User Controlled Encryption Using Automated Key Generation

[1]Halima Abdel Halim Shnishah, [2]David Mulvaney
[1,2](School of Electronic, Electrical and Systems Engineering, Loughborough University LE11 3TU, UK)
H.Shnishah@lboro.ac.uk, d.j.mulvaney@lboro.ac.uk

ABSTRACT

As the practical implementation of many encryption algorithms is controlled by third parties, end users cannot be certain that the generated keys have not been compromised at source. By allowing users to generate their own keys, this concern can be overcome. In the work described in this paper, the encryption key is a generated function produced by an evolutionary search operation executed by Eureqa in modelling pseudo-random input data obtained from a suitable source. This paper describes the results of initial experiments to generate suitable keys representing random number sequences of a range of lengths.

KEYWORDS

encryption algorithms, automated key generation.

1 INTRODUCTION

Kerckhoffs' principle formulated in the 19th century states that a 'cryptosystem should be secure even if everything about the system, except the key, is public knowledge'; rephrased more recently by Shannon as 'the enemy knows the system' [1]. Such statements are not entirely true in modern encryption systems, as users rely on an algorithm whose implementation is formulated by a third party, often a commercial entity, government institution or military establishment. A concern for end users is that it cannot be guaranteed that a key is not retained by that third party or communicated to an authority [2][3]. Only by allowing the end user control over key generation can they have peace of mind that an intruder has not compromised the process. The perception that knowledge of the key generation method is a potential source of security breeches has led a number of users with specific security requirements to implement their own proprietary encryption algorithm or software so that this can also be hidden from potential intruders [4].

Another challenge faced by encryption users is that, due to the complexity of encryption algorithms, many end users have little choice but to employ third party algorithms or service providers to generate keys [5], who then distribute them and devise an appropriate method for their storage [6]. The key management process performed by third parties is an important security consideration, particularly when a number of different users are serviced by a single provider and a secondary form of authentication controlled by the provider is needed to access individual areas [7][8].

This paper describes new results based on an approach initially developed by Blackledge *et al.* [9], in which the key is an equation that is automatically generated so as to model a random number sequence to a suitable accuracy. By this method, users will be able to retain control of the key generation algorithm and so have greater confidence that the key is not accessible by third parties. The principal drawback of the approach is currently the time taken to generate a new key and the contribution of this paper is the investigation of different random sequence lengths in order to establish tradeoffs between calculation time and confidence in security.

The paper is organized as follows. Related work that has investigated the generation of bespoke keys is discussed in Section 2. Section 3 introduces the methodology used in this paper to generate keys, Section 4 describes the experimental results and Section 5 presents the conclusions.

2 RELATED WORK

A number of authors have reported work that generates encryption keys using processes that can be controlled by an end user.

Eureqa [10] uses an evolutionary computing approach to iteratively develop nonlinear functions to describe complex input signals usually connected

with experimental data. Blackledge *et al.* [9] used Eureqa to implement an encryption key generation algorithm that models a 250 sample pseudo random number (PRN) sequence [11]. A single proof of concept result was given in which 23 hours were needed to generate a key, namely an equation that models a random number sequence. The paper concluded that the technique may present a technical solution to the 'democratisation of the cipher bureau.'

Considering the above mentioned works, the present study will extend the initial investigations conducted by Blackledge *et al*. (2013). The reasons for using Blackledge as related work for this paper to develop an encryption method whose operation is both known and controlled by users rather than by third parties.

3 METHODOLOGY

The current work aims to develop further the approach of Blackledge *et al.* [9] in allowing the user to generate their own keys that are Eureqa models of a PRN sequence [9]. The approach follows the steps shown in Figure 1 to produce the encryption key.

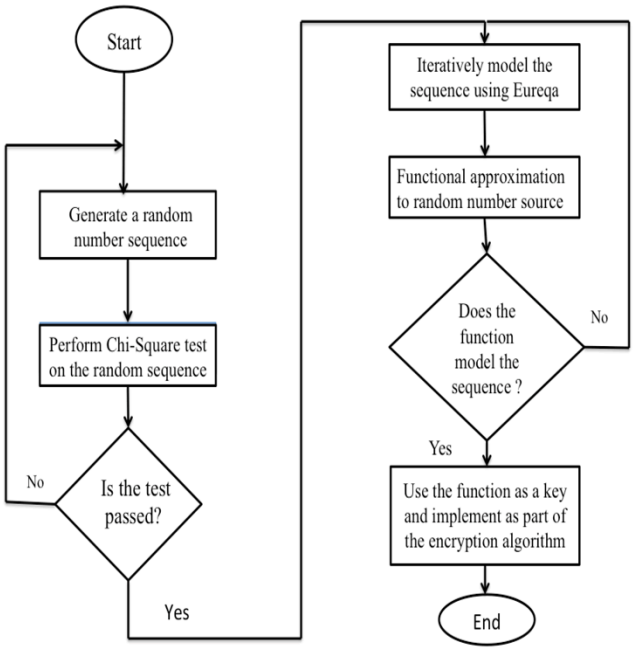

Figure 1 Method of generating encryption keys using Eureqa

3.1 Pseudo random number (PRN) generation

As the key itself will be an equation that models a sequence of random numbers to a suitable accuracy, an appropriate method is needed to generate and test random numbers to ensure they have the necessary characteristics of independence, unpredictability, and 'complexity' to provide strength to cryptographic algorithm [9]. A chi-square test can be used to assess whether the random sequence produced is likely to have come from a specified distribution [3].

Blackledge *et al.* established rules to be followed to obtain suitable PRNs for encryption purposes [12]. PRNs require a 'seed' to initiate the generation of the first number and subsequent values are generated iteratively [13]. The linear congruential generator (LCG) [14] is one of the oldest PRN algorithms and is commonly used in encryption [15] [16]. It is a simple method and the short period between sequence repetitions is generally regarded as a weakness [14]. This method produces integer random numbers stream in the range [0, *m*-1] where in practical implementations *m* is normally the word length of the computer system. The LCG uses the following iterative equation to generate the random values x_i [12].

$$x_{i+1=}(ax_i + c) \bmod m \qquad (1)$$

where $i = 0,1,2,..,n-1,$ n is the length of cycle, x_0 is the seed, a is an integer multiplier and c is an integer additive constant. These generators are characterized by simplicity in implementation and short calculation time [17] [18].

3.2 Key generation using Eureqa

Hod Lipson of the Computational Synthesis Laboratory at Cornell University first released Eureqa in 2009 [19]. Eureqa is able to perform 'symbolic regression', namely to execute iterative search operations to determine an underlying equation that models input data. The main goal of the tool was to reduce the time and effort needed to generate such equations using trial and error [20]. An early success for Eureqa was in determining the basic laws of motion of a double pendulum in a few hours by analyzing data that described its movements [20].

In this paper, Eureqa is used to determine a mathematical equation to model PRNs and to act as the encryption key. The results investigate the ability of Eureqa to model PRN sequences of different lengths in order to establish trade-offs between execution time and encryption security.

4. EXPERIMENTAL RESULTS

All of the calculations in this section were carried out on an Intel i7 computer running a 64-bit Windows operating system. During the experiments, Eureqa performed an evolutionary search to determine an equation to model the values in a sequence of PRNs. To ensure that all the integer points are being accurately modelled, each must be fitted at an error of no more than 0.5. The assumption here is that the resolution and accuracy of the computer calculations is sufficient in both the sending and receiving systems that the same rounding to the nearest integer will occur in each.
A typical example of a non-linear equation generated by Eureqa to represent an encryption key is shown below. It is important to note that a different equation will be generated by Eureqa each time it is executed even though it may be attempting to model an identical sequence of data values.

y=530.3+500.8*cos(4.33-0.047*x-0.386*os(-21*x))
-0.071*x-2.466*cos(-21*x)*cos(4.33-0.0465*x-
0.386*cos(-21*x))-22.62*sin(533.3*cos(x))*cos
(cos(4.332-0.0465*x-0.386*cos(21* x))) (2)

The next section shows the results of several experiments to determine the calculation time, generate an acceptable key and to meet the maximum error requirement.

Example 1 The following LCG was used to generate a PRN sequence of 200 three-digit values and with an initial value of $x_0 = 131$

$$x_{i+1=}(13x_i + 0) \bmod 997,$$
$$\text{i=0,1,2,..,199} \qquad (3)$$

Table 1 shows a series of experiments conducted using the PRN sequence described in equation 4. It can be seen that only the first experiment was successful in that it was able to model all the values in the sequence (the Eureqa model exhibited a maximum error of less than 0.5). However, over 36 hours of calculation time were needed to produce this result, a time too long for most practical uses. It can be seen that none of the other five tests produced a successful model and all of these experiments were halted after a suitable period time had passed and no further convergence was apparent. It can be seen that using this method there is no guarantee that a key will be successfully produced or that the key will be generated in a reasonable time. Note that in Table 1, the values shown for key length are the number of characters in the corresponding Eureqa equations.

Table 1 Calculation time needed to obtain key values for a sequence of 200 PRNs

Experi-ment	calculation time	maximum error	usable?	key length (bytes)
1	36h:20min	0.48	yes	487
2	32h:34min	0.65	no	487
3	57h:57min	8.44	no	222
4	53h:39min	0.57	no	216
5	59h:40min	56.77	no	270
6	74h:03min	0.51	no	495

A second experiment was carried out in which the length of the sequence was reduced to 10 values and the results of these experiments are shown in Table 2.

Table 2 Calculation time needed to obtain key values for a sequence of 10 PRNs

Experi-ment	calculation time	maximum error	usable?	key length (bytes)
1	34min	0.57	no	208
2	18min	0.67	no	111
3	13min	0.53	no	159
4	21min	0.50	yes	130
5	12min	0.29	yes	120
6	19min	5.03	no	132
7	17min	0.34	yes	125

In this case, three of the seven experiments were successful in converging to a suitable Eureqa model, while the calculation time has been considerably reduced. The calculation time may now be sufficiently small for some practical uses. To prevent the pursuit of unfruitful avenues of

investigation, it would perhaps be prudent to establish an upper bound on the acceptable calculation time; a period after which the computations would be abandoned and the process restarted.

Example 2 The following LCG was used to generate a PRN sequence of 200 six-digit values and with an initial value of $x_0 = 131$

$$x_{i+1=}(1597x_i + 51749) \bmod 233944, \quad i = 0,1,2,..,199 \qquad (4)$$

This LCG was described by Press *et al*. (cited in [21]) with the specific intention of being capable of generating long cycles of random values. In the current implementation, Eureqa generated models from a sequence of 20 PRNs and the results are shown in Table 3.

Table 3 Calculation time for a sequence of 200 values obtained from a LCG

Experi-ment	calculation time	maximum error	usable?
1	12h:25min	1893.7	no
2	70h:12min	1751.6	no

It can be seen that the maximum error was very large and little convergence was observed even though the calculations were allowed to proceed for an extended period of time. The allowed values in the sequences produced by the PRN in equation 4 range from 0 to 244943. It became apparent in the experiments that the Eureqa model calculation time performance generally worsened as the PRN range was increased. In many applications, only an ASCII range of 0 to 127 is needed and so the PRNs can normally be rescaled to this range. Table 4 shows the modeling results when the allowed range of the values generated in equation 4 was reduced to the 7-bit ASCII range.

Table 4 Calculation time for an LCG sequence of values in the ASCII range

experiment	calculation time	maximum error	usable ?	key length (bytes)
1	36h:20min	0.48	yes	487
2	32h:34min	0.65	no	487
3	35h:50min	0.51	no	489

Although only one of the experiments produced an acceptable maximum error, restricting the PRN range has significantly reduced both the maximum error and the calculation time.

5. CONCLUSION

This paper has presented a number of initial results that have been obtained to investigate the viability of an approach that allows users to control the generation of encryption keys. Initial investigations are reasonably promising in that Eureqa is able to model random data sequences, although improvements in reliability and a reduction in calculation time will be required for practical usage. Clearly there are trade-offs to be made, since a reduction in the length of the sequence may lead to a less secure individual key, yet will allow a replacement key to be generated more rapidly and so improve security. Future work will look to investigate enhancements aimed at reducing the calculation time while making the key generation process more reliable.

REFERENCES

[1] C. Shannon, "Communication Theory of Secrecy," *Bell System Technical Journal*. [Online]. Available: https://archive.org/stream/bstj28-4-656#page/n5/mode/2up. [Accessed: 12-Aug-2015].

[2] J. M. Blackledge, D. A. Dubovitskiy, and M. Iet, "A Covert Encryption Method for Applications in Electronic Data Interchange," *ISAST J. Electron. Signal Process.*, vol. 4, no. 1, pp. 107 – 128, 2009.

[3] B. Schneier, "Security pitfalls in cryptography," *Schneier on Security*, 1998. .

[4] H. Handschuh, B. Preneel, and K. U. Leuven, "Minding Your MAC Algorithms ?," *Inf. Secur. Bull.*, vol. 9, no. July, pp. 213–220, 2004.

[5] T. Lalith, "Key Management Techniques for Controlling the Distribution and Update of Cryptographic keys," *IJACSA Int. J. Adv. Comput. Sci. Appl.*, vol. 1, no. 6, pp. 163–166, 2010.

[6] E. Dawson, A. Clark, and M. Looi, "Key management in a non-trusted distributed

environment," *Futur. Gener. Comput. Syst.*, vol. 16, no. 4, pp. 319–329, 2000.

[7] L. Harn and H. Y. Lin, "Key management for decentralized computer network services," *IEEE Trans. Commun.*, vol. 41, no. 12, pp. 1777–1779, 1993.

[8] K. Prabha and S. Nalini, "A secure data forwarding in cloud storage," *2013 Int. Conf. Opt. Imaging Sens. Secur. ICOSS 2013*, 2013.

[9] S. Bezobrazov, J. Blackledge, P. Tobin, and F. Zamora, "Cryptography using evolutionary computing," *24th IET Irish Signals Syst. Conf. (ISSC 2013)*, pp. 21–21, 2013.

[10] DaSilva, "Eureqa! Signs of the Singularity?," *Humanity+*, 2011. [Online]. Available: http://hplusmagazine.com/2011/03/25/eureqa-signs-of-the-singularity/. [Accessed: 12-Aug-2015].

[11] "True Random Number Service," *RANDOM.ORG:*, 2013. [Online]. Available: https://www.random.org.

[12] J. Blackledge, *Cryptography Using Steganography : New Algorithms and Applications Cryptography and Steganography :* Warsaw: Centre for Advanced Studies, Warsaw University of Technology, 2011.

[13] P. Kohlbrenner and K. Gaj, "An embedded true random number generator for FPGAs," *Proceeding 2004 ACM/SIGDA 12th Int. Symp. F. Program. gate arrays - FPGA '04*, p. 71, 2004.

[14] F. James, "A review of pseudorandom number generators," *Comput. Phys. Commun.*, vol. 60, no. 3, pp. 329–344, Oct. 1990.

[15] E. E. Schultz, R. W. Proctor, M.-C. Lien, and G. Salvendy, "Usability and Security An Appraisal of Usability Issues in Information Security Methods," *Comput. Secur.*, vol. 20, no. 7, pp. 620–634, Oct. 2001.

[16] J. Gait, "A New Nonlinear Pseudorandom Number Generator," *IEEE Trans. Softw. Eng.*, vol. SE-3, no. 5, 1977.

[17] J. Eichenauer-Herrmann, "On the autocorrelation structure of inversive congruential pseudorandom number sequences," *Stat. Pap.*, vol. 33, no. 1, pp. 261–268, 1992.

[18] D. Knuth, *The art of computer programming,*. Addison-Wesley, 1981.

[19] L. Edwards, "ureqa, the robot scientist (w/ Video)," *phys.org*. [Online]. Available: http://phys.org/news/2009-12-eureqa-robot-scientist-video.html. [Accessed: 09-Aug-2015].

[20] D. Salisbury, "Robot biologist solves complex problem from scratch," 2011. [Online]. Available: http://news.vanderbilt.edu/2011/10/robot-biologist/. [Accessed: 09-Aug-2015].

[21] R. Seyde, *Tools for Computational Finance*, 4th ed. Business Media: springer Science, 2009.