Probabilistic Cycle Detection for Schneier's Solitaire Keystream Algorithm

Wiem Tounsi*, Benjamin Justus*, Nora Cuppens-Boulahia*, Frédéric Cuppens*, Joaquin Garcia-Alfaro**
*Institut Mines-Telecom, Télécom Bretagne, Cesson Sévigné, France
**Institut Mines-Telecom, Télécom SudParis, CNRS Samovar UMR 5157, Evry, France
{*FirstName.SurName*}*@telecom-bretagne.eu* ; *joaquin.garcia-alfaro@acm.org*

***Abstract*—Pencil-and-paper ciphers are plausible solutions that could provide lightweight protection to the communication of resource-constrained devices. A good example in this category is Schneier's Solitaire cipher. In this paper, we propose a probabilistic solution that is able to estimate Solitaire's keystream cycle length. We also present a variation of Solitaire's original design, and evaluate the resulting construction in terms of predictability. We conduct statistical randomness tests on both the original design and the modified version based on the NIST randomness test suite. The results show that our approach improves the randomness of original Solitaire's output sequences.**

Keywords: ICT Security, Cryptography, Pencil-and-Paper Ciphers, Pseudo-Random Number Generators (PRNG), Cycle Detection, Randomness Evaluation.

I. Introduction

Resource-constrained devices, such as sensors and passive RFID tags, require the input of lightweight cipher designs in order to protect their communications [8], [12]. Pencil-and-paper ciphers, also known as hand ciphers or field ciphers by the military, are a plausible solution to address such a concern [16]. Pencil-and-paper ciphers are encryption methods that are operated by human beings, and they hold an appropriate balance between security and lightweight computation. According to studies like [11], humans and resource-constrained devices share similar sets of memory properties. For example, passive RFID tags are able to store only short secrets of about 32 to 128 bits of length in their volatile memory. Human can maintain about seven decimal digits in their immediate memory [13]. Nevertheless, electronic devices are more efficient at performing logical operations (e.g., AND, OR, and XOR-like operations), and at choosing random values.

Schneier's Solitaire cipher is a good example of a pencil-and-paper cipher that could be used to protect the communications of resource-constrained devices. It was designed by Bruce Schneier at the request of Neal Stephenson for the Cryptonomicon novel [18]. Originally called Pontifex in Stephenson's novel, the algorithm behind Solitaire aims at producing a practical cryptosystem calculated with an ordinary deck of playing cards. Despite decades of existence, few attacks or cryptanalysis techniques against Solitaire have been reported in the related literature. Questions such as the period of the Solitaire keystream algorithm, statistical properties of its generated pseudo-random sequences, and complexity of the Solitaire internal states remain open.

In this paper, we address some of the aforementioned questions about the keystream algorithm of Solitaire. Furthermore, we propose a slight variation of the original design which could be used as a pseudo-random number generator (PRNG). We present a cycle detection algorithm, which is computationally feasible when the deck size associated to the PRNG is relatively small. We conduct experimental results and show that it is possible to predict the maximum cycle length period for the Solitaire PRNG of any deck size. To verify the randomness quality of Solitaire keystream, we conduct as well some statistical evaluations of sample sequences generated by both Solitaire and our modified version. The evaluations are based on the NIST statistical test suite for random and pseudo-random number generators [17].

Paper organization: Section II introduces the cycle detection problem for PRNGs. Section III describes Solitaire's keystream algorithm. Section IV introduces our cycle estimation algorithm and reports our experimental results. Section V presents the NIST randomness evaluation results on the quality of the Solitaire PRNG. Section VI outlines related work. Section VII concludes the paper.

II. PRNGs and the Cycle Detection Problem

Deterministic pseudo-random number generators (PRNGs) are often used to generate identical sequences for both the sender and receiver sides of a communication exchange [19]. We assume that this sequence is generated synchronously in each party. We also assume that the same sequence is used to encrypt the messages following the one-time-pad principle [7]. Most PRNGs are designed based on a wide range of cryptographic primitives [3], [6]. Today's PRNGs are typically based on hash function or block cipher designs. Given the limited computing power of resource-constrained devices (e.g., passive RFID tags), block cipher based implementations are expected to be in use.

Solitaire's keystream algorithm can be used as a PRNG whose output is partitioned in blocks of length n. In block cipher-based PRNGs, the unpredictability of the generator is assured as long as a threshold N (related to the number of PRNG outputs) is not reached — within the same initial state. This threshold always exists due to the deterministic nature of PRNGs. Once N is reached, the outputs may enter into a cycle. This cycle is predefined in classical designs of block ciphers,

e.g., Cipher Block Chaining (CBC). It is typically equal to $2^{n/2}$ [5]. In the sequel, we aim at estimating the threshold value N as the maximum cycle length for the Solitaire PRNG.

III. The Solitaire Keystream Algorithm

We start with a concise description of the Solitaire keystream algorithm. Further details, including additional examples can be found in [18]. Solitaire generates its keystream using a deck of cards. Each card in the deck (54 cards, including the 2 jokers) corresponds to a number 1 to 54. For instance here, we use the bridge order of suits: clubs (1 - 13), diamonds (14 - 26), hearts (27 - 39), spades (40 - 52), Joker A = 53, Joker B = 54. To initialize the deck, we have to arrange the cards in the initial configuration that is the state. The state (or key) in this instance corresponds to an element of the permutation group S_{54}. To produce a single output from the keystream algorithm, the followings steps should be carried out:

1) Find the A joker. Move it one card down. If the joker is the bottom card of the deck, move it just below the top card.
2) Find the B joker. Move it two cards down. If the joker is the bottom card of the deck, move it just below the second card. If the joker is one up from the bottom card, move it just below the top card.
3) Perform a triple cut. That is, swap the cards above the first joker with the cards below the second joker.
4) Perform a count cut. Look at the bottom card. Count down from the top card that number. Cut after the card that you counted down to, leaving the bottom card on the bottom.
5) Find the output card (number). To do this, look at the top card. Count down that many cards. If one hits a joker, restart Step 1. This is the output card/number.

Solitaire is a block-permutation based algorithm. The keystream output starts repeating itself after a cycle is reached. The cycle length of the keystream is a crucial piece of information since key recovery becomes possible once an adversary is able to predict the keystream outputs.

A. Modeling by Group Theory

One natural way of modeling the Solitaire keystream algorithm is to give a group theoretic point of view. This line of investigation was extensively used in algebraic cryptanalysis of the Advance Encryption Standard (AES) [9]. Pogorelov and Pudovkina in [14] also used this approach in their investigation of the Solitaire algorithm.

In this theoretical framework, the state function $F(t)$ at t'th keystream number generation step is an element of the permutation group S_n, where n is the number of cards in a deck. The state function F is composed of four transformations: F_1, F_2, F_3, F_4 which correspond to the first four steps of the keystream algorithm. Clearly, if a cycle of length m exists, one would have

$$S(t+m) = S(t), \quad o(t+m) = o(t)$$

from some t'th number onwards. Here $o(t)$ denotes the t'th number in the keystream output, and $S(t)$ the corresponding Solitaire state. Thus, the task of detecting a cycle in the keystream is equivalent to finding a repetition of the Solitaire state at some regular time intervals. This approach turns out to be computationally feasible for us when the number of cards is less than 16 (i.e. $n \leq 16$, cf. Section IV-A).

Let $o(t)$ be a keystream element within a cycle. The cycle length which contains $o(t)$ is equal to the size of the orbit generated by $< S(t) >$ that acted under the transformation F. Pogorelov and Pudovkina give in [14] some algebraic descriptions of the orbits generated by individual actions F_i. At present, a complete algebraic description of those orbits generated by $< F = F_1, F_2, F_3, F_4 >$ does not exist in the literature. The best we can do theoretically is to write down the trivial bound (i.e., the maximum cycle length is necessarily less than the group order $|S_n| = n!$). There are some evidences, as shown in the next sections, that suggest the orbit sizes should grow exponentially with respect to n.

B. Uniform Distribution Property of Keystream Outputs

We start by validating the uniform distribution property of the Solitaire keystream outputs. Figures 1(a) and 1(b) plot the frequency of occurrence of all possible keystream output values. They correspond to, respectively, a deck of 10 cards (Figure 1(a)); and 66 cards (Figure 1(b)). The plots depict the average value associated to the frequency of occurrence of each card (excluding the jokers, which are never provided as a valid output). It also represents their 95% confidence intervals. To generate these two figures, we used 1,000 different sequences, composed of 100,000 keystream outputs each.

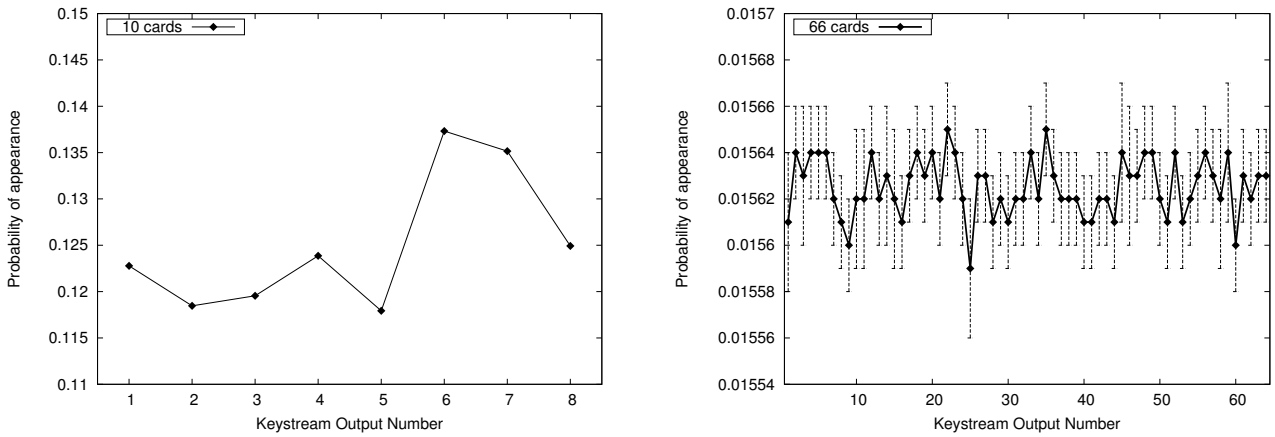
We can observe that the uniform distribution property emerges when the number of cards n gets large. Indeed, as can be seen from Figure 1(a), the probability distribution for a deck of 10 cards lies in the range between 0.118 and 0.137. This is to be compared with the perfect uniform distribution score $1/8 = 0.125$. For a deck of 66 cards, the discrepancy between the actual probability distribution values and the perfect theoretical value $1/64 = 0.015625$ becomes smaller (as shown in Figure 1(b)). The corresponding probability distribution range for 66 cards is between 0.01559 and 0.01565. Note that the results for confidence interval are too small to be shown in the plot of Figure 1(a), since they are in the order of 0.003% for the mean. In Section V, these results are also confirmed via the NIST randomness test suite.

IV. Cycle Estimation of the Solitaire Keystream

The security of Solitaire's keystream based PRNG (e.g., in order to prevent key-recovery attacks) depends on the cycle length of its keystream. In this section, we propose a probabilistic cycle detection method.

A. Cycle Detection Algorithm

The principle of our cycle detection algorithm (cf. Algorithm 1) is as follows: if a cycle of length m exists, then we



(a) 10 cards: 8 cards + 2 jokers

(b) 66 cards: 64 cards + 2 jokers

Fig. 1: Distribution of the Solitaire keystream numbers

would have:

$$S(t+m) = S(t), \quad o(t+m) = o(t)$$

from some t'th number onwards. Here $S(t)$ denotes the state of keystream generation at t'th step, and $o(t)$ the t'th number in a keystream output. For a fixed initial configuration, we could output $S(t), o(t), t \geq 1$ in a file. Figure 2 shows a sample output derived from a keystream generation for the case $n = 18$. Each line of the file corresponds to a $S(t)$, in conjunction with a $o(t)$ derived from the keystream generation algorithm. The left column represents the states of the permutation at the corresponding time steps.

(6,8,12,15,7,4,13,14,1,9,16,2,5,11,3,10) 9
(1,9,2,5,16,6,8,12,11,3,10,15,4,13,14,7) 2
(4,15,1,9,2,5,6,13,14,7,16,12,11,3,10,8) 12
(10,8,15,9,2,5,6,13,14,7,12,11,16,4,3,1) 13
(11,4,3,16,10,8,1,15,2,5,6,13,14,7,12,9) 2
(1,2,15,11,4,3,10,5,6,13,14,7,12,9,16,8) 6
(3,10,5,6,13,14,7,12,9,8,16,2,11,15,4,1) 13
(12,9,8,2,1,16,4,15,3,10,5,6,13,14,7,11) 5
(13,14,7,11,16,15,12,9,8,2,1,4,10,5,6,3) 13
(14,7,11,9,8,2,1,4,10,5,6,3,15,16,13,12) 3

Fig. 2: Solitaire keystream output: state and keystream

Algorithm 1 takes a keystream as an input, and outputs a cycle length when a cycle is found. The algorithm starts by randomly selecting a line L_1 at position i_1 where $1 \leq i_1 \leq \alpha$. Here, α is an algorithm parameter that will be explained later. The algorithm then will try to pattern match among the rest of the lines, and find $L_2, ..., L_\beta$ at the positions $i_2, i_3, ..., i_\beta$ within the keystream, which are identical to L_1, i.e., $S(i_1) = \cdots = S(i_\beta)$ and $o(i_1) = \cdots = o(i_\beta)$.

The random selection step in Algorithm 1 is justified because any keystream number is equally likely to be in a cycle (cf. Section III-B). The algorithm parameter β is a stop counter, which stops the algorithms after β number of matches are found. This parameter is designed to increase the algorithm efficiency, as now the algorithm may terminate well before the end of the keystream is reached. Computation efficiency turns out to be crucial in exhaustive search scenarios as those encountered in Section IV-B.

The parameter β is also significant in the following sense: the algorithm is able to terminate and concludes that a cycle has been found after β matches, because the probability that a non-cycle keystream number appears at equal distance β times is approximately $(\frac{1}{n})^\beta$, where n is the number of cards. Thus if we choose $\beta = 8$ when $n \geq 4$, then the chance that a *fake* cycle is detected instead of an authentic cycle is negligible. The approximation $1/n$ is valid because of the uniform distribution property among all the keystream outputs (cf. Section III-B).

The parameter α is usually chosen to be less than or equal to $\texttt{keystreamlength}/\beta$, so that the algorithm would not terminate before β matches are found in the keystream file. The parameter γ has the range $1 \leq \gamma \leq \beta$. It is an in-built parameter of the algorithm because sometimes a cycle does not start right after the first state repetition. The cycle verification step (line 8 in Algorithm 1) starts only after γ matches are found. Table I summarizes the parameter values α, β, γ we

TABLE I: Parameter choices for the cases $n = 4, 5, ..., 16$

n	α	β	γ
4	100	8	2
5	1,000	8	2
6	1,000	8	2
7	2,000	8	2
8	5,000	8	2
9	10,000	8	2
10	200,000	5	2
11	1,000,000	5	2
12	1,000,000	5	2
13	1,000,000	5	2
14	10,000,000	5	2
15	10,000,000	5	2
10	40,000,000	5	2

TABLE II: Exhaustive cycle search (cards including 2 jokers)

Nb. Cards n	Nb. Config: $n!$	Min. Cycle	Max. Cycle	% Min. Cycle	% Max Cycle
4	24	5	5	100%	100%
5	120	7	14	13.3%	45.8%
6	720	8	54	4.3%	36.3%
7	5,040	4	74	$< 0.1\%$	86.8%
8	40,320	5	936	$< 0.1\%$	40.1%
9	362,880	30	2808	$< 0.1\%$	97.9%

Algorithm 1 Cycle Detection Algorithm

Input: A keystream file of which each line is composed of a state $S(t)$ and a Solitaire output $o(t)$, and the algorithm parameters α, β, γ

Output: A possible cycle Length if the algorithm succeeds

1: Randomly select a line L_1 at the position i_1 in the keystream file, where $1 \leq i_1 \leq \alpha$
2: Sequentially search for repetitions of L_1 in the keystream file
3: **if** No Matches are Found **then**
4: Goto Step 1
5: **else**
6: Record positions $i_2, i_3, ..., i_\beta$ of those lines that are identical to L_1, or until the end of the keystream is reached
7: **end if**
8: **if** $i_\gamma - i_{\gamma-1} = i_{\gamma+1} - i_\gamma = \cdots = i_\beta - i_{\beta-1}$ **then**
9: **return** Cycle Length := $i_\beta - i_{\beta-1}$
10: **else**
11: Test fails
12: **end if**

have used in this paper for the purpose of cycle computations.

Observe that though Algorithm 1 is quite successful in detecting a cycle length during a single run, it is not able to produce the following information: (1) the number of different cycles a keystream contains; (2) whether a detected cycle is a maximum length cycle, or a minimum length cycle.

B. Cycle Detection Results

In this section, we present some practical detection results based on Algorithm 1. We implemented the algorithm in the Perl language, an extremely efficient programming language in tasks such as line pattern matching, and pattern mining within a keystream file. Table II and Table III below summarize our findings. Table II contains cases where an exhaustive search is feasible. By an exhaustive search, we mean that the cycle detection in Algorithm 1 is applied to the keystream generated by every initial cards position on a deck of n cards. There are exactly factorial of n permutations, which correspond to these positions. At the level of coding, we integrated the GNU GSL library [1] with the ANSI C code of Solitaire available in [18]. The goal was to benefit from using the GSL permutation functions during the initial state/permutation rotation stages. The exhaustive search computations were performed on a PC with 32-bit Ubuntu OS running upon an Intel Core i7/2.50 GHz CPU. The exhaustive search of cycles was only feasible when the number of cards was lower than 10 ($n < 10$). The time-stamp shows that computations for the $n = 9$ case took 39 days for checking all the possible initial configurations (which turns to be factorial of 9, i.e., 362,880 possible configurations). Given such a speed, the estimated computation time for the $n = 10$ case would have been 195 days, whereas for 16 cards it would have taken more than three years.

Storage complexity is another issue. Recall from Algorithm 1, the execution of the cycle detection algorithm requires the storage of a keystream file. The length of the keystream should be at least equal to the cycle length multiplied by β (being β the number of matches one needs in order for the program to terminate). For example, in the case $n = 17$, the estimated max cycle length is 76,403,613 (cf. Equation (1)). If one chooses $\beta = 10$, the length of the keystream is: $10 \cdot 76,403,613$. This is equivalent to a 60 Gb textfile. Files of this size exceed the file size limit for a 32-bit machine. As a result, we were only able to run the cycle detection algorithm for the cases $4 \leq n \leq 16$.

As a first observation, the Solitaire keystream may output different cycle lengths depending on the initial state of the deck. We have included in Table II percentages of the minimum and maximum cycles for all exhaustive scenarios. The maximum cycle length grows as n grows, and the growth rate turns out to be exponential. The minimum cycle lengths however, do not seem to be correlated, or have any functional dependence on n. The minimum cycle length for each n occurs extremely rare as shown from Table II when $n \geq 6$. On the other hand, the percentage of the maximum cycle length occurrences for each n is much higher than any other cycle length occurrence percentages. This observation also leads us to the belief that if we perform cycle detections on a set of

TABLE III: Random cycle search (cards including 2 jokers)

Nb. Cards n	Nb. Config Used	Max. Cycle	% Max Cycle
10	1,054,596	10,221	75.9%
11	246,663	7,543	39.9%
12	195,943	33,058	87.2%
13	119,373	554,526	91.2%
14	1,841	1,013,519	-
15	1,343	6,702,967	-
16	17	21,936,204	-

randomly selected initial positions for the case $n \geq 10$, there is a high probability that the maximum cycle length of the set is also the global maximum cycle length.

Therefore, for the cases $n \geq 10$, the cycle detection algorithm is performed on a set of randomly picked states, i.e., a fraction of all cards positions in a same state length. Tables III contains the random configuration search results for the cases $n = 10, ..., 16$. We did not record the percentage of maximum cycle length occurrences for the cases $n = 14, 15, 16$ on the ground that the number of checked samples (i.e., initial positions) was too small to produce any significant results.

C. Extrapolation Results

Extrapolation results are based on the *Maximum length* cycles. Figure 3(a) and Figure 3(b) show the curve fitting for the maximum cycle length growth based on the exhaustive search results, and the random cycles search results, respectively. The resulting fitting curve for both cases is an exponential function in n. Based on Figure 3(a), the coefficients a and b are expressed with 95% confidence bounds. The resulting function is represented in Equation 1.

$$f(n) = a \cdot \exp(b \cdot n), \text{ where } a = 0.02925, b = 1.276 \quad (1)$$

V. Randomness Evaluation of the Solitaire PRNG

We evaluate the robustness of the Solitaire keystream algorithm as a PRNG using the NIST Statistical test suite for Random and Pseudo-random Number Generators [17].

A. Statistical Testing Tool

Our test results are obtained from the NIST test suite. The statistics inferences are based on the P-Value approach. The test consists of computing a test statistic for a sequence s and its corresponding P-value. The P-value is the probability of obtaining a test statistic as large as the one observed from a random sequence. The sequence s passes a statistical test if the P-value $\geq \alpha$, and fails otherwise. Here, α is the significance level.

1) NIST Test Results: The NIST test suite produces a table containing a summary report based on the Solitaire keystream output bits it has tested. The table consists of:

- Ten columns labelled C1 to C10: each column represents the number of tests that has a P-value in the corresponding range (i.e., the range from 0 to 1 is divided into ten equal-length segments called *bins*). A perfect RNG would have P-values uniformly spread over the range 0 to 1.
- P-VALUE column records P-values: In particular, if P-Value $\geq$ 0.001, then the sequences can be considered to be uniformly distributed.[1]
- PROPORTION column indicates the number of P-values that are above the 0.01 confidence interval. Thus, it is acceptable for a few individual tests to fail. The test suite will indicate a problem by flagging the PROPORTION number with an "*".

[1]See the NIST test suite documentation in [17] for further details.

TABLE IV: Proportion of passed tests on different cards number (with jokers)

Cards	Frequency	Block Frequency (16 bits)	Block Frequency (800 bits)	Cumulative sums
10	0.0	0.0	0.0	0.0
18	**0.98**	0.0	0.35	**0.98**
34	**0.99**	0.51	**0.98**	**0.99**
66	**1.0**	0.13	**0.88**	**1.0**

- Name of the test. For the Solitaire outputs, we used the following tests:
  - Frequency (monobit) is to determine whether the number of ones and zeros in a sequence are approximately the same.
  - Block Frequency is to determine whether the frequency of m-bit blocks in a sequence appears as often as would be expected for a truly random sequence.
  - Cumulative Sums is to determine whether the maximum of the cumulative sums in a sequence is too large or too small. In other words, it is used to detect if there are too many zeroes or ones at the beginning of the sequence.

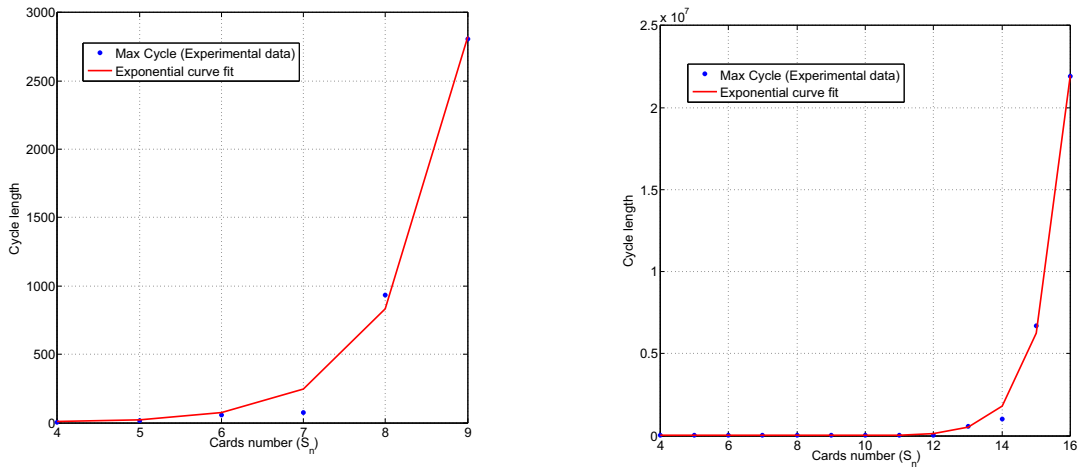
2) Data Management: A large amount of data is generated for the purpose of the NIST evaluation. We conducted various tests for the Solitaire keystream algorithm, varying the deck size n. The input files usually contain billions of bits. We have adhered to the following test procedures: (1) fix the deck size, so that the representation of the card could be completely rendered in bits (e.g., in the 16 cards version of Solitaire, each card is represented in 4 bits); (2) generate necessary amount of keystream output data in decimal; (3) transform the decimal data into bits based on the size of the deck, and align all the outputs into one line of bits; (4) input the bit file into the NIST testing suite (version *sts-2.1*).

B. Statistical Tests

1) A test on Solitaire 34 cards: Figures 4 and 5 show NIST test summaries over a set of data produced by the Solitaire with 34 cards (32 cards + 2 jokers). An input file of one billion (i.e., 10^{12}) bits is used. During the test, 100 individual sequences are constructed, each of which consisting of 10 million bits. For each sequence, three basic and essential tests described in the previous section are performed. The two block values considered in the block frequency tests are16-bit and 800-bit.

For this test, NIST indicates that the minimum pass rate for each statistical test is approximately equal to 0.96 for a sample size equal to 100 binary sequences. As shown in Figure 4, both the frequency test and the Cumulative Sums test have passed. However, the block frequency test fails when the block size is 16 bits. After re-sizing the block up to 800 bits (cf. Figure 5), the Block frequency test is successful.

2) A general test: We have also performed the NIST tests for Solitaire with 10 cards, 18 cards, 34 cards and 66 cards,



(a) Exhaustive search

(b) Probabilistic search

Fig. 3: Evolution of the maximum cycle length

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
15	9	11	8	6	10	14	12	5	10	0.419021	0.9900	Frequency
80	11	5	0	4	0	0	0	0	0	0.000000 *	0.5100 *	BlockFrequency
14	9	10	4	11	15	8	11	10	8	0.455937	0.9900	CumulativeSums
14	12	9	15	10	5	13	5	10	7	0.249284	0.9900	CumulativeSums

Fig. 4: NIST test outputs for 34 cards (16-bit block size)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
15	9	11	8	6	10	14	12	5	10	0.419021	0.9900	Frequency
24	12	10	15	8	8	6	6	8	3	0.000216	0.9800	BlockFrequency
14	9	10	4	11	15	8	11	10	8	0.455937	0.9900	CumulativeSums
14	12	9	15	10	5	13	5	10	7	0.249284	0.9900	CumulativeSums

Fig. 5: NIST test outputs for 34 cards (800-bit block size)

all considered with the two jokers. These values correspond respectively to the 3, 4, 5, and 6 bits output values. As before, an input file of 1 billion bits is used for each deck size. We have used the same test configuration as the previous test case of 34 cards.

Table IV contains the test results. We have highlighted successful test entries in bold. Recall that the minimum pass rate for each statistical test is approximately equal to the proportion of 0.96 for a run of 100 binary sequences.

Test Results for the case of $n = 10$ are not statistically good. Here is an example of Solitaire output when $n = 10$:

`1 3 5 5 5 8 3 3 5 5 5 3 1`

The frequent succession of the same pattern `3 5 5 5` indicates the reason of failure for the frequency test and block frequency test (16 bits block).

From a deck of size $n = 18$, the sequences pass the frequency test and the cumulative sum test. However, only the version of 32 cards passes the block frequency test when the block size is equal to 800 bits. This problem of block frequency can be explained in the following sample output ($n = 18$):

`6 16 16 14 14 ..... 12 15 1 15 13 9 7 7 3 5 10 6 16 16 16`

In the beginning and the end of this sequence, we see a succession of `6 16 16`. Converting this subsequence to a block of 16 bits, we have `0110111111111****`, this block

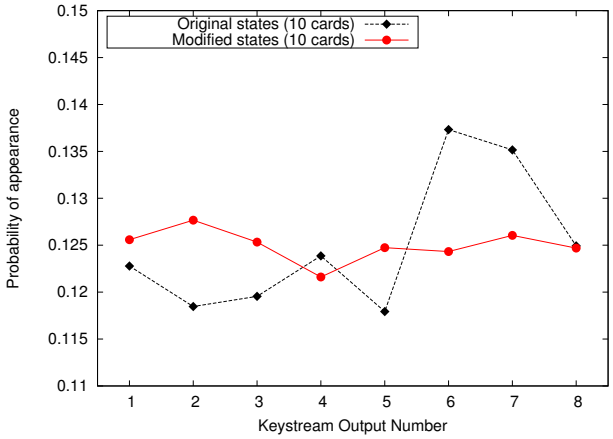


Fig. 6: Distribution of the Solitaire keystream numbers (original and modified states)

TABLE V: Proportions of passed tests in the original and modified Solitaire - 10 cards version

Solitaire (10 cards)	Frequency	Block Frequency	Cumulative sums
Original states	0.0	0.0	0.0
Updated states	0.44	**0.97**	0.44

pattern frequently occurs in the rest of the keystream file, which explains the failure of block frequency test, although the block repetition for $n = 18$ is much less frequent than the case $n = 10$.

We may conclude here that when the block size is larger, the probability of a block repetition is lower. Hence, this produces a higher chance of passing the block frequency test.

C. Modified Solitaire keystream Algorithm

The modified Solitaire algorithm is a slight variation of the original Solitaire algorithm described in Section III. The original keystream algorithm has a fixed initial state, whereas the modified Solitaire requires a random or a pseudo-random change of initial state once a maximum cycle length as predicted by formula (1) is reached (i.e. start the Solitaire with a random initial state after a max. cycle is reached).

1) Comparison when $n = 10$: Figure 6 is a plot comparing the original Solitaire against the modified version. To generate this plot, 1000 experiments are run over a population of 100,000 Solitaire outputs. The modified Solitaire algorithm produces a better uniform distribution result than the original version. The probability distribution range for the original Solitaire lies between 0.118 and 0.137, whereas the modified Solitaire gives a probability distribution range between 0.122 and 0.128.

The NIST statistics confirm these resuts. They also show an enhancement from the modified Solitaire keystream algorithm, see Table V. Again here, the numbers highlighted in bold indicate the success of a particular test, with a minimum pass rate approximately equal to the proportion of 0.96 for a run of 100 sequences.

TABLE VI: Proportions of passed tests in the original and modified Solitaire - 18 cards version

Solitaire (18 cards)	Frequency	Block Frequency	Cumulative sums
Original states	**0.98**	0.93	**0.98**
Updated states	**0.99**	0.93	**0.99**

2) Comparison when $n = 18$: Using formula (1), a max. cycle for $n = 18$ is estimated to occur after approximately 270 million keystream outputs. For the NIST test, an input file consisting of 800 million bits is used. 800 individual sequences are constructed from this file, each consisting of 1 million bits. For the block frequency test, we have set the block size equal to 1000 bits.

Table VI summarises the test results. In particular, we have that the P-values are more less the same for both original and modified Solitaire. Nevertheless, the Block frequency test remains unsuccessful for both versions.

From the experimental results obtained in this paper, we may conclude that the Solitaire keystream (original and modified) algorithm does not operate as a typical block cipher, e.g., Cipher-block chaining (CBC), which has a fixed keystream cycle $2^{n/2}$ (n is the length of the state). In this case, the adversary can easily calculate the cyclic repetition of states and predict the next outputs (i.e., Birthday attack [4]). The Solitaire keystream cycles vary in length, depending on the initial state of cards positions, making the cycle search more difficult for the adversary. We have also shown that updating the Solitaire initial state after a maximum cycle length enhances the uniformity of the Solitaire outputs (cf. Figure 6). However, there is not enough convincing evidences at present for us to conclude that the modified Solitaire with the maximum cycle length is a better candidate than the original Solitaire to be used as a PRNG in constrained devices. In fact, the optimum period for rekeying the Solitaire deck (i.e.,the initial state) remains to be found, in order to show better statistics.

VI. Related Work

Few research work exists in the area of pencil-and-paper ciphers. Similar approaches, such as the straddling checkerboard cipher and the trifid cipher [16] have been reported as vulnerable [7]. A bias in the Solitaire cipher was reported by Crowley in [10]. Assuming the original design of Solitaire, based on a deck of 54 cards, and arithmetic operations modulo 26, Crowley showed that Solitaire tends to have shorter periods than expected. Anwar et al. addressed this issue in [2], and proposed an alternative version that aims at introducing additional shuffling steps and secret key parameters. The new version is claimed to be cryptographically secure, with superior randomness properties, compared to the original Solitaire cipher. Some statistical tests were conducted to assess the validity and overhead of the modified version. However, no formal proofs have been yet provided about its security. No

other improvements to the original Solitaire cipher exist in the related literature.

With regard to the properties of the original Solitaire cipher, some authors have studied its cycle structure and probabilistic relations. In [15], Pudovkina reported some probabilistic relations for the Solitaire keystream generator. Some properties about the distance relationship between the two jokers is described. However, no attack exploiting such a relation is presented. Exhaustive brute-force search is reported as the only potential technique to realistically affect the cipher. In [14], Pogorelov and Pudovkina modeled and investigated the semi groups and group properties of the Solitaire cipher. The deck state of Solitaire is presented as a composition of transformations, and summarized as four basic transformations based on the movement of the cards. The approach is presented as a promising construction to cryptanalyze the cipher or regular modification of the cipher. Their results show that the cycle structure of Solitaire is very unpredictable. Solitaire is reported as non bijective, with non trivial estimation of its cycle lengths.

VII. Conclusion

We proposed the adoption of Solitaire's PRNG (pseudo-random number generator) as an alternative solution to protect communication protocols from constrained devices. To ensure long term secure communications, we proposed an algorithmic solution to detect the cycle length of Solitaire's sequences. The cycle length is supposed to refresh Solitaire's state. This is proposed as a countermeasure to attacks such as the birthday attack. We estimated the maximum cycle length of Solitaire keystream in order to determine an update threshold. Our results show that the maximum cycle length grows exponentially with respect to the state size. To evaluate the security of Solitaire's PRNG, we performed a preliminary cryptanalysis of the keystream algorithm. We also presented a modified version of Solitaire, with a random update of Solitaire's state once a maximum cycle length is reached. We observed that the modified version enhances the uniformity of the generated keystream numbers and slightly improves the output statistics. Further research is expected to improve the design of our modified version of Solitaire's PRNG, and to determine the optimal threshold at which Solitaire's state must be updated.

References

[1] GSL - GNU Scientific Library. http://www.gnu.org/software/gsl/.

[2] Hirra Anwar, Sarah Masood, and Zahid Anwar. PISA: Improving the pseudo-randomness of the pen-and-paper cipher based on the Solitaire algorithm. In *15th IEEE International Multitopic Conference (INMIC 2012)*, pages 239–244. IEEE, 2012.

[3] B. Barak and S. Halevi. A model and architecture for pseudo-random generation with applications to/dev/random. In *12th ACM conference on Computer and communications security*, (CCS'05), pages 203–212. ACM, 2005.

[4] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. In *38th Annual Symposium on Foundations of Computer Science*, (FOCS'97), pages 394–403, 1997.

[5] M. Bellare, T. Krovetz, and P. Rogaway. Luby-Rackoff backwards: Increasing security by making block ciphers non-invertible. *Advances in Cryptology (EUROCRYPT'98)*, 1403:266–280, 1998.

[6] M. Bellare and B. Yee. Forward-security in private-key cryptography. *Topics in Cryptology–(CT-RSA'03)*, 2612:1–18, 2003.

[7] Jonathan M. Blackledge. *Cryptography and Steganography: New Algorithms and Applications*. Center for Advanced Studies Warsaw University of Technology, 2011.

[8] Andrey Bogdanov, Gregor Leander, Christof Paar, Axel Poschmann, Matthew Robshaw, and Yannick Seurin. Hash Functions and RFID Tags : Mind The Gap. In *10th International Workshop Cryptographic Hardware and Embedded Systems*, (CHES'08), 2008.

[9] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. *Algebraic aspects of the advanced encryption standard*. Springer, 2006.

[10] Paul Crowley. Problems with Bruce Schneier's Solitaire, 2001.

[11] A. Juels and S. Weis. Authenticating pervasive devices with human protocols. In *25th Annual International Cryptology Conference*, CRYPTO'05, 2005.

[12] F. Martin and C. Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. In *On the Move to Meaningful Internet Systems*, (OTM'06), 2006.

[13] George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[14] Boris Pogorelov and Marina Pudovkina. Properties of the Transformation Semigroup of the Solitaire Stream Cipher. *IACR Cryptology ePrint Archive, http://eprint.iacr.org/2003/169*, 2003.

[15] Marina Pudovkina. Probabilistic Relations for the Solitaire Keystream Generator. In *IFIP TC-11 WG11.4 Annual Working Conference on Network Security*, pages 61–74. Springer, November 2001.

[16] Dirk Rijmenants. Hand Ciphers, http://users.telenet.be/d.rijmenants/en/handciphers.htm, 2014.

[17] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.

[18] Bruce Schneier. The Solitaire Encryption Algorithm, version 1.2. https://www.schneier.com/solitaire.html, 1999.

[19] Wiem Tounsi, Nora Cuppens-Boulahia, Joaquin Garcia-Alfaro, Yannick Chevalier, and Frédéric Cuppens. KEDGEN2: A key establishment and derivation protocol for EPC Gen2 RFID systems. *Journal of Network and Computer Applications*, 39(1):152–166, 2014.