3D Visualization Applied to PRBGs and Cryptography – Long Version

Michel Dubois, Eric Filiol*

Operational Cryptology and Virology Laboratory, 38 Rue des Docteurs Calmette et Guérin, 53000 Laval, France

Copyright ©2016 by authors, all rights reserved. Authors agree that this article remains permanently open access under the terms of the Creative Commons Attribution License 4.0 International License

Abstract Today there is no easy and quick way to analyse and differentiate random data. However, all day long our computers generate pseudo random data, our cryptographic algorithms tend to act as pseudo random generator of data to better hide the message. So we can then ask whether is it possible to quickly determine the algorithm used to construct a random sequence of numbers and in a second time, distinguish between a PRBG or a cryptographic algorithm? In this paper, we present a new approach, to visualize, in a two and three dimensions environment at the same time, a sequence issued from a pseudo random bit generator or from cryptographic algorithms. To embody our idea, we assume that sequences produced by PRBG and Cryptographic algorithms are comparable to a nonlinear system generating a chronological series of data. We have developed some tools to realize our analysis and use them to well known kinds of PRBG and to the AES. Even, if our approach can't serve as determining proof on the quality of an alea, it can bring a great help to quickly (because visually) distinguish two random sequences and eventually find some statistical bias.

Keywords Block Cipher, Boolean Function, Cryptanalysis, AES

1 Introduction

In computer security, visualization is commonly used for different tasks like log analysis [16], attacks detection [13], binary analysis [4] and reverse engineering [3, 2] but today there is no easy way to analyze and differentiate random data. However, operating systems or cryptographic protocols commonly use randomness, for example, to generate a TCP sequence number or generate a random encryption key for WiFi or Web.

Likewise, the Grail of any cryptographic algorithm is to obtain, at each internal step and after the encryption process, a sequence appearing to be the closest possible of perfect random. Indeed, the security of a cryptographic algorithm depends on its ability to generate unpredictable quantities. Assuming that the perfect randomness is only a philosophical view and, in fact, the perfection of the randomness de-

pends on statistical tests were applied to him [5], we can say the cryptographic randomness must be random in the sense that the probability of a particular value chosen must be low enough to prevent an opponent to gain the advantage by optimizing a search strategy based on this probability [12].

So we have pseudorandom number generator algorithms and cryptographic algorithms which producing data that seem random. The challenge is multiple. First of all, how to quickly determine the algorithm used to construct a random sequence of numbers. Then, is it possible to distinguish between a PRBG or a cryptographic algorithm. Finally, if we have a cryptographic algorithm, is it possible to use visualization to realize a first approach of its security.

This paper is composed as follows. In Section 2, pseudorandom bit generators are described. In Section 3, methodologies to display linear sequence in 2 and 3 dimensions space are presented. In Section 4, we use 2D and 3D visualizations to analyze some pseudorandom bit generator sequences. In Section 5, we apply our approach on the RC4 and AES algorithms. Finally, in Section 6, conclusions and future directions are provided.

2 Pseudorandom Bit Generator

A random bit generator is a device or an algorithm that produces a statistically independent bit sequence and unbiased [12].

Some hardwares generate randomness from the time between the emission of particles during radioactive decay phase or from the thermal noise from resistance or semiconductor diode. Similarly, to generate randomness, some software use algorithms combining various sources such as the time between two keystrokes, mouse movement or content of input/output buffers.

A pseudorandom bit generator (PRBG) is a deterministic algorithm which, from a truly random sequence of bits of length k , produces a sequence of bits of length $l \gg k$ seemingly random. The initial sequence is called the seed while the sequence produced by the PRBG is called pseudo-random bit sequence.

Generate random number is a difficult problem for computers because they are deterministic and as said by John Von Neumann "Any one who considers arithmetical methods of producing random digits is, of course, in a state of

3.2 3D representation

After the two-dimensional approach, we will try to represent the sequences obtained in a 3D environment.

As we have said above, a PRBG is comparable to a non-linear system generating a chronological series of data. One of the most commonly used means for analysing a series of this type is to rebuild its phase space using the method of delays [9]. The phase space is a space of n dimensions that completely describes the state of a n variables system. For example, the phase space describing the landing of a rocket is a two-dimensional space. The first dimension is the velocity of the rocket and the second dimension is the ground distance. The phase space is then a graph representing the velocity on x-axis and the ground distances on y-axis. Thus, for the rocket to land smoothly, it is necessary that the curve describing the progress in its phase space tends to zero (see figure 3).

Figure 3. Phase space of the rocket path

Here, our goal is to represent in three-dimensional a one-dimensional sequence. The method of delays [14] allows to reconstruct the missing dimensions using the previous values as additional coordinates. For this, instead of using the raw values returned by the function, we calculate, for each coordinate, the difference of two successive values. This allows us to generate more useful results to show the dynamics of the function. So if $s[t]$ is the sequence provided by a PRBG in function of time t , then the coordinates x, y, z of a point in our environment are calculated from following equations:

$$\begin{aligned}x[t] &= s[t - 2] - s[t - 3] \\y[t] &= s[t - 1] - s[t - 2] \\z[t] &= s[t] - s[t - 1]\end{aligned}$$

Then, representing the point sequence thus obtained in a three-dimensional environment we obtain a specific shape to the given function of PRBG. This form, called *attractor*, reveals the complex nature of the dependencies between the different elements of the sequence generated by the algorithm investigated [22, 23].

As a concrete example, the sequence of numbers of the figure 4 comes from the PRBG algorithm that generates

the TCP session sequence numbers of the operating system GNU/Linux RedHat in its version 7.3.

```
3281499104, 3271545868, 3287443610,
3238749981, 3274168813, 3302234066,
3229771300, 3287970591, 3295595222,
3298841199, 3292774952, 3294591612,
3294540537, 3294046036, 3296969037,
3293746299, 3300112100, 3292483752
```

Figure 4. Sample sequences of Redhat 7.3 PRBG

At first glance, it is difficult to determine whether there is a link between the different elements of this sequence of numbers. By cons, the figure 5 formalizes this relationship by unveiling a shape that is characteristic of PRBG used by the Linux 2.4.18 kernel used by this Linux distribution.

Figure 5. Attractor of Redhat 7.3 PRBG - view 45°

Through this presentation mode, we are now easier to identify a PRBG algorithm, given that the same algorithm will always give the same attractor.

4 PRBG samples

We will now study some PRBG algorithms.

4.1 True alea

Before starting our comparison of PRBG, we need a reference, that is to say the representation of the attractor of a true alea. As a computer is a deterministic engine, it is not possible to it to produce a true alea. Only hardware equipment based on physical random elements can provide this type of alea.

The website www.random.org provides the ability to generate sequences of true alea. It uses atmospheric noise to produce this alea. This site is the result of a scientific project of Dr. Mads Haahr from the "School of Computer Science and Statistics" from Dublin Trinity College. It is used for online games, to generate the random lottery games, for science projects... We generated a sequence of 10 000 random integers from this site. The first inputs of this sequence are as presented in figure 6.

And the 3-dimensional representation of the corresponding attractor is shown in figure 7.

The figure 7 shows a three-dimensional environment, materialized by a cube and the three axes x in red, y in green

72329, 95447, 11130, 52803,
25986, 58390, 84305, 98618,
54545, 64850, 27412, 15977,
13214, 30421, 91625, 48878,
35783, 58844, 16061, 74799

Figure 6. Sample sequences of true alea

Figure 7. Attractor of a true alea - view 45°

and z in blue. The distribution of points in this space forms a homogeneous cloud and evenly covers the volume along the three axes. No specific pattern emerges. The values provided are between 0 and 99995 with an average value of 54707.79. In the right corner we have the Hilbert curve representation of the data set. The colors in this curve seem completely random.

In addition, the program we use, assigns a specific color to each point: all the colors of the color palette is spread over all the points in chronological order. Thus, the first item on the list receives the first color in the palette, the second point receives the second color and so on until the last point. This principle allows us to add a fourth dimension to our graph: the time.

On our curve the color distribution seems completely random.

4.2 The PI digits

The number Pi is a mathematical constant defined by the ratio of a circle's circumference to its diameter. Pi is commonly approximated as 3.14159265. Being an irrational number, Pi cannot be expressed exactly as a fraction. The digits of Pi appear to be randomly distributed, however no proof of this has been discovered[21].

If we take the 100 first digits of Pi we obtain the sequence presented in figure 8. And the 3-dimensional representation of the corresponding attractor is shown in figure 9.

1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9,
3, 2, 3, 8, 4, 6, 2, 6, 4, 3, 3, 8, 3, 2,
7, 9, 5, 0, 2, 8, 8, 4, 1, 9, 7, 1, 6, 9,
3, 9, 9, 3, 7, 5, 1, 0, 5, 8, 2, 0, 9, 7,
4, 9, 4, 4, 5, 9, 2, 3, 0, 7, 8, 1, 6, 4,
0, 6, 2, 8, 6, 2, 0, 8, 9, 9, 8, 6, 2, 8,
0, 3, 4, 8, 2, 5, 3, 4, 2, 1, 1, 7, 0, 6,
7, 9

Figure 8. 100 first digits of Pi

Figure 9. Attractor of the Pi digits - view 45°

In figure 9, we can see that the Hilbert curve shows a color distribution that seems random. However the 3-dimensional view shows a cloud of point close to the real random but with some ordered alignments. It appears there that the Pi digits are not truly random.

Figure 10. Attractor of the Pi digits (grouping by 4) - view 45°

In fact, this effect of ordered alignments is due to the limit of the Pi digits. Indeed, they are between 0 and 9 and the various possible combinations of these decimals in the subtractions which we make to calculate the coordinates x , y , and z are limited. By grouping them by 4, we obtain one distribution of the results which stages between 0 and 9999. The obtained result is more relevant. The figure 10 she shows then a perfectly random cloud. We thus have a first approach of the confirmation of the hypothesis of the alea of the digits of the Pi number.

4.3 Linear congruence generator

The linear congruence generator produces a pseudo-random sequence of integers $x_1, x_2, x_3, \dots$ according to the following linear recurrence [12, 10]:

$$x_{n+1} = ax_n + b \pmod{m} \quad | \quad n \geq 0$$

The integers a , b and m are the parameters that characterize the generator and x_0 is the seed. An example of generated sequence with $a = 5$, $b = 3$ and $m = 4096$ is given below :

The attractor for the $x_{n+1} = 5x_n + 3 \pmod{4096}$ function is shown in figures 12 and 13.

772, 3863, 2934, 2385, 3736, 2299,
3306, 149, 748, 3743, 2334, 3481,
1024, 1027, 1042, 1117, 1492, 3367,
454, 2273, 3176, 3595, 1594, 3877,
3004, 2735, 1390, 2857, 2000, 1811,
866, 237, 1188, 1847, 1046, 1137

Figure 11. Sample sequences of linear congruence generator

Figure 12. Attractor of the linear congruence generator - axe $y \rightarrow y'$

The sample size is 16384, the minimum is 0 and the maximum 4095 the average value is 2047.50.

The color distribution in the Hilbert curve seems random.

But, unlike true random generator that we have taken as a reference, the distribution of points in figures 12 and 13 has a specific pattern in the form of lines aligned on several levels. These lines correspond to the period of the algorithm.

Finally, we note that only the red predominantly colors appear, it means that only the end of the spectrum is visible. The first points are covered by the latest, the algorithm therefore provides several times the same values.

This type of generator is predictable and is not suitable for cryptographic use. Indeed, from a partial sequence, and without knowledge of the parameters a , b and m , it is possible to reconstruct the rest of the sequence. Moreover, in this case, the both representation – in two and three-dimensional – permit us to refine the randomness property of this algorithm.

Figure 13. Attractor of the linear congruence generator - axe $z \rightarrow z'$

4.4 Nonlinear recurrence generator

For our next example, we use a nonlinear recurrence generator. The latter produces a pseudorandom sequence of integers $x_1, x_2, x_3, \dots$ according to the following recurrence relation [20]:

$$x_{n+1} = \lambda x_n (1 - x_n) \quad | \quad n \geq 0$$

This suite called “logistic” leads, if $\lambda > 3,56995$, to a chaotic suite. Logistic suite is used to model the size of a biological population over generations [11]. An example of generated sequence is given below:

451098855224, 451068728783,
451038599037, 451008465983,
450978329622, 450948189954,
450918046977, 450887900692,
450857751097, 450827598193,
450797441977, 450767282451,

The attractor for this function with the following variables $\lambda = 3.8$ and $x_0 = 0.7364738523$ is given in figure 14.

Figure 14. Attractor of the nonlinear recurrence generator - axe $y \rightarrow y'$

The figure 14 also shows a specific pattern that makes it an unsafe PRBG for cryptographic application. However, unlike the linear congruence generator, we have no more appearance of periodic pattern. The generated data are not random in the first iterations (green color) then appear to become more random so with the increase of the number of iterations.

In this case, studying the Hilbert curve is interesting because the color distribution seems random at the beginning but becomes quickly uniform at the end of the list (same color).

In our example, the sample size is 16384, the minimum is 184264892666.96 and the maximum value 948850856936.47 sample average is 631015962902.33.

4.5 Blum-Blum-Shub generator

The pseudorandom bit generator Blum-Blum-Shub is a computationally secure PRBG as factoring large prime numbers remains insoluble. This generator produces a pseudorandom bit sequence in accordance with the following algorithm:

- generate two great number of Blum-Blum p and q and compute $n = pq$. A Blum-Blum's number is a prime number congruent to 3 modulo 4;

- choose a seed s in the range $[1, n - 1]$ such that $\gcd(s, n) = 1$;
- compute $x_0 = s^2 \pmod{n}$;
- the sequence is defined as $x_{i+1} = x_i^2 \pmod{n}$;
- if x_i is even then $z_n = 0$ and if x_n is odd then $z_n = 1$;
- the output of the sequence is $z_1, z_2, z_3, \dots$

The attractor of this function is shown in figure 15. To facilitate the implementation of our presentation, we did not apply the final step of recovering the least significant bit. Finally, we note that the attractor obtained is spread over the three axes in the form of a homogeneous cloud. We find the same form as in the case of the true alea. Actually, the Blum-Blum-Generator is cryptographically secure assuming the intractability of the quadratic residuosity problem [1]. In our example, the sample size is 16384, the minimum is 56782881.0 and the maximum value is 512462853845.0, the sample average is 257668663192.64.

Figure 15. Attractor for the Blum-Blum-Shub generator - view 45°

5 Cryptographic algorithms

5.1 RC4 attractor

To calculate the attractor of the RC4 algorithm, we start from the sequence of integers $n = 2^{105} + i$ with $i = [1 \dots 60\,000]$ that we convert in 128-bit blocks. We then encrypt each block with the same encryption 64-bit key:

$$k = 0101010101010101 \quad (1)$$

Finally, we convert the result to integers. So we start from a linear sequence of integers to get a pseudorandom sequence of integers. The following list shows the result for the first 8 entries in the sequence:

$$\begin{aligned} RC4_k(2^{105} + 1) &=_{16} (06080c0e182029293933495766768782) \\ &=_{10} (8017150855153813117600873193181448066) \\ RC4_k(2^{105} + 2) &=_{16} (06080c0e182029293933495766768781) \\ &=_{10} (8017150855153813117600873193181448065) \\ RC4_k(2^{105} + 3) &=_{16} (06080c0e182029293933495766768780) \\ &=_{10} (8017150855153813117600873193181448064) \\ RC4_k(2^{105} + 4) &=_{16} (06080c0e182029293933495766768787) \\ &=_{10} (8017150855153813117600873193181448071) \\ RC4_k(2^{105} + 5) &=_{16} (06080c0e182029293933495766768786) \\ &=_{10} (8017150855153813117600873193181448070) \\ RC4_k(2^{105} + 6) &=_{16} (06080c0e182029293933495766768785) \\ &=_{10} (8017150855153813117600873193181448069) \\ RC4_k(2^{105} + 7) &=_{16} (06080c0e182029293933495766768784) \\ &=_{10} (8017150855153813117600873193181448068) \\ RC4_k(2^{105} + 8) &=_{16} (06080c0e18202929393349576676878b) \\ &=_{10} (8017150855153813117600873193181448075) \end{aligned}$$

We developed a suite of specific tools (see Appendix A.2) to generate the list and then display it in a three dimensional environment like the approach outlined above for PRBG. We get the figure 16.

Figure 16. RC4 attractor - axe $x \rightarrow x'$

The attractor of RC4 is very sparse and boils down to three red balls. The color is important, it is the color of the end of the panel color. So we can deduce that all points of the attractor have the same coordinates. We are far from a random distribution of the space like with the Blum-Blum-Shub generator.

If we change the key we obtain different kinds of attractor: figure 17 and figure 18. So we have a close link between the obtained attractor and the key used. The Hilbert curve representation of RC4 shows the same link between the attractor and the key used but more important it shows that this algorithm does not seems really close to the perfect randomness.

5.2 Enigma machine attractor

The Enigma machine was invented by German engineer Arthur Scherbius in 1927 [8]. The Model A was quickly abandoned in favor of the model B, of the size of a typewriter then by a portable version equipped with lamps indicators with the model C. The Enigma machine and the company Scherbius founded for marketing, will vegetate during the inter-war. The Wehrmacht buys some copies for evaluation. Its only when Hitler began to rearm Germany than the cryptology experts of the Wehrmacht decide to adopt it and to equip the German army.

The Enigma machine is a polyalphabetic encryption machine. Its first version has 3 rotors each with 26 positions [12]. The R_1 rotor rotates every time a key is pressed,

By zooming in the graph (figure 20 page 177), we are seeing that points are grouped in clusters and there colors are similar. By counting the points inside clusters we find they contains approximately 26 points, ie a point for each character of alphabet. We have no explanation for this fact, it needs more investigations.

By using the same tools as for the RC4 algorithms, we get the figures 21 and 22.

Instead of the Blum-Blum-Shub generator we find that the attractor of AES takes the form of a cloud of points distributed over the three axes under the form of cubes. It differs, in this, from the attractor of the true alea. Nevertheless we can reasonably infer that the AES is an encryption algorithm designed as a pseudorandom bit generator close to the true hazard.

Michel Dubois (c) 2014, dt: 0.005, FPS: 45.41, Nbr elts: 999999
Min: 41157210695273507969978613057219
Max: 340281804530340149929531690668256442924
Average: 170223607434846015831773057926101596258

Figure 21. AES attractor - axe $x \rightarrow x'$

Michel Dubois (c) 2014, dt: 0.005, FPS: 45.41, Nbr elts: 999999
Min: 41157210695273507969978613057219
Max: 340281804530340149929531690668256442924
Average: 170223607434846015831773057926101596258

Figure 22. AES attractor - view $z \rightarrow z'$

After graphically analysed the behaviour of the AES, we will try to see how the algorithm behaves with different encryption keys. For this, we have reproduced the process described above using the same initial sequence and encrypting it with the two keys:

$$k_1 = 80000000000000000000000000000000 \quad (3)$$

$$k_2 = 40000000000000000000000000000000 \quad (4)$$

These two keys differ only on 1 bit over the 128 bits composing them.

Then assigning a different color to each attractor, we can compare them two by two. The result is shown in figures 23 and 24. To generate these figures we calculated the encryption of a sequence of 450000 identical blocks.

By performing the comparison of these two attractors, we find that they fit together tightly by uniformly covering the three dimensions. However, it is not possible to distinguish a possible correlation between these two sets of elements. To help visualize a possible correlation between the two sets

Michel Dubois (c) 2014, dt: 0.005, FPS: 93.87, Nbr elts: 899998
Min: 6256105871831500249891902403748
Max: 340282309019111759086093123132388536313
Average: 170301660500723989151303499652265831589

Figure 23. Attractors of two AES ciphers with different keys - axe $x \rightarrow x'$

Michel Dubois (c) 2014, dt: 0.005, FPS: 93.87, Nbr elts: 899998
Min: 6256105871831500249891902403748
Max: 340282309019111759086093123132388536313
Average: 170301660500723989151303499652265831589

Figure 24. Attractors of two AES ciphers with different keys - view $z \rightarrow z'$

of encrypted blocks, we will represent the same graph but by joining two by two the corresponding points of the same input. Thus, the point representing the encrypted block with the key k_1 of the first block of the sequence will be linked to the corresponding point in the encrypted block with the key k_2 of the same block in the clear.

The result is shown in figures 25 and 26. In the aims that the graphs are readable, we have reduced the size of the sample to 255 entries. The straight path connecting the points is messy and shows no identifiable pattern.

dt: 0.005, FPS: 60.16, Nbr elts: 500
Min: 322442391136856921185270035730946988
Max: 339043053166272882280386357705447036904
Average: 178684443684029902113715736432727716727

Figure 25. Correlation between two AES attractors - view 45°

We can reasonably deduce that there is no correlation between the encryption of the same clear block with different

Figure 26. Correlation between two AES attractors - axe $z \rightarrow z'$

encryption keys. Calculating Minkowski distances [18] with $p = 2$ for each point of the graph confirms this result. Indeed, the list of distances, whose first inputs are shown below, behave as a random sequence. The attractor arising from the representation of these distances in three dimensions (see figure 27) has the form of a cloud whose specific form is close to that a diamond. It would be interesting to try to understand what justifies this form, but it is not the subject of this work.

12026847709294981120, 12026869209288865792,
19971070890283577344, 26113153353646714880,
25972774069680922624, 14954846745435789312,
7391670208430327808, 1673631091276342528,
2444035869612603904, 2802878496097698304,
2426843600751804928, 1753123651095763712,
9810861268161230848, 10326031878050832384,
16430236778313869312, 14585735298828013568,
14873569053264308224, 13174990125624895488

Figure 27. Attractor of distances between two encryption process with the AES - view 45°

6 Conclusion

Our analysis of the hazard produced by PRBG and in particular by the cryptographic algorithms, although visually speaking, is not sufficient to prove that the sequences produced by its algorithms are statistically random.

However, viewing of the hazard, as we have shown, has two attractive advantages: the first is that it can detect pat-

terns and thus here some problems since we are seeking randomness, and second is the fact that our brain has more ability to quickly distinguish two different pictures, our approach allows us to more easily recognize and differentiate PRBG from another. Thus, just by displaying in environments in two and three dimensions the output of a random sequences, we have the opportunity to detect possible patterns and identify the PRBG used to generate the sequence. This approach also allows us to realize a first and quick analysis of potential biases on unknown cryptographic algorithms.

We developed some tools that offer the opportunities to apply our approach on some PRBG and cryptographic algorithms. Our aim is to implement more and more of these algorithms to construct a database of attractors that can be used to attribute the proper PRBG or cryptographic algorithm to a random number sequence.

Acknowledgements

The authors wish to thank Alexey Zhukov of the Bauman Moscow State Technical University for his valuable comments to this research work.

A Listings of programs

The entire program code developed during this article is available on GitHub (<https://github.com/archoad>). These programs are all under GPL v3.

A.1 PRBG generating and viewing in 3D

Available at the link <https://github.com/archoad/PRBG-3D>. To run this suite of programs requires a C compiler and OpenSSL and OpenGL libraries.

A.2 Encryption algorithms and 3D visualization

Available at the link <https://github.com/archoad/VisCipher3d>. To run this suite of programs requires a C compiler and the OpenSSL libraries (using the libcrypto for encryption and the BIGNUM API) and OpenGL.

REFERENCES

- [1] L. Blum, M. Blum, and M. Shub. Comparison of two pseudo-random number generators. *Advances in Cryptology*, 82:61–78, 1983.
- [2] G. Conti and S. Bratus. Voyage of the reverser: A visual study of binary species. *Black Hat USA*, 2010.
- [3] G. Conti, E. Dean, M. Sinda, and B. Sangster. Visual reverse engineering of binary and data files. *Workshop on Visualization for Computer Security (VizSEC)*, 2008.
- [4] C. Domas. The future of re: Dynamic binary visualization. *Reverse engineering conference (REcon)*, 2013.
- [5] E. Filiol. A new statistical testing for symmetric ciphers and hash functions. *Proceedings of Information and Communications Security 2002*, 2513:342–353, 2002.

- [6] C. Hamilton. Compact hilbert indices. *Technical report CS-2006-07*, 2006.
- [7] D. Hilbert. Ueber die stetige abbildung einer line auf ein fichenstck. *Mathematische Annalen*, 38:459–460, 1891. <http://dx.doi.org/10.1007/BF01199431>.
- [8] D. Kahn. *The Codebreakers*. Scribner, 1996.
- [9] M. Kennel, R. Brown, and H. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45:3403–3411, 1992.
- [10] D. E. Knuth. *The Art of Computer Programming*, volume Volume 2 - Seminumerical Algorithms. Addison Wesley, 1981.
- [11] R. May. Simple mathematical models with very complicated dynamics. *Nature*, 261:459–467, 1976. <http://abel.harvard.edu/archive/118r-spring-05/docs/may.pdf>.
- [12] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.
- [13] K. Nance, S. Tricaud, and P. Saad. Visualizing network activity using parallel coordinates. *44th Hawaii International Conference on System Sciences*, 2011.
- [14] N. Packard, J. Crutchfield, D. Farmer, and R. Shaw. Geometry from a time series. *Physical Review Letters*, 45:712–716, 1980.
- [15] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890. <http://dx.doi.org/10.1007/BF01199438>.
- [16] S. Tricaud and P. Saad. Applied parallel coordinates for logs and network traffic attack analysis. *Journal in Computer Virology*, 6:1–29, 2010.
- [17] J. von Neumann. Various techniques used in connection with random digits. *Monte Carlo Method*, 12:36–38, 1951.
- [18] E. Weisstein. Distance, 2014. <http://mathworld.wolfram.com/Distance.html>.
- [19] E. Weisstein. Hilbert curve, 2014. <http://mathworld.wolfram.com/HilbertCurve.html>.
- [20] E. Weisstein. Logistic map, 2014. <http://mathworld.wolfram.com/LogisticMap.html>.
- [21] E. Weisstein. Pi digits, 2014. <http://mathworld.wolfram.com/PiDigits.html>.
- [22] M. Zalewski. Strange attractors and tcp/ip sequence number analysis. *lcamtuf.coredump.cx*, 2001. <http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>.
- [23] M. Zalewski. Strange attractors and tcp/ip sequence number analysis - one year later. *lcamtuf.coredump.cx*, 2002. <http://lcamtuf.coredump.cx/newtcp/>.