Creating Randomness with Games

Jaak Henno*, Hannu Jaakkola, Jukka Mäkelä*****

*Tallinn Technical University, School of Information Technologies: Department of Software Sciences, Ehitajate tee 5, 19086 Tallinn, Estonia, jaak.henno@ttu.ee

**Tampere University, Pori Campus, P.O. Box 300, FI-28101 Pori, Finland, hannu.jaakkola@tuni.fi

*** University of Lapland, Rovaniemi Finland, jukka.makela@ulapland.fi

Abstract: In our increasingly connected and open World, randomness has become an endangered species. We may soon not have anything private, all out communication, interaction with others becomes publicly available. The only method to secure (temporarily) communication is mixing it with randomness – encoding it with random keys. But massive reuse of the same sources of randomness and rapid development of technology often reveals that used sources were not perfectly random. The Internet security is top-down, based on higher-level certificates, but we can never be quite certain with 'given from above' products in their quality – in order to beat each other producers are 'cutting corners' and even the high-level security certificates are available on Internet dark markets. This clearly shows in tremendous increase of all kind of security accidents, so there is an urgent need for new, independent sources of randomness. Mathematical treatment of randomness is based on infinite concepts, thus useless in practice with devices with finite memory (humans, computers, Internet Of Things). Here is introduced a definition for randomness based on devices with finite memory – k-randomness; it is shown, how this allows to create new randomness in computer games; numerous tests show, that this source is quite on par with established sources of randomness. Besides algorithmically-generated randomness is in computer games present also human-generated randomness - when competing players try to beat each other they invent new moves and tactics, i.e. introduce new randomness. This randomness appears in the sequence of players moves and when combined with the sequences of moves of other players can be used for generating secret keys for symmetric encryption in multi-player game communication system. The method does not use public-key step for creation of shared secret (the key), thus the encryption system does not need any upper-level security authorities.

Keywords: entropy; randomness; encryption; digital games; finite-state machines; human behavior; cyclic order; k-random sequences; player’s actions combination

1 Introduction

Traditional fields of human activity – agriculture, manufacturing and construction are currently producing only 35% of all values consumed by humanity [1], the rest is produced in mental/information sphere, where the input for production of new values is data. Most important source of new data are we self and all producers are trying to capture as much as possible data concerning us.

Thus, it is becoming increasingly important to safeguard our privacy, our 'self', our data and our communication and for this we need randomness. Randomness has become a commercial product, several countries are introducing new random number generators [2] and with rapid increase of communication and data we have growing need for new randomness for encryption. Encryption ciphers are based on modifying messages using random data. But encryption is only temporary measure - when some encryption method/cypher is broken it becomes worthless and the randomness used in it also becomes worthless. Data breaches are increasing by more than 20 percent in a year [3], they have become the most worrying feature of Internet [4] and every breach is decreasing the value of randomness used. Growth of ’big data’ inevitably increases amount of randomness needed to establish ownership for these ‘big data’ items. Thus, we constantly need new sources of randomness. New computing environments - Internet of Things (IoT), virtual/cloud servers etc. all increase the need for randomness.

To satisfy this continually growing need for randomness there are emerging dedicated services to serve random data [5]. For delivering this data was proposed a special new protocol 'Entropy as a Service' [6]. But for delivery this data also should be encrypted, thus it is a new source needing 'fresh' entropy. So it is not clear, whether this kind of ‘top-down’ service will reduce the need for entropy or contrary, increase it.

Trust in the current top-down security practices, based on higher-level security authorities issuing and controlling security certificates is decreasing – the high-level certificates are on sale on ‘Dark Web’ for $260 … $1,600 [7]. Security should be ‘bottom-up’ (Neighborhood Security) and entropy/randomness created just where it is needed (like in blockchain). Everything is simpler and safer if the entropy/randomness is generated where it is used.

Randomness is quite an infeasible concept. Mathematical treatment of randomness is based on infinite concepts, thus not applicable in real-world practice, where all information is handled by devices with limited memory - humans, computers, devices in IoT. Here is introduced a definition of *k*-randomness applicable to devices with finite memory and shown, how this can be used to produce with games of chance random integer sequences; tests show, that the created randomness is quite on level with established sources of randomness. As a practical use of generated with game randomness is introduced herein a method to create secure encryption keys for symmetric encryption.

2 Types of Randomness

It is impossible to generate random values using a computer's basic operations – binary operations conjunction $\&$ (AND), disjunction $\vee$ (OR) and the unary negation $\neg$ (NOT) – all combinations of these connectives return single determined value (if not, then the computer is broken). John von Neumann commented on this: "Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin". But he did not elaborate: “why?” “what are the ‘non-sinful means’ of creating randomness?” and what actually is randomness

Computers are deterministic, orderly; randomness is the opposite of order, the absence of any pattern. The current understanding is that 'true' randomness can be extracted only from physical processes, which have rich deep inner structure – entropy, e.g. thermal fluctuations in processor, pixels found by mouse sensor when user makes some rapid random strokes, atmospheric disturbances [8] etc. These sources are 'Pure Randomness Generators' (PRG), but they are often not rich enough e.g. for network servers which do not have external devices.

Operating systems send ‘new-born’ messages, thus should have means for securing them with ‘new-born’ randomness/entropy and for this all operating systems maintain an entropy pool. The first versions of Linux kernel created entropy from the third derivative of differences in timings of user actions; this information is stored in two files */dev/random* and */dev/urandom*. This method turned out to be too slow and currently uselow-order bits (lest significant, i.e. changing most rapidly) from timing of user actions on keyboard, mouse movements, IDE requests; extracting entropy from audio [9] and video [10] data is also studied.

Programming language's compilers need methods to create random values [11], [12]. All compilers work under an Operation System (OS) and get their randomness from OS, e.g. in Windows environment randomness to all programming languages comes from the same source as to the Microsoft C/C++ compiler (and the Intel compiler [13] or newer [14]) - they use the random values generated in Common Language Runtime [15], using the entropy produced by processor. But there have been found several problems for Intel processors [16], [17] thus specialists distrust randomness produced by Intel processors [18], e.g. in Linux kernel it is only one of many inputs into the random pool. Research has shown that even processors built-in functions (PRG-s) for generating random values can be compromised [19] and processors and microchips may have built-in hardware rojans [20] which can leak information leading to successful key recovery attacks. After the NSA (U.S. National Security Agency) leaks by Edward Snowden, many engineers have lost faith in hardware randomness [21].

The hardware entropy pool decrease every time random numbers are generated from it Requesting many random numbers may starve them; this is a practical issue on servers without input devices. Other PRG sources also decrease, e.g. the online source of randomness *Random.org* [22] limits its daily available amount of free random bytes (currently 10^6 bits). Thus PRG sources do not suffice, for random number generation are needed also computer algorithms.

Computers are finite devices and after a while 'fall into loop', start to repeat computed values. Thus 'calculated randomness' is pseudo-randomness produced by pseudo-random number generators (PRNG). All PRNG-s are loops, which after their period repeat produced values.

The first value in the loop is created using a random seed, i.e. comes from other, usually PRG source. The next value is calculated from the previous one by some recurrent function; common method is to use linear (for speed) recurrent functions with reduction by modulus. For these Congruential Generators (CG) is the period (length of the loop) the most important measure of security of such a generator. For the C language it should be at least $2^{32} = 32767$ [23] - a rather small number for current CPU-s and its use (installing the Microsoft or GNU suite of compilers) requires decent computer skills. A 'high-end' PRNG–s have much bigger period, e.g. period of the 'mersenne twister' is the Mersenne prime $2^{19937} - 1$, but use of these requires good computer skills and good hardware.

Many PRNG-s which at their introduction were considered 'good enough' have later become obsolete. For example, John von Neumann used for generation of random numbers the 'middle-square' method [24] – for the recurrence step earlier produced number was squared and then the middle digits were sliced out. This mix of number's semantics (squaring) and syntax (use only middle digits) was used already in 13th century [25] and seems good, since un-computability results (e.g. the Rice theorem [26]) indicate, that most semantic properties are undecidable from syntax. However, research revealed that with *n*-bit seed the length of generated cycle is $\leq 8^n$ and with many seeds even much shorter, e.g. $3792 \to 79^2 = 6241 \to 24^2 = 0576 \to 57^2 = 3249 \to 24^2$ - a cycle.

Many PRNG-s have similar fate. The RC4 (Rivest Cipher 4) was used in several commercial encryption protocols and standards (e.g. in the TLS - Transport Layer Security – the base of all traffic in WWW), but is currently prohibited; widely known was periodicity in the random function of Microsoft PHP translator. Already in 1999 were presented general methods for prediction of CG-s [27], [28].

For assessment of quality of new PRNG-s have been constructed several suites of statistical tests – the NIST (the U.S. National Institute of Standards and Technology) suite [29], the Dieharder (Marsaglia) suite [30], ENT [31] etc. These tests check presented samples for some common regularities in everyday data, e.g. the Dieharder 3.20 implements 26 tests.

We tested with the ENT suite several established sources:

1. The first 7 KB part of the 2.1 GB file */dev/urandom* from Ubuntu 16.04.3 (a three months old installation, used mainly for making music)

2. 10000 decimal digits downloaded from the *Random.org* (randomness from atmosphere);
3. 10000 decimal digits created using the function *window.crypto.getRandomValues();*
4. 10000 decimal digits created by Wolfram Mathworld with function *RandomInteger[]* using the default method *Rule30CA*

In the following table are shown three characteristics from the test with the ENT suite of statistical tests: entropy (bits per bit), possible compression (randomness can't be compressed) and serial correlation coefficient.

Table 1.
Some characteristics of established sources of randomness

	Entropy	Compression	Correlation
/dev/urandom	0.988577	1%	0.035161
Random.org	0.919040	8%	0.060193
Windows	0.974450	2%	-0.010378
Wolfram	0.974448	2%	-0.010948

The results are rather similar except a bit weaker performance of atmosphere processes – the random sequence downloaded from the site *Random.org*.

However, statistical tests cannot guarantee randomness and the results of these tests do not tell the whole truth. Although the randomness from Linux performed best, visual inspection (the 'Statistics' tool from the free hex editor HxD) reveals, that distribution of frequencies in */dev/urandom* contains a surprising peak.

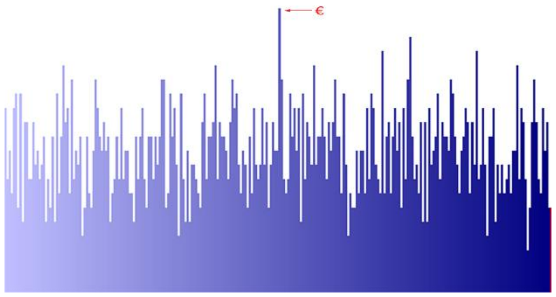

Figure 1.
Distribution of frequencies of bytes in the first 7 KB from the */dev/urandom* file from Ubuntu 16.04.3; the sharp peak in the middle is the code for the € (Euro) symbol

Thus, statistical tests are also uncertain method for evaluation of randomness sources. There could be surprising dependencies in data - the above peak in € symbol code come from computer, which is rather new (in use only for several months) and was never used for any kind of financial data handling.

3 Randomness from Games

Randomness is an essential ingredient in most games and for utilizing this source have been proposed different methods [32] [33]. In [34] authors presented a method for producing on-line in real playtime binary random strings from simple repeated games; here the principles of the proposed method are applied to produce from gameplay *m*-ary ($m > 2$) random sequences.

In the (economics - based) texts on games the game decision mechanism is usually not detailed – it is determined by unpredictable markets. In video games decisions are deterministic, thus we follow computer science tradition (see e.g. [35]) and use for the decision mechanism finite automaton. Games considered here are 'games of luck', where both players have equal chance to win and the best strategy (the Nash equilibrium) for both players is total randomness, i.e. in the game payoff matrix (economics-based format) the sums of all rows and columns are equal, such games are e.g. the rock-paper-scissors and odd-even.

If one player is human or some established source of randomness and the other – the computer algorithm, then the (statistical) result of numerous repeated plays is also an assessment for the quality of computer-created randomness. The length of all considered here random sequences/plays is 10000, following the suggestion: "A reasonable estimate (for humanly interesting cases) reckons that some 10,000 digits would suffice" [36]

Thus in the following game is a structure $\mathrm{G} = < \mathrm{P}_1, \mathrm{P}_2, M, R, \mathrm{A} >$, where

$\mathrm{P}_1,\ \mathrm{P}_2$ are (two) players;

$M = \{0,1,...,m-1\}$, $m > 2$ - the set of legal moves (actions) of players (the same set for both players); in every round both players apply simultaneously one action which initiates some change in automaton A

$R = [r_1, r_2]$ - player's utilities (points); at the start $r_1 = r_2 = 0$

A - a finite automaton, deciding the output (move) and payoffs to players. Here are considered simultaneous (synchronous) games, where players produce their actions (moves) simultaneously at the same time, thus the input for the automaton A are pairs (m_{1i}, m_{2i}), where m_{1i} is the *i* move the first player, m_{2i} - *i* move second player; denote $(m_{1i}, m_{2i})^{-1} = (m_{2i}, m_{1i})$ - actions of players switched.

Automaton's (possible) outputs are "1" (player P_1 won, $r_1 += 1$), "-1" (player P_2 won, $r_2 += 1$), "0" – draw. Thus, the automaton has four distinguished states:

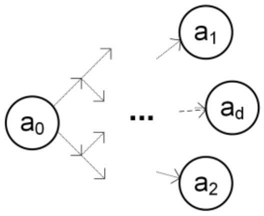

Figure 2.
Game automaton with distinguished states: a_0 – the start state, a_1 – first player won, a_2 – second player won, a_d – draw

Here a_0 is the game start state, a_1 - here automaton outputs, that the first player won, a_2 - the second player won, a_d - draw; in any other state (not shown) automaton does not produce output; $F = \{a_1, a_2, a_d\}$ is the set of final states

Automaton does not have cycles, thus the graph of the automaton is a tree (with possible loops with limited length at some nodes) and all rounds are finite. The length $D(\mathrm{A})$ of the longest round (the depth of the tree) is the depth of the game. Game is repeated and after some fixed number (here 10000) of moves automaton announces if the result of the game is draw or who won.

Automaton is deterministic, i.e. in any state $a \in \mathrm{A}$, $a \notin F$ and for any move (m_i, m_j) there is a single transition $a(m_i, m_j) \to a' \in \mathrm{A} \setminus \{a_0\}$

Transitions are in natural way extended to words from $M^{2D(\mathrm{A})} = \{(m_i, m_j)\}^+ = \{(m_{11}m_{21})...(m_{1t}m_{2t}),\ t < D(\mathrm{A})\}$

$$a((m_{11}, m_{21})(m_{12}, m_{22})...(m_{1t}, m_{2t})) =$$
$$(a(m_{11}, m_{21}))((m_{12}, m_{22})...(m_{1t}, m_{2t}))$$

Action of words on states of automaton A creates partition of the set $M^{2D(\mathrm{A})}$ of words into three sub-languages:

$$\mathrm{L}_1 \equiv \{w \mid a_0 w = a_1\}$$

$$\mathrm{L}_{2=} = \{w \mid a_0 w = a_2\}$$

$$\mathrm{L}_d \equiv \{w \mid a_0 w = a_d\}$$

Call all words $w \in M^{2D(\mathrm{A})}$ plays.

4 Symmetry-based Non-Learnable Games

In games of chance, nobody wants to have worst chances by design of the game and all actions of players should be significant, i.e. could change the result, thus these games obey the following symmetry principle:

in any move (m_i, m_j) from any play both players have equal chances, i.e. if all other moves in the play remain the same they can change their action so that expectation of outcomes a_1, a_2 is the same, i.e. 0.5.

Since there are m^2 possibly moves it follows that if m is odd, the set L_0 can't be empty – otherwise $|\mathrm{L}_1| = |\mathrm{L}_2|$ is impossible.

Therefore the games with $D(\mathrm{A}) = 1$ (one round, i.e. every single state of the automaton) should satisfy the following conditions.

1. All games are zero-sum, i.e. the involution $\alpha : (m_i, m_j) \to (m_i, m_j)^{-1} = (m_j, m_i)$ produces an automorphism of the automaton A, i.e. $\alpha(\mathrm{L}_1) \subseteq \mathrm{L}_2,\ \alpha(\mathrm{L}_2) \subseteq \mathrm{L}_1,\ \alpha(\mathrm{L}_d) \subseteq \mathrm{L}_d$.

2. Any substitution

$$\beta : \{m_0, ..., m_{k-1}\} \xrightarrow{1-1} \{m_0, ..., m_{k-1}\}$$

of actions produces automorphism of automaton A, which does not break the partition $\{\mathrm{L}_1,\ \mathrm{L}_2,\ \mathrm{L}_d\}$.

3. The sublanguage L_d contains all words $(m_i, m_i),\ m_i \in M$ and is minimal – it should not contain words which could be moved into L_1 or L_2 without breaking conditions 1,2.

Proposition. Conditions 1-3 define for given m unique (up to isomorphism) game payoff function.

Proof. Consider the set $\mathrm{M}_1 = \{(m_1, m_i), m_i \in M \setminus \{m_1\}\}$, i.e. moves, where the first player selects action m_1. From the condition 3. it follows, that the set L_d can contain at most one of them, otherwise we could pairwise move them one to L_1, another to L_2 without breaking conditions 1.-2.

From the condition 2. it follows, that there should be equal number of elements from the set M_1 inside sets $\mathrm{L}_1,\ \mathrm{L}_2$. If these sets were of different size then substitutions which keep m_1 fixed, but move other actions will break the condition 2.

According to condition 2. actions inside L_1 could be re-arranged so that $(m_1,m_2),(m_1,m_3),...,(m_1,m_{k1})\in \mathrm{L}_1$, $k1=\lfloor m/2 \rfloor$. Using the substitution $\chi : m_i \to m_{i+k1}$ and the property 2. we get that all sets $\{(m_i,m_{i+1}),(m_i,m_{i+2}),...,(m_i,m_{i+k1})\}$ should belong to L_1. If m is odd, then all moves are now evenly divided between sets L_1 and L_2. If m is even, then from the above discussion it follows that the moves $(m_i,m_{i+k/2})$ should belong both to L_2 and L_1, i.e. they should be moved to set L_d.

Thus, a game with properties 1.-3.- has an unique (up to involution α) payoff function, based on cyclic order [37] on moves: if moves of players are $\mathrm{P}_1(m)=m_i$, $\mathrm{P}_2(m)=m_j$, then output from the automaton A is:

$$\operatorname{sgn}((m_i-m_j) \bmod m - m/2)$$

Table 2.
Decision table of cyclic 5-ary order; e.g. $(0,2),(1,2)\in \mathrm{L}_1$, but $(0,3),(2,1)\in \mathrm{L}_2$

	0	1	2	3	4
0	0	1	1	-1	-1
1	-1	0	1	1	-1
2	-1	-1	0	1	1
3	1	-1	-1	0	1
4	1	1	-1	-1	0

In case $m=3$ this is isomorphic to the well-known game rock-paper-scissors, which appeared in China at the beginning of Current Era. Apparently, Chinese know how to use symmetry groups for inventing amusing games.

There are variants of this game with greater cycle length e.g. movements of fighters can be *punch, kick, grab, push* (the next one stronger than the previous), in some games even more than ten with non-linear order [38]. The pay-offs could be any (increasing) sequence of numbers, e.g. in the above 5-ary game the payoffs (ordinals) could be integers [-2,-1,0,1,2] calculated by $(m_1-m_2) \bmod m - m$.

Table 3.
Payoff table for the first player in a cyclic 5-ary game (payoffs for the second player – multiply by -1 – game is zero-sum)

	0	1	2	3	4
0	0	-1	-2	2	1
1	1	0	-1	-2	2
2	2	1	0	-1	-2
3	-2	2	1	0	-1
4	-1	-2	2	1	0

5 Randomness for Finite Devices – k-randomness

Games where all actions produce similar reward (payoff) are non-learnable even for Tensorflow [39]. Nevertheless there are professional players of rock-paper-scissors [40], in USA is a league of professional players of the "Rock, Paper, Scissors" game [41], regular tournaments [42] and programming competition [43]. These professionals win their opponents not since they have learned the game (impossible!), but they have learned to learn their human opponents, i.e. create better randomness than their opponents.

Randomness is an evasive concept to define. The widely accepted definition is the Kolmogorov- Martin-Löf definitions: [44] [45]

sequence is random if it can't be compressed - expressed by any algorithm or device which can be described using less symbols than what are in the sequence.

This definition and other consequent definitions, e.g. [46] are using infinite concepts ('any algorithm') and apply to infinite sequences, thus useless in practice for evaluating quality of a source of randomness, where all actors/devices are finite (have finite memory) and produce finite sequences. All deterministic devices inevitably go to cycle after enough time (they do not have any new states and have to repeat already used states). Thus if deterministic devices A_1, A_2 with finite memory (humans or computers) interact, the deciding factor (who can predict/learn whom or contrary, cannot learn/predict and concludes, that the other produces random output) depends on available memory of these interacting devices [47], [48]. If A_1 has (sufficiently) more memory than A_2 so that it can remember the whole cycle produced by A_2, then when A_2 goes into cycle A_1 can always predict the next response of A_2 (A_1 has learned, 'pwnd' A_2), but since A_2 cannot store in its smaller memory the whole cycle produced by A_1, it has to conclude, that A_1 is creating random output. This insight is the base for the following definition:

a finite sequence of integers is k-random if its length > k and it can't be created as the sequence of outputs by any deterministic finite automaton with less than k states.

This definition can be expressed also in terms of the Zif-Lempel compression [49] thus this is a (particular case) of the Kolmogorov's definition:

a finite sequence of length > k is k-random if it can't be compressed using dictionary with item length < k.

When $k \to \infty$ this definition yields the presented above definition. All PRNG-s are interactive (input is the seed) deterministic finite automata – with the same seed they produce the same output and a PRNG with cycle length *k* produces (maximally) a *k*-random sequence.

6 The Cycle Disruption Algorithm

The *k*-randomness of a sequence (with length > *k*) actually means, that the sequence does not have a loop at its end. Any finite deterministic automaton with *k* states and *m* input symbols produces a periodic sequence [50], i.e. 'goes into loop', if the length of its input is longer than $k \times m$ - there are no new possibilities for the pair (*state*, *input*), thus a deterministic automaton produces the same subsequence which already occurred when it first arrived at (*state*, *input*). The evolutionary game theory of bounded rationality [51] of human players also predicts cyclic patterns in playing behavior [52]. Thus for successful play one has to find when the loop begins, i.e. automaton repeats its moves and then disrupt the loop. This is the idea of the loop disruption algorithm for creating random sequences:

scan the sequence of stored moves (input-output pairs) and when you find a situation similar to the current one (see that the sequence of last moves already appeared earlier) make the move that in the previous situation would be winning.

Suppose the sequence of moves in a game up to now is

$$a_0(m_{11}, m_{21})(m_{12}, m_{22}), ..., (m_{1n}, m_{2n}), ...(m_{1k}, m_{2k})(m_{1k+1}, m_{2k+1})...,$$
$$(m_{1n}, m_{2n}), ...(m_{1k}, m_{2k})$$

and $(m_{1n}, m_{2n}), ..., (m_{1k}, m_{2k})$ is the longest repeated subsequence of moves (looking from the current state backwards). Then algorithm should select the move which wins in the state (m_{1k+1}, m_{2k+1}) .

For instance, in the following situation from a real play of 3-ary game (moves follow in pairs, first human then computer, e.g. on the second move human played '1', computer – '2') computer discovered a repeated sequence (underlined), thus its next move will be '2':

1, 1, 1, 2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 2, 2, 2, 2, 1, 0, 0, 1, 1, 0, 2, 0, 0, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 0, 2, 2, 0, 1, 1, 1, 1, 2, 0, 0, 2, 2, 2, 2, 1, 1, 0, 0, 1, 2, 1, 0, 0, 2, 2, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 1, 0, 0, 2, 2, 0, 2, 0

The above sequence shows 45 moves (there are 90 symbols); length of the repeated subsequence is 4 moves; thus, the all sequence is 41-random.

The algorithm has been implemented in several browser games [53].

7 Tests

We tested the algorithm using it as the computer opponent in several games of luck (odd-even, rock-paper-scissors etc.) in many plays. Against human players (students from the Tallinn University of Technology and others) computer was in most cases already winning if the length of the game was >30. Humans are not sufficiently random to beat computer, especially if the memory requirements (length of the game) grows; it seems that here works the famous human short-term memory size principle – human memory is also finite [54].

As opponent players for testing were used several well-established sources of randomness: JavaScript's functions *Math.random()* and *window.crypto.getRandomValues()*, random numbers produced by Wolfram's *Mathematica* and a table of 10000 random integers downloaded from *https://www.random.org/*.

Tests indicated that the algorithm plays quite well against all these common sources of 'computed' randomness, i.e. its own randomness is on the same level. Below is a table of results from three tests, each a 10 series of plays, each play 10000 rounds with $m = 3$. Player P_1 is in the first test random numbers produced by the JavaScript function *Math.random()*, in the second – random numbers produced by the function *RandomInteger*[] of Wolfram's *Mathematica* (using the default rule *Rule30CA* in Mathematica for creating pseudorandom sequences) and in the third – random numbers produced by function *window.crypto.getRandomValues()*; player P_2 is our algorithm; L was the length of the longest cycle in the sequence of player's moves (i.e. the repeated sequence in above example). The last row indicates how many times each player won and length of the longest repeated sequence.

Table 4.
Results of tests

P_1	P_2	L	P_1	P_2	L	P_1	P_2	L
3350	3365	16	3403	3289	16	3356	3287	18
3396	3237	16	3369	3242	20	3277	3285	16
3328	3332	16	3392	3286	16	3281	3351	18
3428	3209	18	3392	3317	18	3342	3305	18
3310	3377	16	3512	3163	16	3299	3405	16
3369	3365	16	3424	3278	18	3366	3259	16
3360	3345	16	3440	3316	18	3367	3263	16
3315	3402	16	3355	3265	18	3283	3446	20
3322	3412	18	3409	3301	19	3383	3354	16
3294	3364	16	3330	3453	16	3324	3314	16
4	6	18	9	1	20	6	4	20

These results show, that used in tests sequences were (at least) 9980-random according to the above definition – they did not contain repeated sequences longer than 20 moves.

In the following table are discretized results (showing not actual results, but showing how many times player was better than the opponent) from 10×10000 series of tests against random numbers table from *Random.org* (the first column in all three sub-partitions), JavaScript function *Math.Random()* (the second column in all three sub-partitions) and the function *window.crypto.getRandomValues()* (the third column); the last row is the summary of results.

Table 5.
Discretized results of tests

Better P_1			Better P_2			Draw		
4	7	2	4	3	8	2	0	0
6	4	7	4	3	3	0	3	0
5	7	1	3	2	9	2	1	0
6	3	1	3	7	9	1	0	0
2	5	2	7	4	8	1	1	0
3	4	4	5	5	5	2	1	1
4	6	4	4	4	6	2	0	0
4	4	3	5	5	7	1	1	0
7	4	3	3	2	7	0	4	0
4	4	4	4	4	3	2	2	3
85	100	75	88	74	115	27	26	10

As seen from this table, our algorithm was nearly on the same level against *Math.Random()*, slightly outperformed the randomness from *Random.org* and slightly lost to *window.crypto.getRandomValues()*.

As output (new randomness) could be used two sequences – the sequence of 'full' moves (pairs of moves from player and computer) or the sequence of only computer-generated moves (twice shorter). We tested both as the source of random sequence against our computer's algorithm. In the following table are results from 10 series of plays, each 10000 rounds with $m = 3$; player P_1 is in the first series (the first three columns of the table) generated in a previous game (10000 moves against JavaScript *Random()*) sequence of full moves (pairs), in the second (the last three columns) – sequence of computer moves; player P_2 is our algorithm.

According to Table 6 the created in the game randomness already mostly outperformed our algorithm, its results are better than that of commonly established sources. When the generation process was iterated, i.e. generated randomness was used as input for the next play, it become more difficult to predict and our algorithm started to loose.

Table 6.
Tests against randomness, created in game

P_1	P_2	L	P_1	P_2	L
3298	3344	16	3403	3275	16
3377	3351	16	3328	3337	16
3439	3303	16	3391	3342	16
3419	3284	16	3375	3297	18
3328	3376	16	3490	3272	18
3471	3212	16	3408	3273	18
3360	3294	20	3379	3343	20
3367	3314	16	3342	3370	16
3513	3250	16	3376	3288	16
3416	3362	16	3362	3316	18
7	3	20	8	2	20

In the following table are results of play against randomness, created on third iteration, i.e. after three rounds of 10x50000 moves; player P_1 is in the first column the table of full moves (pairs), in the second – sequence of computer-generated moves.

Table 7.
Tests with iterated randomness

P_1	P_2	L	P_1	P_2	L
16811	16740	22	16617	16834	24
16840	16550	18	16636	16759	18
16785	16599	20	16729	16592	20
16779	16701	18	16703	16641	20
16777	16412	18	16601	16720	18
16928	16672	18	16801	16682	20
16904	16610	20	16757	16589	20
16902	16599	18	16787	16547	22
17017	16445	22	16581	16702	20
16680	16655	20	16458	16854	18
10	0	22	5	5	20

8 Use in Practice - Creating Encryption Keys with the Move Sequences Combination

Participants of online multiuser communities (multiplayer games, social networks) often want to establish also a direct communication with fellow players (chat).

This communication/chat system should not burden the game server, thus has to be implemented as a separate sub-process.

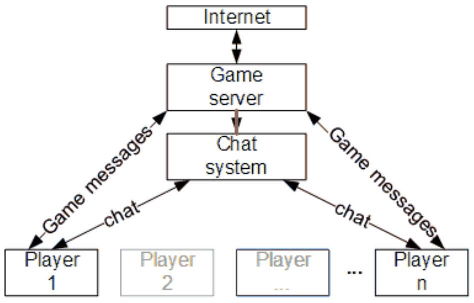

Figure 3.
Information flows in a multiplayer game with communication (chat system) for players

To ensure security of game and players communication (this may involve exchange of substantial game values) the communication system should be 'sand-boxed', should be encrypted and should not reveal any information to outside/Internet. This makes undesirable the commonly used first phase of encryption key creation – use of public-key encrypted communication, which requires security certificates from outside.

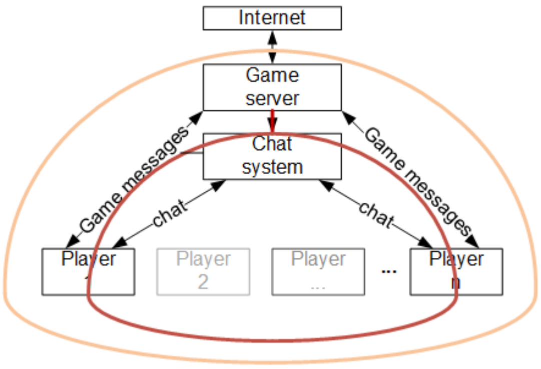

Figure 4.
Security surfaces of information flows in a multiplayer game with communication (chat system)

Statement "Players of games create randomness" is similar to many other non-provable statements. Faith in its correctness comes from long history of game-playing – nobody would play games where everything is pre-determined, just randomness makes games enjoyable. We play them more and more, thus in gameplay is created randomness and this randomness could be utilized.

A multiplayer game is a communication system where players constantly generate new randomness with their moves, thus for key generation could be used randomness from player's moves; for greater security could be added also a computer-directed player, who for its play uses the algorithm presented above.

The server records sequence of player's moves, e.g. for a game with two players *Alice* and *Bob* this sequence of their moves could be $m_{11}m_{21}m_{12}m_{22},...,m_{1t}m_{2t},...,m_{1l}m_{2l}$; here m_{1t}, m_{2t} are respectively moves of *Alice* and *Bob* in gameplay move/moment *t*.

To generate a key server sends to players the sequence of all moves from which the player's own moves are removed, e.g. server sends to Alice the sequence $*m_{21}*m_{22},...,*m_{2t},...,*m_{2l}$ - this information with holes does not give to an eavesdropper any information (it is assumed, that the game server communication with players is secure;, here is the only time when the game communication is used for the chat system). When players replace holes in the received sequences with their own moves they all get the same random sequence which could be used as the secure random key for symmetric encryption.

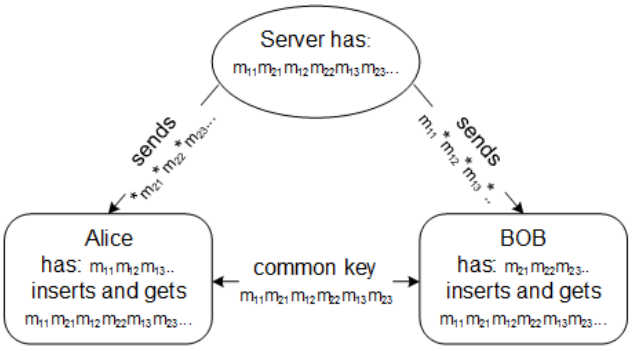

Figure 5.
Key generation combining a sequence with holes from server with sequence of player moves

This 'move sequences combination method' for symmetric key generation could be applied in any multiplayer game where players send their moves to game server (it is not essential, that moves alternate as in the above example). It has many desirable properties: key may be created for any subset of players (for any pair of them or for the whole player's community), after the first (secure) communication players could easily switch to a new key (without announcing the server) just with message „From now on use moves from time moment t_0 to t_1 " etc. To increase security of the key server could use some filters, e.g. remove all moves certain properties (produced certain result); to speed up the game could be used multi-moves, i.e. participants send in every move a fixed-length sequence of moves etc.; several test applications are in implementation.

Conclusions

This work analyzed the concept of randomness (*k*-randomness), applicable for use in devices with finite memory humans orcomputers. A method was presented (the loop disruption algorithm) for creating random sequences in gameplay; the quality of the created randomness was tested in a series of plays against established sources of randomness. Tests show, that the randomness is quite on a par with established sources of random numbers. As a practical use of game-created randomness is shown how this could be used for generating secure encryption keys for symmetric encryption without using the open-key procedure, typically used for creating common random sequence; the introduced 'moves combination method' is currently under implementation. Using a generated in game randomness for symmetric encryption makes such a communication systems very secure – they do not depend on any 'upper-level' security principals or certificates for key creation.

References

[1] FAO 2019. The future of food and agriculture: Trends and challenges. Food and Agriculture Organization of the United Nations, ISSN 2522-7211

[2] Sophia Chen. Why are countries creating public random number generators? Science, Jun 28, 2018

[3] BSA. Encryption: Why It Matters. Retrieved May 9, 2018 from http://encryption.bsa.org/

[4] Fortinet 2018. Data Breaches Are A Growing Epidemic. How Do You Ensure You're Not Next? Retrieved May 08,2018 from https://www.fortinet.com/blog/threat-research/data-breaches-are-a-growing-epidemic--how-do-you-ensure-you-re-n.html

[5] Nist 2016. Entropy as a Service. https://csrc.nist.gov/projects/entropy-as-a-service

[6] EaaSP 2018. EaaSP - Entropy as a Service Protocol. https://github.com/usnistgov/EaaS

[7] David Maimon et all 2019. SSL/TLS Certificates and Their Prevalence on the Dark Web – Venafi https://www.venafi.com/sites/default/files/2019-02/Dark-Web-WP.pdf

[8] https://www.random.org

[9] vanheusden.com 2018. audio entropy daemon. https://vanheusden.com/aed

[10] vanheusden.com 2018. video_entropyd. https://vanheusden.com/ved/

[11] Generating Random Data in Python. https://realpython.com/python-random/

[12] How to generate random numbers, characters, and sequences in Scala. https://alvinalexander.com/scala/how-to-generate-random-numbers-characters-sequences-in-scala

[13] CryptGenRandom. https://docs.microsoft.com/en-us/windows/desktop/api/wincrypt/nf-wincrypt-cryptgenrandom

[14] RtlGenRandom function. https://docs.microsoft.com/en-us/windows/desktop/api/ntsecapi/nf-ntsecapi-rtlgenrandom

[15] Common Language Runtime (CLR) overview. https://docs.microsoft.com/en-us/dotnet/standard/clr

[16] A Provable-Security Analysis of Intel's Secure Key RNG. https://eprint.iacr.org/2014/504.pdf

[17] Gagallium: How I found a bug in Intel Skylake processors. http://gallium.inria.fr/blog/intel-skylake-bug/

[18] The Register. Torvalds shoots down call to yank 'backdoored' Intel RdRand in Linux crypto. Sept 10, 2013, https://www.theregister.co.uk/2013/09/10/torvalds_on_rrrand_nsa_gchq/

[19] G. T. Becker, F. Regazzoni, C. Paar, W. P. Burleson. Stealthy Dopant-Level Hardware Trojans. Journal of Cryptographic Engineering, April 2014, Volume 4:1, pp 19-31

[20] M. Ender, S. Ghandali, A. Moradi, C. Paar. The First Thorough Side-Channel Hardware Trojan. https://eprint.iacr.org/2017/865.pdf

[21] arstechnica. “We cannot trust” Intel and Via’s chip-based crypto, FreeBSD developers say. https://arstechnica.com/information-technology/2013/12/we-cannot-trust-intel-and-vias-chip-based-crypto-freebsd-developers-say/

[22] https://www.random.org

[23] ISO/IEC 9899:2011. https://www.iso.org/standard/57853.html

[24] John von Neumann, “Various techniques used in connection with random digits,” in A. S. Householder, G. E. Forsythe, and H. H. Germond, eds., Monte Carlo Method, National Bureau of Standards Applied Mathematics Series, vol. 12 (Washington, D.C.: U.S. Government Printing Office, 1951): pp. 36-38

[25] I. Ekeland. The Broken Dice, and Other Mathematical Tales of Chance. University of Chicago Press 1993, pp. 1-190, ISBN: 9780226199924

[26] H. G. Rice. "Classes of Recursively Enumerable Sets and Their Decision Problems". Trans. Amer. Math. Soc. 74 1953, pp. 358-366

[27] H. Krawczyk. How to predict congruential generators. Journal of Algorithms, Vol. 13:4, December 1992, pp. 527-545

[28] J. Stern. Secret linear congruential generators are not cryptographically secure. 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), DOI: 10.1109/SFCS.1987.51

[29] NIST SP 800-22. A Statistical Test Suite for Random. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

[30] R. G.Brown. Dieharder: A Random Number Test Suite. https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[31] ENT. A Pseudorandom Number Sequence Test Program. http://www.fourmilab.ch/random/

[32] R. Halprin, M. Naor. Games for Extracting Randomnes. SOUPS '09 Proceedings of the 5th Symposium on Usable Privacy and Security 2009

[33] M. Alimomeni, R. Safavi-Naini. Human Assisted Randomness Generation Using Video Games. https://eprint.iacr.org/2014/045.pdf

[34] Henno, J., Jaakkola, H., Mäkelä, J. Using games to understand and create randomness. SQAMIA2018 - Proceedings of the 7th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, Vol. 2217, CEUR-WS, http://ceur-ws.org/Vol-2217/

[35] E. Ben-Porath. Repeated Games with Finite Automata. Journal of Economic Theory 59:1, 1993, pp. 17-32

[36] Sergio B. Volchan. What Is a Random Sequence? https://www.maa.org/sites/default/files/pdf/upload_library/22/.../Volchan46-63.pdf

[37] A Quilliot. Cyclic Orders. European Journal of Combinatorics 10:5, 1989, pp. 477-488

[38] Gang Beasts Controls Guide. https://www.gameskinny.com/ly5jv/gang-beasts-controls-guide

[39] Shai Ben-David et all 2019. Learnability can be undecidable. Nature Machine Intelligencevolume 1, pp 44-48, https://www.nature.com/articles/s42256-018-0002-3

[40] The World Rock Paper Scissors Association. https://www.wrpsa.com/

[41] USA Rock Paper Scissors League. https://myspace.com/usarps

[42] Rock Paper Scissors tournament rules. https://do317.com/p/rpsrules

[43] Rock Paper Scissors Programming Competition. http://www.rpscontest.com/

[44] Kolmogorov, A. N. (1965) Three Approaches to the Quantitative Definition of Information. Problems Inform. Transmission. 1(1), pp. 1-7

[45] Martin-Löf, P. (1966) The definition of random sequences. Information and Control. 9 (6): 602-619

[46] Schnorr, C. P. (1971) A unified approach to the definition of a random sequence. Mathematical Systems Theory. 5 (3), pp. 246-258

[47] Jaak Henno (2015) Information and Information Security. Information Modelling and Knowledge Bases XXVII, pp. 103-120

[48] B. A. Trakhtenbrot, Ya. M. Barzdin 1973. Finite Automata. Behavior and Synthesis. North-Holland, p. 211

[49] Ziv, J.; Lempel, A. (1978) Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory. 24 (5): 530

[50] A. Cobham, Uniform tag sequences, Math. Systems Theory, 6 (1972), pp 164-192

[51] H. Matsushima. Bounded Rationality in Economics: A Game Theorist's View. The Japanese Economic Review (1997), 48:3, pp 293-306

[52] Z. Wang, B. Xu, H. Zhou. Social cycling and conditional responses in the Rock-Paper-Scissors game. eprint arXiv:1404.5199, https://ui.adsabs.harvard.edu/#abs/arXiv:1404.5199

[53] http://staff.ttu.ee/~jaak/games

[54] GA. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review. (1956) 63, pp. 81-97