Analysis of PRNGs with Large State Spaces and Structural Improvements

Gabriele Spenger, Jörg Keller
Faculty of Mathematics and Computer Science
FernUniversität in Hagen
58084 Hagen, Germany
gabriele@spenger.org, joerg.keller@fernuni-hagen.de

Abstract

The security of cryptographic functions such as pseudo random number generators (PRNGs) can usually not be mathematically proven. Instead, statistical properties of the generator are commonly evaluated using standardized test batteries on a limited number of output values. This paper demonstrates that valuable additional information about the properties of the algorithm can be gathered by analyzing the state space. As the state space for practical use cases is usually huge, two approaches are presented to make this analysis manageable. Results for a practical application of these approaches to the algorithms AKARI and A5/1 are provided, giving new insights about the suitability of these PRNGs for security applications.

Key words: PRNG; Security; AKARI; A5/1; Improvement

1. Introduction

The growth of electronic communication over the last decades and the development of technologies like Radio Frequency Identification (RFID) has led to a large interest in data security. Sending sensitive information over communication channels that are accessible by attackers, e.g. the Internet or the air in case of radio transmission requires measures to secure the privacy as well as the integrity of the transmitted data. In order to achieve this, cryptographic protocols have been developed and standardized that make use of cryptographic base functions like symmetric or asymmetric encryption, hashing and pseudo random number generators (PRNGs).

There are many standardized cryptographic functions that provide at least some confidence in their suitability for a given application, because they have resisted attacks and analysis over the years. These standardized functions are at least well tested and have usually been subject for a lot of scientific work trying to find any kinds of weaknesses before their standardization. But there are also occasions where the standards do not provide the right tools for an application. One example is the application on low cost RFID transmitters, which only allow a very low complexity of the algorithms due to the cost restraints (the cost is mostly driven by the required chip area [1]). In these cases, it might be required to use a non-standardized algorithm. This implies the danger of choosing an algorithm that does not provide the necessary security.

Besides the algorithms themselves also a couple of methods for the assessment of the suitability of cryptographic functions have been standardized. Examples for methods for assessing the security of PRNGs are the Marsaglia suite of Tests of Randomness [2], the Entacher collection of selected pseudorandom number generators [3] and the NIST test suite [4]. While these test batteries provide valuable information about the statistical properties of a given PRNG, they do not examine the complete state space of the algorithms. Instead their analysis is based on the statistical properties of a limited number of output values. Algorithms that do not meet the pass criteria of the test batteries are usually not suited for security applications, because the statistical properties might be exploited by an attacker to predict future generated output values. But algorithms that do pass the criteria do not necessarily provide high security. This fact is typically included as advice in the documentation of the test batteries, e.g. in the abstract of the NIST publication. Further examples for non-statistical properties of PRNGS that are suitable for cryptographic purposes include Forward Secrecy and Backward Secrecy (high difficulty to calculate past or future generated outputs or states from a compromised current random number) [5].

In order to fully assess the quality of a PRNG, the full state space needs to be examined. Unfortunately, this is not feasible for realistic use cases. While algorithms exist that have a state with the same size as the pseudo random number that is generated, this is not the common case. Typically, PRNGs make use of a state that has a significantly greater size than the generated number. Examples for such PRNGs (PRNGs and stream ciphers can be considered equal in this context) are the A5/1 algorithm [6] with a state length of 64 bits for an output of a single pseudo random bit, or the AKARI algorithm with 64 bits [7]. Both of these algorithms have been designed specifically for the use in low cost RFID transmitters and have a low algorithmic complexity. The state space for both of these algorithms has a size of 2^{64}, which is a number of values that can neither be stored nor traversed in a reasonable amount of time.

This paper presents different approaches to assess the security of such PRNGs with huge state spaces. After the general discussion of the approaches in Section 2, A5/1 and AKARI are examined as examples for real world cryptographic PRNGs in Section 3. While no solution can be given to reliably ensure the security of PRNG algorithms, the methods still prove to be valuable by detecting possible weaknesses that go beyond the standardized test batteries. The decision if these weaknesses render the algorithm inappropriate depends heavily on the given application. It might be that the weakness does generally not show up in the application, or that it can be avoided by a careful choice of start values or other parameters. Nevertheless, it is important to know these weaknesses so that it is possible to select the right algorithm for the right application.

2. State space analysis methods

Pseudo Random Number Generators are generally deterministic state transition functions f: M → M mapping a finite state space to itself as long as they do not receive new seed or entropy bits. Every output of the PRNG results in a state transition. This means that the generated sequences of pseudo random numbers are periodic. The output is deterministic and dependent on the state. Therefore, only the state is considered in the following. If a single state is interpreted as a node and the transition between a state and its unique successor state is interpreted as an edge, the result is a directed graph $G_f = (V;E)$ with $V:= M$ and $E:=\{(x;f(x)) \mid x \in M\}$. The structure of the generated graph provides information about the behavior of the pseudo random generator. For non-bijective transition functions, the graph typically consists of several weakly connected components. Each of these components consists of one cycle and generally several trees with roots located on the cycle. Figure 1 depicts the structure of a component.

Properties of the graph include the number and size of the connected components, length of the cycles and maximum depth of the trees. In order to identify all connected components of a graph, the complete state space would have to be analyzed, e.g. by a depth or breadth first search. Alternatively, only a part of the state space could be analyzed, accepting the fact that one or several components might be missed. A possibility to reduce the size of the entire state space, so that it can be analyzed completely, is to reduce the length of the state variables of the algorithm and accordingly the length of the period of the generated pseudo random numbers. In the following sections both approaches, the partial analysis of the state space and the reduction of the state space are discussed in more detail.

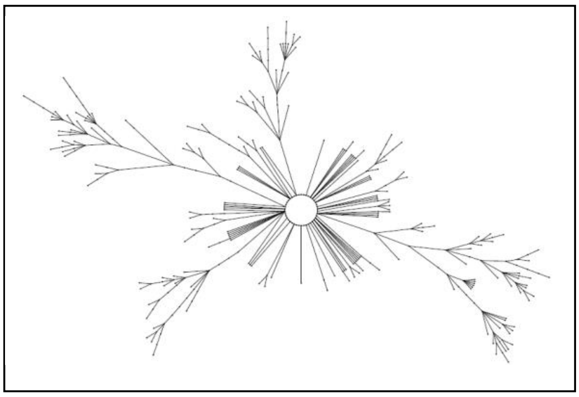

Figure 1. A typical connected component of a state transition graph [8]

2.1 Reducing the state space size

One way to analyze the state space of a PRNG is to run a depth-first search on the directed graph [9]. As each node has exactly one outgoing edge, an iterative formulation is sufficient. Each path in the graph is first treated as a new component. If it meets a node from an already known component, the nodes in the path are re-labeled for that component. This allows finding all connected components, their sizes, and the size of their cycles.

To reduce the required memory for large graphs, only a limited number of components plus a “remainder” component is allowed. Under the assumption that only some large components exist which are found first (see 2.3), the remainder component only comprises several small components, so that the inaccuracy is small.

For the analysis of 30 components (plus a number for unlabeled components and a number for the common rest component) the storage of the component number needs 5 bit. For the A5/1 algorithm the state consists of three state variables with 19, 22 and 23 bits length. This results in a memory requirement of $2^{64} \cdot 5$ bit for the DFS analysis approach, or about $11.5 \cdot 10^9$ GB, which is far too much even for the most powerful supercomputers. The same is true for the AKARI with its two 32-bit state variables also resulting in a 64-bit state. For algorithms with such large state spaces an additional approach is needed to reduce the memory requirements.

One option is to reduce the word length of the algorithm under the assumption that the basic properties do not change too much. In the case of AKARI, instead of using 32-bit state variables one could e.g. use 16-bit state variables. This results in a state size of N = 2·16 bit = 32 bit or 2.6 GB, which is easily addressable on a modern PC.

The question that remains unanswered is, in how far the analysis results for the algorithm running on a reduced word length can be transferred to a longer word length. While the practical analysis results presented in 3.1 indicate for the depth-first search that certain properties of the state graph seem to be identical for different word lengths, it is debatable if this is true for all cases. Obviously there are cases, in which the algorithm will behave completely differently, e.g. for rotating bit shifts that degenerate to no operations if the shift length is equal to the word length.

2.2 Sampling the State Space

An alternative way to analyze the state space of a large graph is to use a realistic word length and to sample the state space instead of analyzing it completely. Obviously this has the risk of missing a weakness of the state graph, as the sampling results in only a very small part of the state space being analyzed. For this reason, this approach is not suitable for positively approving the security of an algorithm. Instead it can be used to randomly find weaknesses, which might be sufficient to make an algorithm unsuited for a given use case. If random sampling of the state space shows a weakness, chances might be good that more instances of the same weakness exist in the state space.

The approach described in 2.1 depends on the analysis of the complete state space, as every node of a cycle needs to be marked with a component number in order to be able to identify cycle lengths. While cycle lengths can be found before every node in the state space has been marked, the memory for all nodes needs to be allocated for this approach, to allow marking any node as visited. This prevents the analysis of component cycle lengths for realistic word lengths. In [10] a method is presented to avoid storing a number for every single node. The idea that was also used previously e.g. in [11], is to only store certain nodes while traversing the tree. If only the nodes at distances 2^n from the start value are stored (n=1, 2 …), the required memory usage is significantly reduced. For an analysis run starting at any given node and running for N steps until a cycle is detected, this means that a maximum of $\log_2(N)-1$ nodes need to be stored. This results in a low memory requirement even for extremely long runs. Still, the cycle at the end of a path can be detected if one of the stored nodes is reached again, and the cycle length can be computed. Figure 2 shows the process of cycle detection.

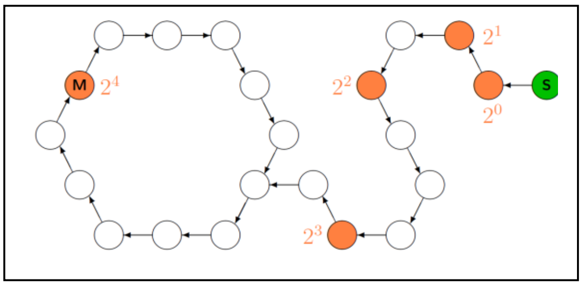

Figure 2. A cycle is detected [10]

Storing the nodes in these increasingly higher distances means that other aspects of the tree structure are harder to determine. The root of a tree on a cycle can only be determined by performing a simultaneous run from the last node outside of the cycle M and a congruential node K_N with the same distance to the node that has been identified to be on a cycle (see Figure 3). By comparing the nodes during this simultaneous run, the entry node E can be found, as this is the first node that the simultaneous runs reach at the same time.

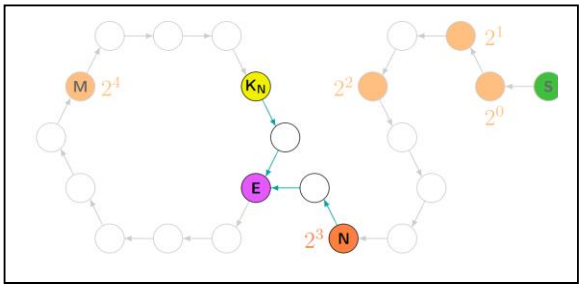

Figure 3. Finding the cycle entry point [10]

This method allows performing an analysis on an arbitrary large state space. The memory required for the analysis is not only small, but also only logarithmically growing with the size of the state space. Still, the required time for the analysis prevents to examine the complete graph for realistic word lengths. But it allows taking arbitrary, randomly chosen start values and walking through the graph starting from these. This way, the state space is sampled and the resulting tree and cycle structure for these samples might provide valuable insights with respect to the security of the algorithm. Also, the sizes of the connected components can be guessed from the sampling within a confidence interval.

2.3 Metrics for a "good" PRNG

While information about the structure of a state transition tree provides valuable insight about the properties of a PRNG, this information needs to be evaluated somehow. The question how the structure looks like for a "good" PRNG and how for a "bad" one needs to be answered in order to be able to compare PRNGs reasonably. Obviously, it depends heavily on the application, what "good" and "bad" means and what the thresholds for such a classification are. Therefore, the definition of such criteria is out of the scope of this document, but some reasonable guidance is provided in the following. The properties that can easily be deducted from the structure analysis are:

- number of components
- cycle lengths of the components
- size of the components

It is desirable that a PRNG has a small number of components. The fewer components are present in the state structure, the larger can the components and their cycle lengths be. According to [12] the expected number of components for a randomly chosen non-bijective state transition function for a set M with n elements is $1/2 \cdot \log(n)$. For an invertible state transition function, the expected value is ln(n) [13]. For a "good" PRNG this value should be equal or less than the expected value.

Furthermore, it is desirable that the number of nodes on a cycle (which is equivalent to the cycle length) is as large as possible. This results in a high number of steps until the states and the produced pseudo random numbers are repeated. According to [12] the expected cycle length for non-bijective state transition functions is $\mathrm{sqrt}(\pi \cdot n/2)$. The largest cycle length should be about $c_1 \cdot \mathrm{sqrt}(n)$ with $c_1 \approx 0.78248$. It follows that the expected size of the largest cycle is approximately $0.78248 \cdot \mathrm{sqrt}(n)$. A "good" PRNG should probably have one cycle with at least an according size or larger. For bijective state transition functions the expected length of the largest cycle is $(1-1/e) \cdot n = 0.632 \cdot n$.

Finally, the number of nodes in any component (equivalent to the size of the component) should be as large as possible. Ideally the state structure consists of a single component containing all nodes of the graph. According to [12] the expected number of nodes of the largest component for a non-bijective state transition function is $c_2 \cdot n$ with $c_2 \approx 0.75782 \cdot n$. PRNGs with bijective state transition functions consist of cycles only [9] and therefore the cycle lengths are equal to the component sizes.

3. Practical application

Two PRNGs have been analyzed using the approaches described above in order to evaluate the benefit of an additional evaluation of properties of the state space structure. AKARI [7] is a low complexity PRNG with good statistical properties as demonstrated by the authors with results of the ENT, Diehard and NIST test batteries. A5/1 is the stream cipher used in GSM mobile phones.

3.1 Analysis of AKARI-1

AKARI in its variants AKARI-1 and AKARI-2, is a lightweight PRNG that has been designed by Pedro Peris López et al. specifically with a use on RFID transponders in mind. The design criteria were good statistical properties and suitability for security applications with the hardware limitations of RFID transponders taken into consideration. Further requirements were a tiny footprint, high throughput and a low power consumption. The approach taken was to combine a T-function as proposed in [14] with a non-linear filter function based on a composition of extremely light operands. The state consists of two state variables with a length of 32 bits resulting in a state length of 64 bits. The analysis results presented by the authors for the ENT, Diehard and NIST test batteries show promising results, leading to the conclusion that the algorithm is well suited for its intended use in security applications.

The first approach was to analyze the state space of AKARI-1 for a reduced word length. Table 1 shows the result of the analysis for a word length of 14 bits respectively a state length of 28 bits: 30 components have been identified, all of them with the same size being identical to the cycle length. This clearly indicates that the algorithm is bijective, at least for this word length.

Table 1. Analysis results of AKARI for 7 bit output

Component	*Cycle Length*	*Size*	*Relative Size*
1	16384	16384	0.0061%
2	16384	16384	0.0061%
3	16384	16384	0.0061%
4	16384	16384	0.0061%
5	16384	16384	0.0061%
…	…	…	…
27	16384	16384	0.0061%
28	16384	16384	0.0061%
29	16384	16384	0.0061%
30	16384	16384	0.0061%
Rest	n/a	267943936	99.8169%

In order to verify if the algorithm behaves the same for the word length of 16 bit intended by the authors of AKARI, a sampled analysis was performed. The result showed that the cycle lengths of the sampled components for AKARI-1 are accordingly 2^{32}=4294967296, so the behavior of the algorithm appears to be the same for this word length.

Due to the bijectivity of AKARI, an inverse of the transition function might be derivable and thus backward security might be compromisable if an internal state is leaked. As a side note, the cycle lengths are far worse than what is proposed for a "good" PRNG in 2.3. Still, AKARI appears to be a good choice for cryptographic RFID use cases, as it easily passes the test batteries and shows reasonable cycle lengths.

3.2 Analysis of A5/1

The A5/1 stream cipher consists of three different irregularly clocked linear feedback shift registers (LFSRs) that are combined via a clock control. Whenever a register is clocked, the feedback bits (e.g. 13, 16, 17, and 18 for R1) are XORed and inserted into bit 0 after a left shift. The feedback taps of the three LFSRs in the A5/1 stream cipher were chosen in a way that the registers have maximum length periods, i.e. all other possible states of a register will be generated before a state will be generated for the second time.

To determine which register is clocked in each iteration of the A5/1 stream cipher, each register has a bit position marked as the clock tap (C1, C2 and C3) and a majority clock function takes these three bits as arguments. A register is clocked if its clock bit equals the majority value of the three clock bits. That means that either two or all three registers are clocked at the same time in each iteration. The values of the three clock bits form eight different combinations. For each clock bit there are two combinations where this bit differs from the other two causing it not to be clocked. Therefore, a single register is clocked in three out of four cases [8].

In the case of A5/1 the analysis approach with a reduced state length was not evaluated. Due to the asymmetric structure of the algorithm with its three registers of different length it seemed unclear, how the length could be reduced in a reasonable way. Therefore, only a sampled analysis with 3000 randomly chosen start values was performed. In the analysis of the A5/1 nearly all of the randomly chosen start values were part of an individual component of the state transition graph. Table 2 shows the analysis values for some of the identified components and the average of all analyzed values.

The analysis result shows a mostly consistent result over all components. All identified components have a reasonable cycle length, although it is still far worse than the criteria given in Section 2.3). Additionally, the maximum height of the tree that was found for the given start values is shown in the table. The components seem to be rather small, because for a component with a notable fraction of the nodes (e.g. 0.1% of the nodes, although this is much smaller than the expected largest component in a randomly chosen transition function) one would expect at least 3 start values from this component.

Only a small part of the state space of A5/1 has been sampled, so this result does not necessarily apply to the major part of the state space. Still, it can be concluded that due to the limited cycle length the algorithm is not suitable for applications requiring a large number of pseudo random numbers.

Table 2. Analysis results of A5/1 after 3000 start values

Component	*Cycle Length*	*Tree Height*
1	11185723	87232350
2	78294094	11141730
3	257249966	5058705
4	11184282	94273308
5	33553192	11932929
6	22369661	116796486
7	44738889	133192933
8	33554075	213048501
9	55924149	84497440
10	11184672	5124288
11	22371100	65284466
12	22369676	1153416
13	11185515	247669895
14	178959545	9097123
15	33553145	65218605
16	11184719	79567731
17	55924079	112327672
18	67108060	463350246
19	55923432	3736262
20	22369569	133299868
...	…	…
Average	35396755	101931457

4. Improvement by parameter switching

In some cases, it might be desirable to modify a given PRNG algorithm with low impact on the algorithm itself. Any modification of the algorithm needs a careful analysis of the new properties and creates a risk that a new weakness is introduced. One way to minimize this risk is to keep the algorithm as it is and only switch to a different parameterization of the algorithm at certain points in time. This allows “breaking out” of a cycle of the original algorithm and can increase the cycle length. Figure 4 illustrates this mechanism, with B being the break out point.

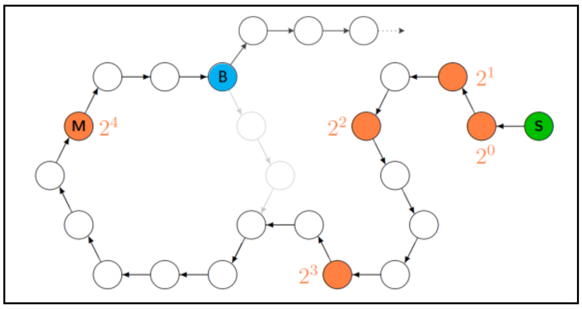

Figure 4. Break out of a cycle

One advantage of this approach is, that the algorithm that is applied stays the same and therefore no additional implementation is needed apart from the switching logic. This means that for a hardware implementation, the chip area stays nearly the same and the impact on the productions cost is small. Furthermore, the impact on the speed of the PRNG is minimal, as only the switching logic is added, which is typically much less complex than an PRNG algorithm implementation.

Different strategies for switching to the second parametrization are imaginable. Two simple approaches are to apply the break-out algorithm:

- once after a certain number N of PRNG numbers
- alternating between the two algorithms every N calls

In the following, these two approaches are evaluated for A5/1 as an example for a real world algorithm.

4.1 Application on A5/1

One of the weaknesses of A5/1 that have been identified in Section 3.2 is the comparably low typical cycle length. This might be a problem for applications in which many pseudo random numbers are needed. For such applications, increasing the cycle length of A5/1 will improve the suitability of this algorithm.

A5/1 does not use a key that allows a parameterization of the algorithm. One way to parametrize it, is the selection of the feedback taps, as shown in [6]. The work groups different feedback tap selections into two different classes: combinations that behave similar to the original tapping bits, and combinations that show a statistically worse behavior of the algorithm. According to the work, A5/1 generally exhibits a large state space structure independency of the selection of the feedback tapping bits. A large number of randomly selected feedback taps is analyzed in the work and shows a comparable behavior to the original selection of feedback taps.

For the following analysis, a random selection of feedback taps with similar behavior to the original algorithm was selected from [15] as alternative parameterization. This was done in order to impact the behavior of the algorithm as little as possible. For the breakout algorithm, instead of the original feedback taps tapR1 = 0x072000, tapR2 = 0x300000 and tapR3 = 0x700080 the following values have been used:

- tapR1 = 0x69BB6
- tapR2 = 0x22B545
- tapR3 = 0x6FFA53.

The results for this parametrizations are shown as example in the following. Other tapping bit selections have been analyzed in addition, showing identical results.

4.2 Single break out call

The minimum cycle length from Table 2 is 11184282. To avoid entering such a cycle, it seems reasonable to call the break out algorithm before the cycle can be completed. For the analysis, a break out after 2^{23} = 8388608 calls is chosen as an arbitrary value smaller than the minimum cycle length.

Although the analysis program was run for an extensive amount of time, it was not able to find a single cycle. This means that the cycle lengths of the modified algorithm are much longer than the ones of the original algorithm. In order to get a measureable result nevertheless, in the following the analysis was limited to a search of 10^9 steps. For all start values, this limit was reached, meaning that the cycles lengths are greater than 10^9 (see Table 3). This is a significant improvement compared to the original algorithm.

Table 3. Results for single break out after 2^{23} calls

Component	*Cycle Length*	*Tree Height*
1	>1000000000	Unknown
2	>1000000000	Unknown
3	>1000000000	Unknown
4	>1000000000	Unknown
5	>1000000000	Unknown
...	…	…
Average	>1000000000	Unknown

4.3 Alternating parametrization

The same analysis was run an A5/1 with alternating parametrization. After every 2^{23} calls, the parametrization was switched to the alternative one for the next 2^{23} calls. Then the parametrization was switched back to the original one and so on. Table 4 shows the results, which are identical to the results of the single break out method. The alternating approach is coming at a slightly higher complexity than the single break out approach, as an additional state information must be maintained that stores the currently applied variant of the parameters. For the single break out method, this state information is not needed. The impact on complexity is probably practically negligible, but nevertheless the single break out approach is probably slightly preferable out of this reason.

Table 4. Result for alternating algorithm after 2^{23} Calls

Component	*Cycle Length*	*Tree Height*
1	>1000000000	Unknown
2	>1000000000	Unknown
3	>1000000000	Unknown
4	>1000000000	Unknown
5	>1000000000	Unknown
...	…	…
Average	>1000000000	Unknown

5. Conclusion

Drawing conclusions about the security of cryptographic functions from analysis data is difficult. As it is typically not possible to prove the security mathematically, the available data can only be used to give hints on a usability of the algorithm for a certain use case. A single type of analysis is usually not sufficient, instead several types of analysis should be performed to increase the confidence in the properties of an algorithm.

For PRNGs a range of standardized test batteries exists that provides valuable information about the statistical properties. But additional structural analysis of the state space can add more information that increases the knowledge about the algorithm. Especially properties like cycle length, number of components or component size have an impact on the usability for a cryptographic application. These properties cannot easily be determined due to the large state space of such algorithms. The reduction of the word length or sampling of the state space can make this analysis manageable.

The results of this analysis performed on A5/1 and AKARI-1 as examples for real world cryptographic functions indicate that both approaches can give valuable information. For AKARI-1 both approaches hint to the fact that AKARI-1 is a bijective function. For A5/1 the sampled state space approach shows that the cycle lengths of the components appear to be smaller than ideally could be the case for an algorithm with the given state space size. This does not necessarily disqualify A5/1 for security related applications, but shows that for these applications a careful selection of a suitable algorithm is needed (in GSM, only a few hundred bits are used after each seeding).

A simple switching method that can improve the state space structure of a given PRNG is introduced. The practical application of this method to the A5/1 algorithm shows a significant increase of the cycle lengths and might be a promising way to improve the behavior with a low impact on the complexity of the implementation. The scope of this work is limited to the impact of this method to the state space structure and does not include any analysis of the generated output values of the PRNG.

Therefore, as future work it needs to be analyzed, if the modification of the original algorithms has an impact on the statistical properties of the generated output values. This can e.g. be verified by running standardized test suites and comparing the result to the original algorithms. Furthermore, other methods of switching could be investigated, including the use of a completely different algorithm for the break out. Another area of investigation is the approach on when to use the break out mechanism. Instead of applying it after a fixed number of calls, the selection could be based on properties of the internal state or the output of the algorithm. Furthermore, it might be worthwhile to evaluate the impact of higher numbers of parametrizations to switch between, instead of two.

6. References

[1] A. Y. Poschmann, “Lightweight cryptography: Cryptography engineering for a pervasive world,” Dissertation, Ruhr-Universität, Bochum, 2009.

[2] G. Marsaglia, “The marsaglia random number cdrom including the diehard battery of tests of randomness,” 1995. [Online]. Available: http://www.stat.fsu.edu/pub/diehard/

[3] K. Entacher, “A collection of selected pseudorandom number generators with linear structures,” ACPC-Austrian Center for Parallel Computation, Tech. Rep., 1997.

[4] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker,“Statistical test suite for random and pseudorandom number generators for cryptographic applications: Special publication 800-22, revision 1a,” 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800222rev1a.pdf

[5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. CRC Press, 1996.

[6] J. Golic, “Cryptanalysis of alleged a5 stream cipher,” in Advances in Cryptology — EUROCRYPT ’97, ser. Lecture Notes in Computer Science, W. Fumy, Ed. Springer Berlin Heidelberg, 1997, vol. 1233, pp. 239–255. [Online]. Available: http://dx.doi.org/10.1007/3-540-69053-0 17

[7] H. Martin, E. S. Millan, L. Entrena, P. P. Lopez, and J. C. H. Castro, “Akari-x: A pseudorandom number generator for secure lightweight systems,” 11th IEEE International On-Line Testing Symposium, vol. 0, pp. 228–233, 2011.

[8] A. Beckmann, J. Fedorowicz, J. Keller, and U. Meyer, “A structural analysis of the a5/1 state transition graph,” in First Workshop on GRAPH Inspection and Traversal Engineering, ser. Electronic Proceedings in Theoretical Computer Science, vol. 99. Open Publishing Association, 2012, pp. 5–19.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms (3. ed.). MIT Press, 2009. [Online]. Available: http://mitpress.mit.edu/books/introduction-algorithms

[10] J. Keller, “Efficient sampling of the structure of crypto generators’ state transition graphs,” in EC2ND 2006. Springer London, 2007, pp. 3–12. [Online]. Available: http://dx.doi.org/10.1007/978-1-84628-750-3 1

[11] J. Keller, “Parallel exploration of the structure of random functions,” in Proceedings of the 6th Workshop Parallele Systeme und Algorithmen (PASA) in conjunction with the International Conference on Architecture of Computing Systems, ARCS. VDE, 2002.

[12] P. Flajolet and A. M. Odlyzko, “Random mapping statistics,” in Advances in Cryptology. Springer Verlag, 1990, pp. 329–354.

[13] R. Sedgewick and P. Flajolet, An Introduction to the Analysis of Algorithms. Reading and Mass. and USA: Addison-Wesley, 1996.

[14] A. Klimov and A. Shamir, “A new class of invertible mappings,” in Proceedings Cryptographic Hardware and Embedded Systems (CHES), Volume 2523 of the series Lecture Notes in Computer Science, pp 470-483, 2003

[15] J.-P. Ismer, “Zyklenstruktur des A5/1 Zustandsgraphen bei veränderten Schieberegistern,” Abschlussarbeit im Studiengang Bachelor in Informatik, FernUniversität in Hagen, 2014.