Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications. In *Cryptographic Hardware and Embedded Systems—CHES 2002*, JR. BURTON S. KALISKI, ÇETIN K. KOÇ & CHRISTOF PAAR, editors, volume 2523 of *LNCS*, 431–449. Springer-Verlag, Berlin, Heidelberg. ISSN 0302-9743 (Print), 1611–3349 (Online). https://doi.org/10.1007/3-540-36400-5_31.

BRUCE SCHNEIER (2008). Random Number Bug in Debian Linux. Weblog. https://www.schneier.com/blog/archives/2008/05/random_number_b.html.

CHRIS SEARLE (2008). Increase entropy on a 2.6 kernel Linux box. https://www.chrissearle.org/2008/10/13/Increase_entropy_on_a_2_6_kernel_linux_box/. Last access 27 June 2015.

ADI SHAMIR (1983). On the Generation of Cryptographically Strong Pseudorandom Sequences. *ACM Trans. Comput. Syst.* **1**(1), 38–44. ISSN 0734-2071. https://doi.org/10.1145/357353.357357.

PETER W. SHOR (1999). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review* **41**(2), 303–332.

RON STEINFELD, JOSEF PIEPRZYK & HUAXIONG WANG (2006). On the Provable Security of an Efficient RSA-Based Pseudorandom Generator. In *Advances in Cryptology: Proceedings of ASIACRYPT 2006, Shanghai, China*, XUEJIA LAI & KEFEI CHEN, editors, volume 4284 of *Lecture Notes in Computer Science*, 194–209. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-49475-1. ISSN 0302-9743 (Print) 1611-3349 (Online). https://doi.org/10.1007/11935230_13.

JACQUES STERN (1987). Secret linear congruential generators are not cryptographically secure. *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles CA* 421–426.

THEODORE TS'O (2016). Commit of ChaCha based urandom to Linux kernel. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/\?id\=e192be9d9a30555aae2ca1dc3aad37cba484cd4a. Last access 29 August 2017.

UMESH V. VAZIRANI & VIJAY V. VAZIRANI (1984). Efficient and Secure Pseudo-Random Number Generation (Extended Abstract). In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, Singer Island FL*, 458–463. IEEE Computer Society Press. ISBN 0-8186-0591-X. ISSN 0272-5428.

Texts & Monographs in Symbolic Computation

Veronika Pillwein
Carsten Schneider *Editors*

Algorithmic Combinatorics: Enumerative Combinatorics, Special Functions and Computer Algebra

In Honour of Peter Paule on his 60th Birthday

Springer

Texts & Monographs in Symbolic Computation

A Series of the Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria

Veronika Pillwein • Carsten Schneider
Editors

Algorithmic Combinatorics: Enumerative Combinatorics, Special Functions and Computer Algebra

In Honour of Peter Paule on his 60th Birthday

Springer

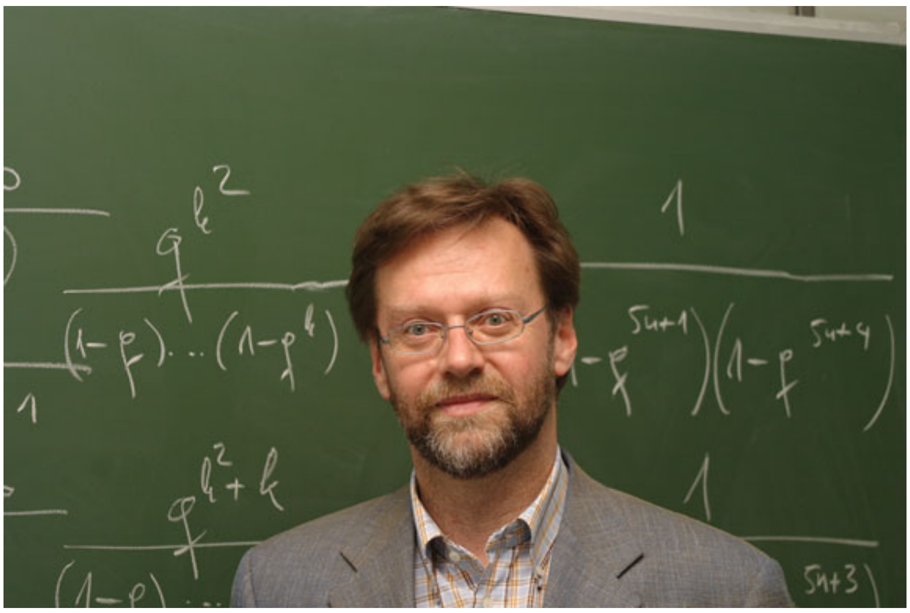
Editors

Veronika Pillwein
Research Institut for Symbolic Computation
Johannes Kepler University of Linz
Linz, Oberösterreich, Austria

Carsten Schneider
Research Institut for Symbolic Computation
Johannes Kepler University of Linz
Linz, Austria

ISSN 0943-853X ISSN 2197-8409 (electronic)
Texts & Monographs in Symbolic Computation
ISBN 978-3-030-44558-4 ISBN 978-3-030-44559-1 (eBook)
https://doi.org/10.1007/978-3-030-44559-1

Mathematics Subject Classification: 33xx, 68W30, 05Axx, 05A17, 05A15, 05A19, Primary 11P83; Secondary 05A17


This volume is devoted to the 60th birthday of Peter Paule

Foreword

This volume is dedicated to Peter Paule on the occasion of his 60th birthday. It contains the proceedings of the workshop, Combinatorics, Special Functions and Computer Algebra (Paule60), held on May 17–18, 2018, at the Research Institute for Symbolic Computation (RISC) in Hagenberg, Austria. This is overwhelmingly merited in light of Peter's distinguished career. A former Humboldt Fellow, Peter has been a major player in the applications of computer algebra and has been director of RISC since 2009. He is a member of Academia Europaea and a Fellow of the American Mathematical Society.

In the early 1980s, Professor Johann Cigler gave a wonderful talk at an Oberwolfach conference on combinatorics. In the talk, he extolled the outstanding work of several of his students especially that of Peter Paule. Subsequently, at the next Oberwolfach conference on combinatorics, Peter himself gave a presentation, and I was more than pleased to make the acquaintance of this rising star. This was the beginning of a grand and lasting friendship.

I have written more papers with Peter than anyone else, 15 in all. The majority concern the computer algebra implementation of P. A. MacMahon's Partitions Analysis (often joint with Peter's student, Axel Riese). This collaboration was one of the most wonderful adventures of my career. Notable partly because this allowed us to track down a number of fascinating mathematical objects, but mostly because it is a joy to collaborate with this optimistic man who is always full of joie de vivre.

Peter is the opposite of the stereotypical mathematician. As we all know, the way to get ahead in mathematics is to sit alone in a small room for days on end concentrating intensely on esoteric abstractions. As a result, a number of us are somewhat socially challenged. Peter has completely avoided anything like this outcome. Not only do mathematicians enjoy time spent with him, but often the RISC visitor's entire family remembers him warmly. Indeed, he is often the only mathematician they do remember.

It is a great honor to be asked to prepare this foreword. Best wishes to you, Peter, from all your friends and admirers (and me especially). We look forward to your many future achievements.

University Park, PA, USA George E. Andrews
November 2019

Preface

The book is centered around *Algorithmic Combinatorics* which covers the three research areas of *Enumerative Combinatorics*, *Special Functions*, and *Computer Algebra*. What these research fields share is that many of their outstanding results do not only have applications in Mathematics but also other disciplines, such as Computer Science, Physics, Chemistry, etc. A particular charm of these areas is how they interact and influence one another. For instance, combinatorial or special functions' techniques have motivated the development of new symbolic algorithms. In particular, the first proofs of challenging problems in Combinatorics and Special Functions were derived by making essential use of Computer Algebra.

This book addresses these interdisciplinary aspects with research articles and up to date reviews that are suitable for graduate students, researchers, or practitioners who are interested in solving concrete problems within mathematics and other research disciplines. Algorithmic aspects will be emphasized and the corresponding software packages for concrete problems are introduced whenever applicable.

When the Search for Solutions Can Be Terminated (Sergei A. Abramov) addresses the problem that in algorithms often the nonexistence of solutions can only be detected in the final stages, after carrying out a lot of heavy computations. In this article, it is shown how to introduce early termination checkpoints in an algorithm for finding rational solutions of differential systems.

In *Euler's Partition Theorem and Refinements Without Appeal to Infinite Products* (Krishnaswami Alladi), combinatorial arguments on 2-modular Ferrers diagrams are combined in a novel way in order to find and prove analogues of some important fundamental theorems in the theory of partitions.

In *Sequences in Partitions, Double q-Series and the Mock Theta Function $\rho_3(q)$* (George E. Andrews), a skillful mix of techniques based on q-difference equations, generating functions, series expansions, and bijective maps based on the combinatorics of integer partitions (and overpartitions) are applied to gain new insights of combinatorial aspects of a family of double sum hypergeometric q-series and their connection to many famous identities for integer partitions.

In *Refined q-Trinomial Coefficients and Two Infinite Hierarchies of q-Series Identities* (Alexander Berkovich and Ali Kemal Uncu), symbolic summation tools and classical methods from enumerative combinatorics are combined to discover and explore new doubly bounded polynomial identities in terms of refined q-trinomial coefficients.

In *Large Scale Analytic Calculations in Quantum Field Theories* (Johannes Blümlein), a general overview of computer algebra and special function tools is presented that are heavily used to solve large-scale problems in relativistic renormalizable quantum field theories. These analytic tools originate from algorithmic combinatorics or are suitable for problems coming from this field.

In *An Eigenvalue Problem for the Associated Askey–Wilson Polynomials* (Andrea Bruder, Christian Krattenthaler, and Sergei K. Suslov), an auxiliary bivariate function is introduced that links to associated ordinary Askey–Wilson polynomials. With the aid of computer algebra, from this relation an eigenvalue problem for the associated Askey–Wilson polynomials is constructed.

Context-Free Grammars and Stable Multivariate Polynomials Over Stirling Permutations (William Y.C. Chen, Robert X.J. Hao, and Harold R.L. Yang) resolves two open questions raised by Haglund and Visontai in their study of stable multivariate refinements of second-order Eulerian polynomials.

In *An Interesting Class of Hankel Determinants* (Johann Cigler and Mike Tyson), Hankel determinants $d_r(n)$ of a binomial sequence are considered for which for general integers n and r no closed-form exists. Using the methods presented here, formulas valid for all $r \geq 0$ and particular arithmetic progressions are given.

In *A Sequence of Polynomials Generated by a Kapteyn Series of the Second Kind* (Diego Dominici and Veronika Pillwein), an infinite sum involving squares of Bessel-J functions with an extra parameter n is explored. A closed-form representation is derived for this series in terms of a specific polynomial whose degree depends on n, and a recurrence relation is computed and verified with computer algebra methods that produces the coefficients of this polynomial efficiently.

In *Comparative Analysis of Random Generators* (Johannes vom Dorp, Joachim von zur Gathen, Daniel Loebenberger, Jan Lühr, and Simon Schneider), the research field of random number generators is introduced for nonexperts. A careful comparison between pseudorandom number generators and hardware controlled versions is carried out carefully in terms of their output rate.

In *Difference Equation Theory Meets Mathematical Finance* (Stefan Gerhold and Arpad Pinter), the authors make those two ends meet unexpectedly through Pringsheim's theorem and two asymptotic methods (saddle point and Hankel contour asymptotics).

In *Evaluations as L-Subsets* (Adalbert Kerber), logical systems beyond classical Boolean logic are considered by utilizing lattice-valued evaluations of statements in a novel way. In particular, examples are elaborated that demonstrate the practicality of real-world problems.

In *Exact Lower Bounds for Monochromatic Schur Triples and Generalizations* (Christoph Koutschan and Elaine Wong), exact and sharp lower bounds for the number of generalized monochromatic Schur triples subject to all 2-colorings are explored. In their challenging enterprise, they use low-dimensional polyhedral combinatorics leading to many case distinctions that could be treated successfully by the symbolic computation technique of cylindric algebraic decomposition.

In *Evaluation of Binomial Double Sums Involving Absolute Values* (Christian Krattenthaler and Carsten Schneider), double sums from a general family are considered, where the main difficulty lays in the appearance of absolute values. It is shown that these sums in general can be expressed as a linear combination of just four simple hypergeometric expressions.

In *On Two Subclasses of Motzkin Paths and Their Relation to Ternary Trees* (Helmut Prodinger, Sarah J. Selkirk, and Stephan Wagner), paths with alternating east and north-east steps are shown to give nice enumeration formulas via generalized Catalan numbers.

In *A Theorem to Reduce Certain Modular Form Relations Modulo Primes* (Cristian-Silviu Radu), a question raised by Peter Paule is settled that is of algorithmic relevance to the theory of modular forms. It is shown that the problem to decide if a certain modular form relation modulo a prime holds can be reduced to check congruences modulo p between meromorphic modular forms.

In *Trying to Solve a Linear System for Strict Partitions in "closed form"* (Volker Strehl), a challenging linear system for strict partitions in relation to Schur functions and symmetric functions is investigated that utilizes graph theory in combination with the theory of partitions in a novel way.

In *Untying the Gordian Knot via Experimental Mathematics* (Yukun Yao and Doron Zeilberger), two new applications of automated guessing are given: one related to enumerating spanning trees using transfer matrices and one about determinants of certain families of matrices. Using symbolic computation, the painful human approach is avoided.

This book is an offspring of the workshop "Combinatorics, Special Functions and Computer Algebra" at the occasion of Peter Paule's 60th birthday (https://www3.risc.jku.at/conferences/paule60/). We would like to thank Tanja Gutenbrunner and Ramona Pöchinger from RISC for all their help to organize this wonderful event. In particular, we would like to thank the Austrian FWF in the frameworks of the SFB "Algorithmic and Enumerative Combinatorics" and the Doctoral Program "Computational Mathematics" for the financial support. Furthermore, we thank Karoly Erdei for providing us with the above picture that illustrates Peter Paule's mathematical passion. Finally, we would like to thank all the authors for their stimulating contributions and the referees in the background for their valuable comments.

Linz, Austria — Veronika Pillwein
Linz, Austria — Carsten Schneider
November 2019

Contents

List of Contributors

Sergei A. Abramov Dorodnitsyn Computing Centre, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia

Krishnaswami Alladi Department of Mathematics, University of Florida, Gainesville, FL, USA

George E. Andrews The Pennsylvania State University, University Park, PA, USA

Alexander Berkovich Department of Mathematics, University of Florida, Gainesville, FL, USA

Johannes Blümlein Deutsches Elektronen-Synchrotron, DESY, Zeuthen, Germany

Andrea Bruder Department of Mathematics and Computer Science, Colorado College, Colorado Springs, CO, USA

William Y. C. Chen Center for Applied Mathematics, Tianjin University, Tianjin, People's Republic of China

Johann Cigler Fakultät für Mathematik, Universität Wien, Vienna, Austria

Diego Dominici Johannes Kepler University Linz, Doktoratskolleg, "Computational Mathematics", Linz, Austria

Department of Mathematics State, University of New York, New Paltz, NY, USA

Stefan Gerhold TU Wien, Vienna, Austria

Robert X. J. Hao Department of Mathematics and Physics, Nanjing Institute of Technology, Nanjing, Jiangsu, People's Republic of China

Adalbert Kerber Mathematics Department, University of Bayreuth, Bayreuth, Germany

Christoph Koutschan Johann Radon Institute for Computational and Applied Mathematics, Austrian Academy of Sciences, Linz, Austria

Christian Krattenthaler Fakultät für Mathematik, Universität Wien, Vienna, Austria

Daniel Loebenberger OTH Amberg-Weiden and Fraunhofer AISEC, Weiden, Germany

Jan Lühr Bonn-Aachen International Center for Information Technology, Universität Bonn, Bonn, Germany

Veronika Pillwein Johannes Kepler University Linz, Research Institute for Symbolic Computation (RISC), Linz, Austria

Arpad Pinter TU Wien Alumnus, Vienna, Austria

Helmut Prodinger Stellenbosch University, Department of Mathematical Sciences, Stellenbosch, South Africa

Carsten Schneider Johannes Kepler University Linz, Research Institute for Symbolic Computation (RISC), Linz, Austria

Simon Schneider Bonn-Aachen International Center for Information Technology, Universität Bonn, Bonn, Germany

Sarah Selkirk Stellenbosch University, Department of Mathematical Sciences, Stellenbosch, South Africa

Cristian-Silviu Radu Johannes Kepler University Linz, Research Institute for Symbolic Computation (RISC), Linz, Austria

Volker Strehl Department Informatik, Friedrich-Alexander Universität, Erlangen, Germany

Sergei K. Suslov School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ, USA

Mike Tyson

Ali Kemal Uncu Johannes Kepler University Linz, Research Institute for Symbolic Computation (RISC), Linz, Austria

Johannes vom Dorp Fraunhofer FKIE, Bonn, Germany

Joachim von zur Gathen Bonn-Aachen International Center for Information Technology, Universität Bonn, Bonn, Germany

Stephan Wagner Stellenbosch University, Department of Mathematical Sciences, Stellenbosch, South Africa

Elaine Wong Johann Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences, Linz, Austria

Harold R. L. Yang School of Science, Tianjin University of Technology and Education, Tianjin, People's Republic of China

Yukun Yao Rutgers University-New Brunswick, Piscataway, NJ, USA

Doron Zeilberger Rutgers University-New Brunswick, Piscataway, NJ, USA