IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 4, Ver. V (Jul.-Aug. 2016), PP 42-46
www.iosrjournals.org

Auto Mutating Cryptosystem- An approach towards better security

Harsha S[1], N Bhaskar[2], M N Sheshaprakash[3], G Raghavendra Rao[4]
[1.] Assistant Professor, Information Science &Engineering, Jyothy Institute of Technology, Bengaluru, Affiliated to VTU, Belagavi, Karnataka, India
[2.] Professor & Head, Department of Engg. Maths, VVCE, Mysuru, Affiliated to VTU, Belagavi, Karnataka, India
[3.] Professor & Head, Department of Civil Engineering, VVIET, Mysuru, Affiliated to VTU, Belagavi, Karnataka, India
[4.] Professor, Department of Computer Science & Engineering, NIE, Mysuru, Affiliated to VTU, Belagavi, Karnataka, India

Abstract: *The concept of learning systems has been an intriguing one since first appearance of Artificial Intelligence (AI) in literature. With the progress in each decade of research, new systems have been developed to suit a host of requirements and environments. However the idea of AI with a cryptosystem is relatively novel and is less explored. This paper discusses the implementation and analysis of multiple cryptosystems with an AI engine to provide better security for information. Here the attempt is made to combine number of cryptosystems with an AI engine to understand a situation where a hacker is trying to cryptanalyse data and prevent it from happening.*

Keywords: *Artificial Intelligence, Data Security, Mutating Algorithms, Learning Systems*

I. Introduction

Artificial Intelligence [1] and learning systems [2], has given rise to many new and rapidly expanding domains, of which the most promising are Mutating algorithms. The idea of an auto mutating [3] cryptosystem hence forth termed as "***AMCS***", as proposed is implemented using the results of DIGgER [9], Pythocrypt [11], MACCS [10] and 3-d Pythocrypt [12] . The systems designed in each paper contribute to the final solution that is presented in this paper. The frame work should be able to learn over time [4], the parameters that define a user's behavior and use the knowledge to build a profile for each user. With the aid of this profile, the system will then be able to validate the legitimacy of the user. In case of an attempt to hack, the system has to choose from a set of cryptosystems available based on certain rules to avoid redundancy and successively encrypt the data to fend off the threat. This is achieved using a non-linear learning system DIGgER [9].

II. Algorithm

The system proposed is described in this section as an abstract level algorithm.

- Start on OS start up as Kernel script (Non accessible for users once installed, request restart)
- On start ask for registration with Communication systems (mobile phone number or/and email id)
- Run dialog for files to be encrypted (run Encrypt subroutine)
- On encryption ask for password
- If entered password= any previous password by more than 50%, ask to enter a new password
- Confirm password
- Lock cipher text and key file with password
- Provide CHAP [5] (run Setup_CHAP (security question) subroutine)
- Send file_id, password and CHAP response to registered mobile and email
- Delete plain text files selected for Encryption permanently
- Wait
- On read/copy request (file_id) Run DIGgER on User behavior (run DIGgER subroutine)
- Set conditions (cond 1= Is user behavior erratic (change of more than 15% of normal), cond2= wrong password)
- If cond1 && cond2== TRUE, then run Secure_data subroutine
- Else if cond1 **EXOR** cond2==TRUE run CHAP subroutine
- On CHAP==FALSE, run Secure_data subroutine
- Else run Decrypt subroutine
- Wait

- On copy request (file_id) run CHAP subroutine
- On CHAP==FALSE, run Secure_data subroutine
- Else run OTP subroutine
- On OTP== FALSE, run Secure-data subroutine
- Else set “read-write-execute permissions for file-id to 111” and Wait

Subroutines
Encrypt subroutine

- Select file-id (browse or enter path or pick file from Secure_data subroutine
- Read the list of encryption systems available(Get number N)
- Generate a random number from 1 to N, n
- Select encryption system using n
- If Selected Cryptosystem is already present in the list of used Encryption systems And it is already altered and used
- Generate another number and use the method at that number
- Else
- Pick one of these
- a different Key with the same algorithm or
- change the length of the plain text block and corresponding key or
- both
- Else select key (string or file dependant on the crypto system)
- Append to list of used Encryption systems for the file
- Save cipher text and key files in respective folders (created at start up)
- Delete plain text file
- Return

Setup_CHAP (security question) subroutine

- Select or create Challenge Question
- Enter response
- The response has to have minimum 2 words with 1 number and one special character
- Return

DIGgER subroutine

- Start counters for
  - Timestamp of each read request
  - Time to enter the password in each attempt
  - Number of wrong password attempts per file
  - Time on the machine before the read request
- Store the wrong passwords
- Compute % error in each attempt between the stored and entered passwords (Horspool method [8])
- Normalize all entries to range (0-1)

$V_2 – V_1 = D_1$ (1)
$V_3[p] = V_2 + D_1$ (2)
$V_3[p] – V_3 = D_{21}$ (3)
$V_3[p] + D_{21} = V_4[p_1]$ (4)
$V_3 + D_{21} = V_4[p_2]$ (5)
$V_4[p] = V_4[p_2] + D_{22}$ (6)

- Where V is the parameter considered, D_1 = Linear difference between V_2 and V_1, D_{22} = Min [Diff ($V_4[p_1]$,$V_4[p_2]$)], (7)
- p – Predicted Value and the indices indicate the iteration level
- D – Delta function
- Return (Difference)

Secure_data subroutine

- Run Encrypt Subroutine (Without UI) and replace the old cipher text file with new file.
- Send “threat detected” message with new method, key/key filed to the authentic user.

- return

CHAP subroutine
- Post CHAP challenge
- Read CHAP response
- If CHAP response != stored response return FALSE
- Else return TRUE

Decrypt subroutine
- Select filed (browse or enter path)
- Select encryption system
- Select/Enter key/keys (string or file dependant on the crypto system)
- Run corresponding Decryption algorithm
- Return plain text in Dialog box with file operation options

OTP subroutine
- Read CHAP response and Password
- Take the total length of the two strings $L=(L_{cr} + L_p)$
- Generate 4 random numbers between 1 and L
- Pick the characters at each number
- Send array to user's mobile and email
- Store for comparison in designated file
- After 1 comparison delete the contents of the file
- Return TRUE if equal
- Else return FALSE

III. Implementation

The algorithm is implemented using the systems developed in the previous papers. Each system developed in those papers becomes a module for AMCS.

DIGg ER module

This module runs DIGgER [9] algorithm on data generated by user login/decrypt attempts. The data set includes login time(s), keystroke rate(/s), file search time(s), key search time(s) and percentage error in password. Using these parameters, the module builds a profile for the user over time [7]. The profile is used to compute DIGgER coefficient [12] as shown in Table 1. Higher value of this parameter indicates that a user is likely to be malicious.

Search module

Search module uses MACCS [10] to obtain key files based on user's choice for encrypt / decrypt module. It uses Horspool algorithm [8] to eliminate repetitions of key files and algorithms. When a user wishes to encrypt some file, this module searches a random file or generates a random number to act as the key for the selected cryptosystem. The context parameter of MACCS [10] eliminates redundancy.

Encrypt/Decrypt Module

This is the data security module of AMCS. Here a multitude of algorithms can be implemented. For this research work, four algorithms including ***"3-d Pythocrypt"*** [12]are implemented. It runs on user's direction as well as autonomously in case of an attack.

Mutation

By using this module, the system can change its behavior and form to adapt itself to changing conditions which is the core of AMCS. Here a fuzzy limit set on the DIGgER coefficient (Table 1) is the triggering mechanism for mutation. Mutation occurs once it is found that the coefficient is high. The changing parameters are: Block length or/and Key value/file or/and Algorithm. Each change is logged and learnt by the system for developing knowledge of the attacks. The coefficient is plotted against attempts in Fig. 1. The mutation of the algorithm is mostly autonomous. Although the parameters that can be changed are hardcoded in the current system, the choice of selecting them is left to the system and also the mutation chosen for them is decided by AMCS.

Complexity

Each attempt of a hacker to decrypt a file on AMCS resulting in failure, a report is sent to the user after encrypting the encrypted file again. This successive encryption increases the complexity of cryptanalysis exponentially. This feature is clearly shown in Table 3. Even though for the purpose of this research brute force method is considered as cryptanalysis mode, any method used would fail with successive encryptions. Even an attempt to copy would trigger this mutation. This nature of AMCS provides enhanced security to block data in the best way possible.

IV. Analysis

The system is tested for its feature to increase the difficulty of cryptanalysis. Human login/decrypt attempts are recorded and each time the difficulty is measured using brute force method [6]. The results are shown in Table 3. It can be clearly seen that immediately after 2 attempts, the cryptanalysis time surpasses basic computational limits. Also the same set is run with a script to attempt copying/cryptanalysing the files encrypted by AMCS. The login attempts themselves are foiled by AMCS due to the high keystroke rate. A sample of the parameters generated is shown in Table 2. It is in sharp contrast to Table 1 as the keystroke rate and percentage error parameters are very high. They contribute to a high value of DIGgER coefficient. The script is written to check strings for correctness with bitwise EX-OR. This enables the script to select correct letters from the trials strings and their position in the password. Even though successive attempts decrease the error, the time required to cryptanalyse increases exponentially making it infeasible to obtain the plain text file.

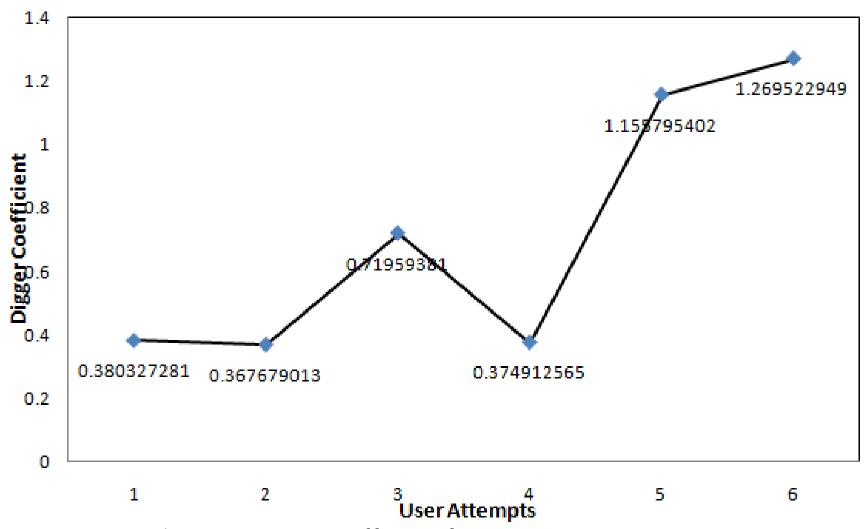

Fig. 1. DIGgER coefficient for Login/Decrypt attempts

Table 1. DIGgER subroutine running on user login/decrypt request data with DIGgER coefficient

Parameters	Parameter titles	Login 1	Login 2	Login 3	Login 4	Login 5	Login 6
	Login time (s)	3.1425	3.1832	3.4101	3.3021	4.0016	3.9167
	File access time (s)	3.4572	3.1272	3.2911	3.1596	5.3514	6.1158
	Password entry time(s)	4.0961	3.862	3.6128	3.7625	4.6147	4.9816
	Keystroke rate (per sec)	2.41	2.49	2.502	2.487	1.621	1.314
	Key file access time (s)	3.2655	3.3548	3.1988	3.4039	5.0148	5.3674
	%Error in password	0	0	12	0	19	21
Normalised Parameters	**Login time**	0.7853	0.7955	0.8522	0.8252	1.0000	0.9788
	File access time	0.5653	0.5113	0.5381	0.5166	0.8750	1.0000
	Password entry time	0.8222	0.7753	0.7252	0.7553	0.9263	1.0000
	Keystroke rate	0.9632	0.9952	1.0000	0.9940	0.6479	0.5252
	Key file access time	0.6084	0.6250	0.5960	0.6342	0.9343	1.0000
Result	**DIGgER coefficient**	**0.3803**	**0.3677**	**0.7196**	**0.3749**	**1.1558**	**1.2695**
Fuzzy limits	**Result (IS AUTHENTIC?)**	**YES**	**YES**	**MOSTLY**	**YES**	**MAYBE**	**NO**
Subroutine	**Action to be taken**	**Decrypt**	**Decrypt**	**Pose CHAP**	**Decrypt**	**Pose CHAP, Secure data**	**Alert Threat, Secure Data**

Table 2. DIGgER subroutine running on script login/decrypt request data

Parameters	Parameter titles	Login 1	Login 2	Login 3	Login 4	Login 5	Login 6
	Login time (s)	0.1223	0.2088	0.1987	0.1205	0.2376	0.2014
	File access time (s)	3.4572	3.1272	3.2911	3.1596	5.3514	6.1158
	Password entry time(s)	0.8913	1.0129	1.0028	1.0165	1.1897	1.3001
	Keystroke rate (per sec)	382.6	411.2	426	389.9	461	452.1
	Key file access time (s)	1.1925	1.0456	1.1048	1.2014	1.2896	1.523
	%Error in password	86	54	42	38	31	27
Normalized Parameters	Login time	0.5147	0.8788	0.8363	0.5072	1.0000	0.8476
	File access time	0.5653	0.5113	0.5381	0.5166	0.8750	1.0000
	Password entry time	0.6856	0.7791	0.7713	0.7819	0.9151	1.0000
	Keystroke rate	0.8191	0.8803	0.9120	0.8347	1.0000	0.9679
	Key file access time	0.7830	0.6865	0.7254	0.7888	0.8467	1.0000
Result	DIGgER coefficient	2.8599	1.9795	1.6317	1.4677	1.4402	1.3629
Fuzzy limits	Result (IS AUTHENTIC?)	NO	NO	NO	NO	NO	NO
Subroutine	Action taken	Alert Threat, Secure Data	Alert Threat, Secure Data	Alert Threat, Secure Data	Alert Threat, Secure Data	Alert Threat, Secure Data	Alert Threat, Secure Data

Table 3. Increasing time for cryptanalysis with each wrong attempt

Parameter	Attempt 1	Attempt 2	Attempt 3	Attempt 4	Attempt 5	Attempt 6
Login time (s)	3.2832	3.1445	3.4108	3.3021	3.401	3.2123
Algorithm chosen (random)	1 (RSA)	4 (3-d Pythocrypt)	3 (Steganography)	2 (DES)	1 (RSA)	3 (Steganography)
Key size (bytes)	128	683	8192	56	128	65536
Decryption time (s)	0.4176	0.7823	0.9128	1.3435	1.9027	2.4134
Cryptanalysis time (s)	3.40282E+38	4.0132E+205	∞	∞	∞	∞
Note:	1. For the experiment 4 cryptosystems including 3-d Pythocrypt are used					
	2. The cryptosystems and keys/key files are selected randomly for a 1 kB file					
	3. Cryptanalysis time is computed for Brute force method					

V. Conclusion

From the analysis of the results it can be concluded that the Autonomous nature of AMCS is a prime contributor to the enhanced security provided to information. Since the user gets the information via varied channels, the probability of a hacker getting all of the same becomes physically infeasible. Whereas the feature to have multiple cryptosystems as options implies that any attempt to cryptanalyze fails. Every failing attempt only increases the complexity and time required to cryptanalyze the data. Thus AMCS is more powerful, if not final solution to the ever increasing security threats to information security.

References

[1]. Holland, John H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press. ISBN 0-262-58111-6

[2]. Haugeland, John (1985). *Artificial Intelligence: The Very Idea*. Cambridge, Mass.: MIT Press. ISBN 0-262-08153-9

[3]. Kahneman, Daniel; Slovic, D.; Tversky, Amos (1982). *Judgment under uncertainty: Heuristics and biases*. New York: Cambridge University Press. ISBN 0-521-28414-7.

[4]. Asada, M.; Hosoda, K.; Kuniyoshi, Y.; Ishiguro, H.; Inui, T.; Yoshikawa, Y.; Ogino, M.; Yoshida, C. (2009). "Cognitive developmental robotics: a survey". *IEEE Transactions on Autonomous Mental Development 1 (1): 12–34*

[5]. Andrew S Tenenbaum, (2006) "*Computer Communication Networks*", McGraw Hill, Revised 4th edition

[6]. Abraham Sinkov, 1966(e-book)"*Elementary Cryptanalysis: A Mathematical Approach*" Mathematical Association of America

[7]. Hinton, G. E. (2007). "Learning multiple layers of representation". *Trends in Cognitive Sciences **11**: 428–434*

[8]. R. N. Horspool (1980). "*Practical fast searching in strings*". Software - Practice & Experience 10 (6): 501–506.

[9]. Harsha S, N Bhaskar, Amith Kumar V P, Nibha J, "A Novel Method to Dig Information from Generic Groups of Enriched Records (DIgGER)" *I J C S S E I T, Vol. 5, No. 2, December 2012, pp. 243-248*

[10]. Poonam Ghuli, Swapna Rao P, Harsha.S, Rajashree Shettar; A NOVEL METHOD TO SEARCH INFORMATION THROUGH MULTI AGENT SEARCH AND RETRIEVE OPERATION USING CONTENT AND CONTEXT BASED SEARCH, *INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY (IJCET), Volume 4, Issue 3, May-June (2013), pp. 512-518*

[11]. Harsha S, N Bhaskar, Chandan CM "PYTHOCRYPT- A CRYPTO SYSTEM FOR MEDICAL IMAGES", *INTERNATIONAL JOURNAL OF ENGINEERING, SCIENCES AND MANAGEMENT, Volume 3, Issue 2, Jul-Dec 2013, pp. 48-52*

[12]. Harsha S, N Bhaskar, M N Sheshaprakash, "A 3-d advancement of PythoCrypt for any file type", *Springer JOOI-TMC, DOI 10.1186/s40852-015-0022-8*