International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 3, June 2020, pp. 3295~3306
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i3.pp3295-3306

Efficient two-stage cryptography scheme for secure distributed data storage in cloud computing

Rabab F. Abdel-Kader[1], Samar H. El-Sherif[2], Rawya Y. Rizk[3]
[1,3]Department of Electrical Engineering, Port-Said University, Egypt
[2]Faculty of Management Technology and Information Systems, Port-Said University, Egypt

Article Info

Article history:

Received Oct 23, 2019
Revised Dec 18, 2019
Accepted Jan 8, 2020

Keywords:

Authentication
Chaos model
Cloud computing security
Cryptography
Decryption/Encryption

ABSTRACT

Cloud computing environment requires secure access for data from the cloud server, small execution time, and low time complexity. Existing traditional cryptography algorithms are not suitable for cloud storage. In this paper, an efficient two-stage cryptography scheme is proposed to access and store data into cloud safely. It comprises both user authentication and encryption processes. First, a two-factor authentication scheme one-time password is proposed. It overcomes the weaknesses in the existing authentication schemes. The proposed authentication method does not require specific extra hardware or additional processing time to identity the user. Second, the plaintext is divided into two parts which are encrypted separately using a unique key for each. This division increases the security of the proposed scheme and in addition decreases the encryption time. The keys are generated using logistic chaos model theory. Chaos equation generates different values of keys which are very sensitive to initial condition and control parameter values entered by the user. This scheme achieves high-security level by introducing different security processes with different stages. The simulation results demonstrate that the proposed scheme reduces the size of the ciphertext and both encryption and decryption times than competing schemes without adding any complexity.

Corresponding Author:

Rabab F. Abdel-Kader,
Department of Electrical Engineering,
Port-Said University,
Port-Said, 42523, Egypt.
Email: rababfakader@eng.psu.edu.eg

1. INTRODUCTION

Cryptography is very important to convert original text (plaintext) into encrypted text (ciphertext) to keep sensitive data more secure. This process can be done by many encryption techniques, and the inverse of this process is done to get back the plaintext using the corresponding decryption techniques. Cryptography techniques use a secret key to encrypt and decrypt sensitive data. Cryptography achieves many security issues like data integrity, authentication, non-repudiation, and confidentiality, so it gains high importance [1].

Standard cryptography algorithms are classified into three main categories [2] as shown in Figure 1. Firstly, symmetric algorithms which use a single secret key for encryption and decryption. This key is known to the sender and receiver. There are many famous symmetric algorithms such as Data Encryption Standard (DES), 3DES, Advanced Encryption Standard (AES), Blowfish, RC6. Secondly, asymmetric algorithms which use two keys, the public key for encryption and private key for decryption. These algorithms can be described by high computational cost and slow speed in comparison with symmetric algorithms. There are many examples of asymmetric algorithms such as Elliptic Curve Cryptography (ECC), Diffie Hellman, RSA, and Elliptic Curve Digital Signature Algorithm (ECDSA). Third, Hash algorithms, compress data to be converted from arbitrary size to fixed size. Some examples of hash algorithms are MD5 and SHA.

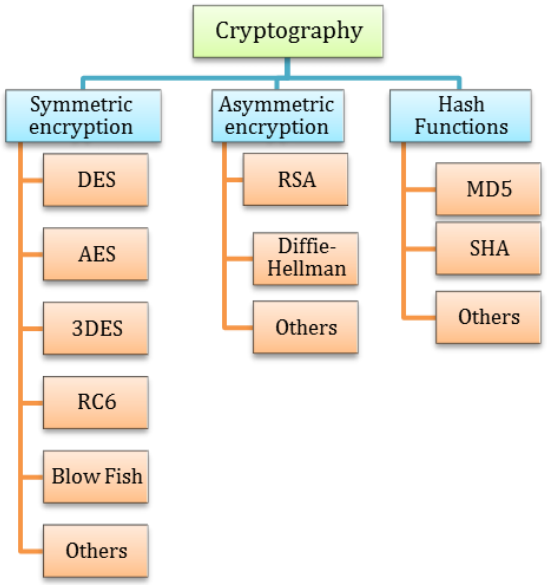

Figure 1. The three main categories of standard cryptography algorithms

Cloud computing Infrastructure as a Service (IAAS) presents an important service to users through the internet which called cloud storage [3-6]. There are many advantages to this service such as possible remote data access at any time and on-demand resource deployment. Data must be stored safely into cloud storage, so cryptography is very important. Encryption process must be done on files before storing into the cloud. In addition, authentication in cloud is a very important step which proves that the accessing user is the real user [7-9]. Many authentication schemes were proposed to authenticate users in cloud, but a lot of them had some security troubles. The previous work in authentication process suffered from some problems [10, 11]. These problems can be summarized in using mobile phone applications, using extra hardware, receiving message through the internet, and extra time to identity the user.

In this paper, a two-stage cryptography scheme for secure distributed data storage (TCS_DD) is proposed to encrypt files before sending them into the cloud. It includes both authentication and cryptography stages. First, a two- factor authentication scheme is presented to overcome the weaknesses found in existing authentication schemes. Second, the plaintext is divided into two parts sent to different cloud servers in order to guarantee more security. In each part, only half of the characters are encrypted. Each part is encrypted using a unique key that is generated using logistic chaos model theory. Users can download the two parts of his file at any time. Each character in each part is decrypted using its unique key which is generated from the same chaos equation used in the encryption process.

The organization of this paper is as follows: a brief overview of related work is presented in section 2. The proposed two-stage cryptography scheme is introduced in section 3. Section 4 shows the simulation results of the proposed scheme compared to the existing schemes. Finally, the main conclusions are presented in section 5.

2. RELATED WORK

Different cryptography schemes were proposed to keep data secure but each with its own limitations and drawbacks. Subarea in [12] presented a scheme which used ECC to encrypt the plaintext and then communicated with the destination through a secured channel. Simultaneously, MD5 was used to get the hash value from the same plaintext. This value was encrypted using DUAL RSA and also sent to the destination. This scheme achieved integrity by comparing the new hash value with the decrypted one and also achieved difficulty for attackers to extract the text from the encrypted one, but it used two asymmetric encryption algorithms (ECC and DUAL RSA) which exploit extra time for encryption.

Another scheme that used a combination of symmetric and asymmetric cryptography algorithms was presented in [13]. In this scheme, AES was used firstly for encryption and then ECC. MD5 was used to get the hash value from the ciphertext and then was compared with one which was evaluated at the destination. The decryption process was also performed by AES and ECC. This scheme required more time for encryption and decryption because two cryptography algorithms were used sequentially.

A hybrid of symmetric and asymmetric algorithms was proposed by Zhu in [14]. AES was used to encrypt the plaintext and ECC was used to encrypt the key and digital signature belonged to the AES algorithm. The key KAES belonged to the AES was used for one time only by the sender. At the receiver side, signature verification was done to obtain the original information. This scheme suffered from a low-security level.

A two-phase hybrid cryptography algorithm (THCA) for wireless networks was presented in [15]. This scheme splits the plaintext into two parts. The first part was encrypted by the AES algorithm and its key was encrypted using the ECC algorithm. Both are considered asymmetric cryptography algorithms. The second part was encrypted using the RSA algorithm. It also used the MD5 algorithm for integrity. The reverse of the previous steps was done for decryption and retrieving the plaintext.

A secure cluster-based routing protocol (SCBRP) is also presented for wireless sensor networks [16]. It uses adaptive particle swarm optimization (PSO) with optimized firefly algorithms during data transmission. Data in encryption form was sent to the sink node. The message that should be sent was divided into two sub-blocks. The first one was encrypted using AES algorithm, but the other one is encrypted using RC6 algorithm. MD5 algorithm was also used to achieve data integrity. SCBRP aimed to reduce energy consumption over an individual node to improve the whole network lifetime. Energy-efficient clustering, secure routing, and security verification were used to design SCBRP. However, this algorithm uses two symmetric algorithms which are considered a very weak point in its design.

Existing traditional cryptography algorithms are not suitable for cloud storage. There are some requirements must be achieved for the cloud computing environment such as secure access for data from the cloud server, small execution time, and low time complexity. Many recent encryption schemes were applied to data before uploading into cloud storage, some are discussed next.

Bansal in [17], presented a scheme to store files into the cloud storage in a secure manner. The user accesses the cloud storage safely using a unique key. Image matching was used for authentication for more security. Then, the user could upload his file into the cloud. To keep data safe in the cloud storage, the user divides the file into blocks and chose some bits from each one; these bits were encrypted using the ECC algorithm. After that, the file was stored in the cloud safely. In this scheme, Metadata was chosen from each block and encrypted using ECC, and then stored at the back of the file. Integrity could be achieved by comparing Metadata with the original file and making sure that no changes have occurred. In this scheme, the whole file was not encrypted, only some bits so it achieved less overhead, CPU power and execution time. This scheme did not achieve enough security for the whole data in the file. It might be useful only to check if data had been modified or not.

A hybrid of the symmetric and the asymmetric encryption algorithms for cloud computing security was proposed in [18]. After selecting one file to be uploaded into the cloud, the symmetric Blowfish algorithm is applied to encrypt the file. The secret key of the Blowfish algorithm was encrypted using the asymmetric RSA algorithm. After that, secure hash algorithm-2 (SHA-2) was applied to the encrypted file to generate the message digit. Next, the digital signature algorithm (DSA) was applied to the previously generated message digit. SHA-2 and DSA were used to achieve secure transmission and authorization. Using a combination of symmetric and asymmetric algorithms made the decryption process very difficult for hackers to attack. However, it suffers from high complexity because of using a hybrid of symmetric and asymmetric algorithms in addition to SHA-2 and DSA for authentication.

Chueh et al. designed and implemented a security system for cloud storage [19]. The scheme used the third party auditor (TPA) with the AES encryption algorithm for storing files into the cloud safely. Before uploading a file into the cloud, a 32-byte password is generated randomly and subjected to a password-based key derivation function 2 (PBKDF2) and HMAC-SHA256 to produce the derived encryption key. This key was used to encrypt the original file using the AES encryption algorithm. After the encryption process, the user could enter a password which would be considered a master key after applying (PBKDF2). This master key was used to encrypt the encryption key. Finally, the encrypted file was sent to the cloud storage provider (CSP), and the encrypted encryption key was sent to the TPA. Using the TPA for storing the encrypted key of the encrypted file prevented any user or CSP itself from decrypting the encrypted file. This scheme achieved better key management in addition to safer verification. Any time, the user could get the encrypted file from the CSP and the encryption key from the TPA. Then, he could decrypt the encryption key by using his pre-chosen password to get the master key. Finally, he could decrypt the encrypted file and get the source file safely. In this scheme, the key needed to be encrypted before storing into the TPA which caused more overhead and increased complexity.

A ciphertext Policy Attribute-Based Encryption (CP-ABE) scheme was proposed by K. Han and others in [20]. This scheme depended on generating keys with multiple attributes using multiple distributed parts. It also achieved backward secrecy by updating attributes and re-encrypted ciphertext. In this scheme, Different cryptography schemes were proposed to keep data secure but each with its own limitations and drawbacks. Subarea in [12] presented a scheme which used ECC to encrypt the plaintext and then

communicated with the destination through a secured channel. Simultaneously, MD5 was used to get the hash value from the same plaintext. This value was encrypted using DUAL RSA and also sent to the destination. This scheme achieved integrity by comparing the new hash value with the decrypted one and also achieved difficulty for attackers to extract the text from the encrypted one, but it used two asymmetric encryption algorithms (ECC and DUAL RSA) which exploit extra time for encryption.

Security-Aware Efficient Distributed Storage (SA-EDS) scheme was proposed in [21]. There were three algorithms used in this scheme. The first is called the Alternative Data Distribution (AD2) algorithm which was used to know if the plaintext needs to be divided and stored in distributed cloud servers. The second one is called Secure Efficient Data Distributions (SED2) algorithm which took the plaintext as input and gave two separate ciphertexts as output. The third one is called the Efficient Data Conflation (EDCon) algorithm which was used to enable users to retrieve the text by merging the two separate ciphertexts from distributed cloud servers. It took the two encrypted parts of the text and the key and then gave the plaintext. In this scheme, using three algorithms caused more overhead and more complexity.

Design and implementation of the self-encryption method on file security were proposed in [22]. The scheme was applied to the text before uploading it into the cloud. It divided the plaintext and the ciphertext into 1024 bits chunks using the XOR process. The scheme used the date and starting time of the encryption process as a seed key. It also used a database for storing the ID of the file which contained the plaintext and the key. At the decryption phase, the key was retrieved from the database to get the plaintext. Storing the key into the database was considered a drawback in this scheme because it could be attacked.

3. THE PROPOSED TCS_DD IN CLOUD COMPUTING

In this paper, a two-stage cryptography scheme is proposed to access and store data into cloud storage safely. In the first stage, the scheme archives user authentication using the proposed Two-Factor Authentication Scheme (TFAS) to prove that the accessing user is the real user. The first factor depends on a traditional user name and password. The second factor depends on the one-time password (OTP) technique. The user accesses his cloud account for download or uploads his file using OTP which is successful for only one login.

In the second stage, the file to be stored is split into two parts and encrypted at the user side before sending it into the cloud. The two encrypted parts are stored into two different servers. Files can be divided into more parts and stored into more than two servers, but limiting the division to two reduces the overhead and improves the efficiency. Therefore, the user must create two cloud accounts.

A unique key is used to encrypt each odd character of data, so we need a number of stream keys equal to half the size of the plaintext. To generate the required keys, logistic chaos model theory is used which depends on two important parameters (initial condition x0 and control parameter μ). The user enters different parameters to encrypt each part, so it is very difficult for an attacker to predict these values and decrypt the plaintext. The characteristics of the generated random sequences adhere to all performance characteristics of chaos models as described in [23, 24]. These characteristics can be summarized as, ending the problem of repetition, good randomness and complexity, extreme sensitivity to initial conditions, and low cost with simple iteration.

To retrieve his file, the user first downloads the two parts of the file and decrypts each part separately, and then merges the two decrypted parts forming the original file. This scheme keeps data more secure because each specific user is capable of decrypting his file as he has the sole access to the control parameters used in generating the keys. It also achieves less complexity and reduces the execution time of encryption and decryption processes.

3.1. Stage 1: Authentication using TFAS

In this stage, a two-factor authentication technique is presented. This scheme uses OTP as the second factor of authentication in addition to the traditional user name and password as the first factor. The process consists of four phases; registration phase, login phase, encryption phase, and authentication phase. During the initial user-registration phase, the user selects his user name and password. Subsequently, the user selects a pattern contains four cells in any order from a 3×3 grid. The grid contains 9 cells numbered from C1 to C9. Each of these cells contains a number ranging from 0 to 99. These numbers are randomly generated. For the chosen pattern, the user enters the number which represents each cell and appears on it. Both the numbers in the pattern and the length of it is changed every login for the same user. For example, he can enter the numbers which are appeared on C1, C2, C3, C6 or C1, C2, C5, C4 or C3, C6, C9, C8. The only restriction for the user is that he must recall the order of the chosen cells to use it at the login phase. Upon the completion of a new-user registration, the user name, password and the pattern which are chosen by the user are stored in the database. This phase is illustrated in Figure 2.

At the login phase, a login page appears to the user, in which he must enter his user name and password which he had chosen at the registration phase. Then a grid of 3×3 cells will appear to the user. The numbers shown on the grid cells are randomly generated. Every cell contains a number ranging from 0 to 99. So the numbers can be formed from one digit or two digits. Therefore, the length of the pattern formed from the four cells chosen at the registration phase maybe 4, 5, 6, 7, or 8 digits. Each attempt by the user to log in, the grid displays a random number in each cell. The user must remember the pattern which he had chosen at the registration phase and enters the numbers listed on the cells corresponding to the chosen pattern. These numbers are then encrypted using a special encryption application.

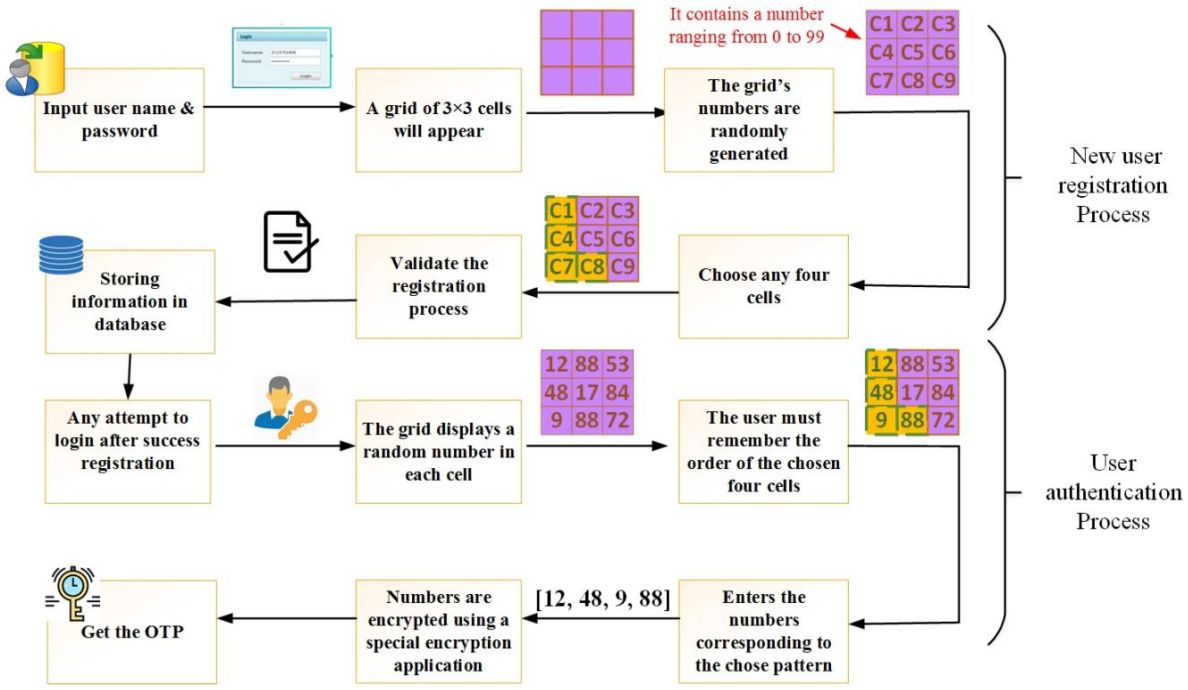

Figure 2. The proposed TFAS scheme

Next, during the encryption phase, a desktop application is used to encrypt the password. When the user runs this application, a window form is displayed. The form asks the user to enter his user name and password, and then the contents of the 4-cell which had been got at the login phase. If all information is entered correctly in the form, the user can click on a button to run the code of the encryption process. The output of the encryption code maybe 4, 5, 6, 7, or 8 digits. This output is used as OTP which is the second factor of authentication in this scheme.

Finally, the authentication phase, the user must enter his username, password, and OTP into the login page. This OTP may be 4, 5, 6, 7, or 8 digits. Its length may be changed every login for the same user. All of this information is sent to the server. On the server-side, the server has the same encryption code which exists at the user side. It is used to encrypt the contents of the cells which represent the user pattern. A comparison is performed between the received OTP and the evaluated one. If the comparison process is successful, the user can access the system. Otherwise, an error message is displayed to the user. The authentication process is illustrated in Figure 2.

The proposed TFAS scheme can resist practical attacks. It overcomes the weaknesses in the existing schemes. It is easy for users, does not have strong constrains, does not depend on receiving a message through the internet, and does not require specific extra hardware or extra time to identify the user.

3.2. Stage 2: Cryptography

In this stage, the plaintext is divided into two parts sent to different cloud servers. Each part is encrypted using a unique key that is generated using logistic chaos model theory. In the decryption, each character is decrypted using its unique key which is generated from the same chaos equation used in the encryption process.

3.2.1. The encryption process

The steps of the encryption process are illustrated in Figure 3. In the encryption process, firstly, the length of the plaintext (n) is determined. The plaintext is divided into two parts; Part A (0: (n/2) -1) and Part B (n/2: n). Only odd-numbered characters in each part will be encrypted by a unique key generated using the chaos model theory while the even-numbered characters remain without encryption in order to minimize the encryption and decryption time. The encryption keys are generated using the formula of the logistic chaos model [25]:

$$k_{n+1} = \mu k_n (1 - k_n) \quad (1)$$

The user chooses a random number for μ and k_0 such that $\mu \in [0,4]$, and $k \in (0,1)$ respectively. Then for each n, the corresponding k_{n+1} is calculated by (1).

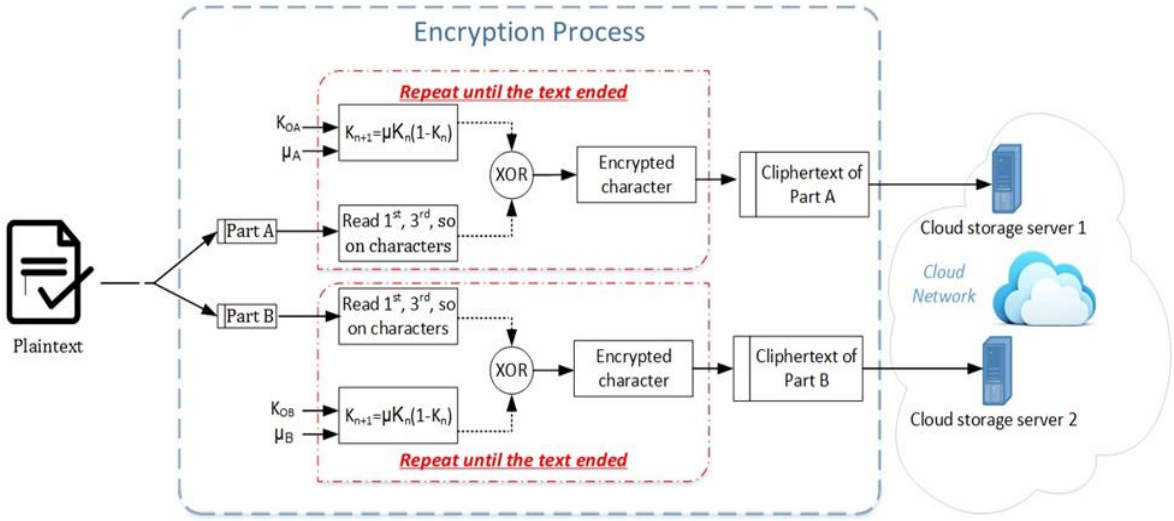

Figure 3. The encryption process of TCS_DD

For Part A, the user enters the initial condition of the chaos formula k_{0A} and the control parameter μ_A. Stream keys are generated using logistic chaos formula. Some symbols must be encrypted in this part; each one is encrypted with its unique key using XOR.

$$C_A = \text{ASCII}(s_i) \text{XOR ASCII}(k_A[l]) \quad (2)$$

Where C_A represents each encrypted character of Part A, s_i (for i=0 to i=(n/2)-1) represents a character in Part A, $k_A[l]$ is the array of keys used to encrypt Part A. The encrypted characters of Part A are appended to the Ciphertext of Part A (E_A) using the function: E_A.appendtext(C_A) in the encryption algorithm shown in Figure 4 while the unencrypted characters are appended using the function: E_A.appendtext(s_i) as shown in the algorithm.

For Part B, the user enters the initial condition of chaos formula k_{0B} and the control parameter μ_B. The same previous steps are done to encrypt some symbols of Part B.

$$C_B = \text{ASCII}(s_j) \text{XOR ASCII}(k_B[l]) \quad (3)$$

Where C_B represents each encrypted character of Part B, s_j (for j=n/2 to j=n) represents each character in Part B, $k_B[l]$ is the array of keys used to encrypt Part B, and E_B is the Ciphertext of Part B which is appended using the function: E_B.appendtext (C_B) in the algorithm. The other symbols of Part B are appended using the function: E_B.appendtext (s_j) as illustrated in the algorithm which is shown in Figure 4. When the encryption process is complete, the user can send E_A to a cloud server, while E_B is sent to another cloud server to ensure data security. Figure 4 shows the pseudo-code of the encryption process.

Pseudo-code of encryption process		
Input : F (plaintext) Output : E_A (ciphertext of Part A), E_B(ciphertext of Part B)		
1.	n = $F.length$	//Get length of plaintext
2.	Let i =0; Let j =n/2; Let l =0;	//initial values
3.	Declare K_A[ ] and K_B[ ]	//Arrays K_A and K_B save keys used to encrypt Part A and B respectively
4.	Input k_A[0]	//Input initial condition of chaos formula (random value from 0 to 1) for Part A, k_{0A}
5.	Input μ_A	//Input control parameter (random value from 1 to 4) for Part A
6.	Input k_B[0]	//Input initial condition of chaos formula (random value from 0 to 1) for Part B, k_{0B}
7.	Input μ_B	//Input control parameter (random value from 1 to 4) for Part B
8.	Do {	
9.	$P_A = \sum_{i=0}^{i=\frac{n}{2}-1} s_i$	//Part A, s_i represent symbol of Part A
10.	C_A = ASCII(s_i) XOR ASCII ($k_A[l]$)	//Encrypt each symbol with its unique key
11.	E_A.appendtext (C_A)	//Append each encrypted char to E_A
12.	i++	//increment i
13.	E_A.appendtext (s_i)	//append char s_i to E_A
14.	$K_A[l+1] = \mu_A\ k_A[l](1- k_A[l])$	//Generate a key using chaos formula
15.	i++	//increment i
16.	$P_B = \sum_{j=n/2}^{j=n} s_j$	//Part B, s_j represent the symbol of Part B
17.	C_B = ASCII (s_j) XOR ASCII ($k_B[l]$)	//Encrypt each symbol with its unique key
18.	E_B.appendtext (C_B)	//append each encrypted char to E_B
19.	j++	//increment j
20.	E_B.appendtext (s_j)	//append char s_j to E_B
21.	$K_B[l+1] = \mu_B\ k_B[l](1- k_B[l])$	//Generate a key using chaos formula
22.	j++	//increment j
23.	l++}	//increment l
24.	While(i< n/2 && j < n)	//Repeat this loop until i=n/2 and j= n

Figure 4. The pseudo-code of the encryption process

3.2.2. The decryption process

Figure 5 illustrates the steps of the decryption process. The user can download his data at any time. Each data part is retrieved from the corresponding cloud server saved on and decrypted separately. As explained before in the encryption process, the odd characters of the Ciphertext of Part A (E_A) are decrypted using the following equation after entering the initial condition of chaos formula k_{0A} and the control parameter μ_A which were used at the encryption process for this part.

$$D_A = \text{ASCII}(s_i) \text{XOR ASCII}(k_A[l]) \quad (4)$$

Where D_A represents each decrypted character of part A, s_i represents each character in E_A, K_A [l] is the array of TCS_DD keys used to decrypt E_A. Each decrypted character is appended to the plaintext (F) using the function: F.appendtext (D_A) in the algorithm. The other symbols of this part are appended using the function: F.appendtext (s_i) as shown in the algorithm in Figure 6.

For the second part of the data, the exact same steps are repeated. First, the user enters the initial condition of chaos formula k_{0B} and the control parameter μ_B which were used at the encryption process for this part. Next, the TCS_DD stream keys are generated using logistic chaos formula.

$$D_B = \text{ASCII}(s_j) \text{XOR ASCII}(k_B[l]) \quad (5)$$

Where DB represents each decrypted character of Part B, s_j represents each character in EB, KB [l] is the array of keys used to decrypt EB. The function: F.appendtext (DB) in the algorithm is used to append the decrypted characters to Plaintext (F), while the function: F.appendtext (s_j) is used to append other symbols of this part as shown in the algorithm presented in Figure 6.

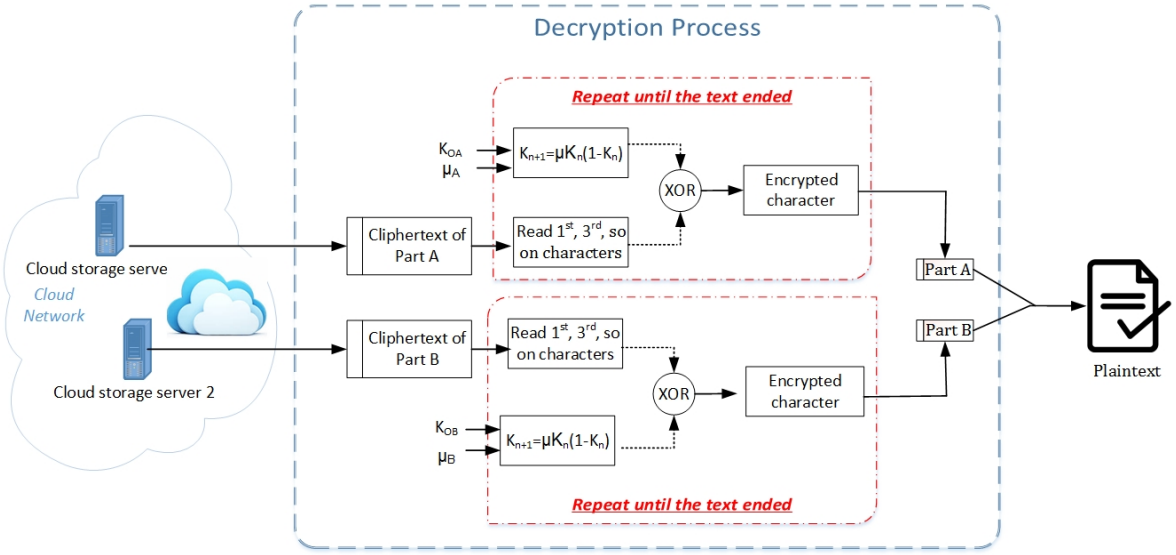

Figure 5. The decryption process of TCS_DD

Pseudo-code of decryption process

input: E_A (ciphertext of Part A), E_B (ciphertext of Part B)
output: F (plaintext)

1.	$n_A = E_A.length$	// Get the length of the first decrypted part
2.	Declare k_A [ ] : array of double	//Array of K_A save keys which are used to decrypt the ciphertext of Part A
3.	Input $k_A[0]$	//Input initial condition of chaos formula k_{0A} (used at the encryption process of the first part)
4.	Input μ_A	//Input control parameter (used at the encryption process of the first part)
5.	Let i =0	//Initialize i with zero
6.	$n_B=E_B.length$	//get the length of the second decrypted part
7.	Declare k_B[ ] : array of double	// Array of K_B save keys which are used to decrypt the ciphertext of Part B
8.	Input $k_B[0]$	//Input initial condition of chaos formula k_{0B} (used at the encryption process of Part A)
9.	Input μ_B	//Input control parameter (used at the encryption process of Part B)
10.	Let j = 0;	//Initialize j with zero
11.	Let l = 0;	//Initialize l with zero (index of array)
12.	Do{	
13.	$E_A = \sum_{i=0}^{i=n_A} s_i$	// ciphertext of Part A
14.	D_A = ASCII (s_i) XOR ASCII ($k_A[l]$)	//decrypt each symbol with different key
15.	F.appendtext (D_A)	//append each decrypted char to F
16.	i++	// Increment i
17.	F.appendtext (s_i)	//append char s_i to F
18.	$K_A[l+1] = \mu_A\ k_A[l](1- k_A[l])$	//Generate a key using chaos formula
19.	i++	//Increment i
20.	$E_B = \sum_{j=0}^{j=n_B} s_j$	//ciphertext of Part B
21.	D_B = ASCII (s_j) XOR ASCII ($k_B[l]$)	//decrypt each symbol with different key
22.	F.appendtext (D_B)	//append each decrypted char to F
23.	j++	// increment j
24.	F.appendtext (s_j)	//append char s_j to F
25.	$k_B[l+1] = \mu_B\ k_B[l](1- k_B[l])$	//Generate a key using chaos formula
26.	j++	// increment j
27.	l++}	//increment l
28.	While(i<n_A&& j <n_B)	//Repeat this loop until i=n_A and j = n_B

Figure 6. The pseudo-code of the decryption process

3.2.3. Illustration of the proposed TCS_DD algorithm

To further explain the proposed TCS_DD algorithm, a detailed example run on a small plaintext of size 44 characters. The plaintext is assumed to be:

Plaintext: https://www.sciencedirect.com/science/articl

After running the proposed encryption algorithm, the resulting ciphertext is:

Ciphertext: "Ittpr:./vwv.rcheocddhrdcu.com/rchencd/`ruibl"

That can be decrypted using the decryption algorithm to form the same plaintext. The results are listed in Table 1 and Table 2. The detailed steps are as follows: The file is first divided into two parts; each one contains half the number of characters in the file which is 22 characters. Only half of the characters in each part will be encrypted and the other half will remain without encryption. The purpose of that is to reduce both encryption and deception time. For the first part, assuming the random initial condition k_{0A} of (1) can be 0.73, and the control parameter μ_A can be 2.37. For the second part, there are 11 characters must be encrypted. The random initial condition k_{0B} of (1) can be assumed to be 0.512, and control parameter μ_B also can be assumed to be 3.95. From Table 1 and Table 2, it is shown how randomly keys are generated using the logistic chaos model. These keys are used to make encryption and decryption processes more efficient.

Table 1. Detailed cryptography process for part A

Original character	Encryption key	Encrypted character	Decryption key	Output character
h	0.73	I	0.73	h
t	0.467127	T	0.467127	t
s	0.58993889711427	R	0.58993889711427	s
/	0.573329057642509	.	0.573329057642509	/
w	0.57975615285347	V	0.57975615285347	w
w	0.577424325914373	V	0.577424325914373	w
s	0.578292972803391	R	0.578292972803391	s
i	0.57797239867077	H	0.57797239867077	i
n	0.578091122957898	O	0.578091122957898	n
e	0.578047210340964	D	0.578047210340964	e
i	0.578063460110444	H	0.578063460110444	i

Table 2. Detailed cryptography process for part B

Original character	Encryption key	Encrypted character	Decryption key	Output character
e	0.512	D	0.512	e
t	0.9869312	U	0.9869312	t
c	0.0509471255429115	C	0.0509471255429115	c
m	0.190988487970214	M	0.190988487970214	m
s	0.610321947460609	R	0.610321947460609	s
i	0.93942481823857	H	0.93942481823857	i
n	0.2247780250082	N	0.2247780250082	n
e	0.68829881470237	D	0.68829881470237	e
a	0.847447047707646	`	0.847447047707646	a
t	0.510658168705002	U	0.510658168705002	t
c	0.98705129358743	B	0.98705129358743	c

4. SIMULATION RESULTS

To evaluate the performance of the proposed TCS_DD scheme and its effectiveness, three performance measures were calculated: the size of the ciphertext, encryption and decryption times, and time complexity.

- The size of the ciphertext is the size of the encrypted plain test file in bytes.
- Encryption time is calculated as the time required for encryption i.e. the time taken to convert the plaintext into ciphertext. Decryption Time is the time required for converting the ciphertext back into the plaintext.
- Time complexity is the computational complexity that describes the amount of time it takes to run an algorithm.

These performance measures were compared to other recent competing encryption approaches which use a combination of symmetric and/or asymmetric algorithms. These algorithms are: Subasree algorithm [9], Kumar algorithm [10], Zhu algorithm [11], THCA algorithm [12], and SCBRP algorithm [13]. The proposed algorithm was implemented using C# programming language and the simulations were run on an Intel i5-3317U 1.70 GHz CPU with 4.00 GB of RAM using 64-bit implementations to ensure maximum utilization of the hardware.

4.1. Size of ciphertext

TCS_DD scheme along with Subasree, Kumar, Zhu, THCA, and SCBRP were used to encrypt several test files with different sizes ranging from 609 bytes to 184,162 bytes. The resulted ciphered file (ciphertext) size was measured in bytes and listed in Table 3. From Table 3, each of Subasraa, Zhu, SCBRP, and TCS_DD produces a ciphertext size which is equal to the size of the plaintext. While both THCA and Kumar have a bigger ciphertext size. This is due to the fact that in the proposed algorithm each odd character of plain text is encrypted using a unique key generating one encrypted character.

Table 3. Size of ciphertext (bytes)

Size of Plaintext (bytes)	Subasree	Kumar	Zhu	THCA	SCBRP	TCS_DD
609	609	846	609	641	609	609
25,615	25,615	35,142	25,615	25,647	25,615	25,615
35,080	35,080	48,226	35,080	35,112	35,080	35,080
61,386	61,386	84,340	61,386	61,418	61,386	61,386
184,162	184,162	353,008	184,162	184,194	184,162	184,162

4.2. Encryption and decryption time

Tables 4 and 5 show the encryption and decryption times for different sizes of plaintext ranging from 609 bytes to 184,162 bytes using TCS_DD, Subasree, Kumar, Zhu, THCA, and SCBRP. It is clear that the proposed TCS_DD yields the least processing time for both encryption and decryption processes. This is achieved because the proposed scheme splits the file into two parts which can be encrypted simultaneously. However, when the size of file reaches 184,162 bytes, SCBRP consumes less time. This is due that SCBRP uses two symmetric algorithms which are very simple to implement however they are very weak against hackers. Figure 7 and Figure 8 show the difference of encryption and decryption times between algorithms.

Table 4. Encryption time (ms)

Size of Plaintext (byte)	Subasree	Kumar	Zhu	THCA	SCBRP	TCS_DD
609	2063	1500	998	998	650	45
25,615	3683	1518	1022	1022	725	384
35,080	5651	1526	1059	1059	743	526
61,386	15351	4219	3143	3143	2150	920
184,162	105889	5752	3814	3814	2256	2762

Table 5. Decryption time (ms)

Size of Plaintext (byte)	Subasree	Kumar	Zhu	THCA	SCBRP	TCS_DD
609	1078	966	562	562	356	40
25,615	1085	972	713	713	452	320
35,080	1082	980	824	824	483	438
61,386	1197	991	891	891	525	520
184,162	2087	1099	907	907	547	767

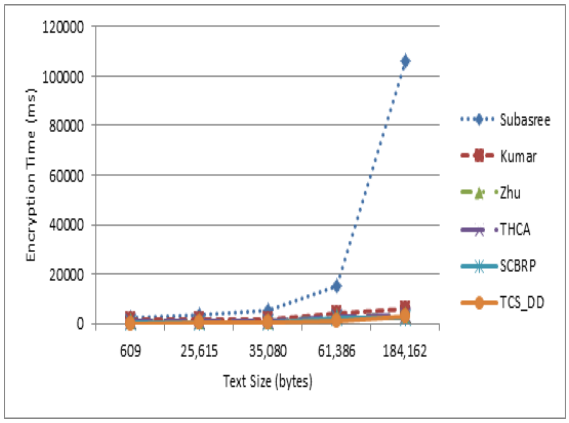

Figure 7. Comparison of encryption time with competing algorithms

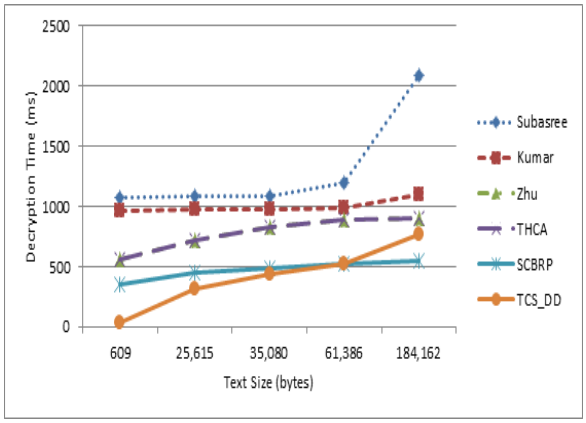

Figure 8. Comparison of decryption time with competing algorithms

4.3. Algorithm time complexity

The time complexity of an algorithm is the amount of time which is needed to run to completion. Table 6 shows the time complexity of the proposed TCS_DD compared to the other related algorithms for the encryption and decryption process. From this comparison, it is clear that the time complexity of the proposed TCS_DD scheme, as well as the other algorithms is *O(n)* where *n* is the size of the plaintext. This means that the proposed algorithm enhances the overall performance without adding any complexity.

Table 6. Time complexity of encryption and decryption processes

Algorithm	Encryption process	Decryption process
Subasree	$O(\log(n^2)+4n) \cong O(n)$	$O(\log(2n^3)+4n) \cong O(n)$
Kumar	$O(\log_2(n+1)+\sqrt{n}+4n) \cong O(n)$	$O(\log_2(n+1)+\sqrt{n}+5n) \cong O(n)$
Zhu	$O(\log_2(2n+1)+\sqrt{n}+4n) \cong O(n)$	$O(\log_2(2n+1)+\sqrt{n}+4n) \cong O(n)$
THCA	$O(\log(n^2)+\log(n)+3n) \cong O(n)$	$O(\log(n)+\log(2n^3)+2n) \cong O(n)$
SCBRP	$O(2n+1) \cong O(n)$	$O(2n+1) \cong O(n)$
TCS_DD	$O(n/2) \cong O(n)$	$O(n/2) \cong O(n)$

5. CONCLUSION

In this paper, an efficient two-stage encryption scheme is proposed. The scheme enables the users to encrypt files before safely storing them into the cloud. First, a two-factor authentication scheme is proposed to authenticate users in the cloud. Second, the file is split into two parts each one is saved in a different cloud server to achieve distributed storage. Each character is encrypted by a unique key generated using logistic chaos model theory which uses initial parameters entered by the user. Therefore, it is very difficult for an attacker to know these values and decrypt the plaintext. Simulation results show that this scheme presents better security and achieves lower processing overhead. It also achieves less encryption and decryption time comparing with related cryptography algorithms because of splitting the original file into two parts which can be encrypted simultaneously.

REFERENCES

[1] B. Carpentieri, "Efficient Compression and Encryption for Digital Data Transmission," *Security and Communication Networks*, vol. 2018, ID. 9591768, pp. 1-9, 2018.

[2] A. Bhardwaj, *et al*., "Security algorithm for cloud computing," *Proc. of Comp. Sci.*, vol. 85, pp. 535-542, 2016.

[3] J. Xiong, Y. Zhang, S. Tang, X. Liu, and Z. Yao, "Secure encrypted data with authorized deduplication in cloud," *IEEE Access*, vol. 7, pp. 75090–75104, 2019.

[4] I. M. Mahmoud, S.H. Nour El-Din, R. Elgohary, and H. Faheem, "A robust cryptographic-based system for secure data sharing in cloud environments," *Security and Communication Networks*, vol. 9, no. 18, pp. 6248-6265, 2016.

[5] M**.** Gamal, R. Rizk, H**.** Mahdi, and B**.** E. Elnaghi, "Osmotic bio-inspired load balancing algorithm in cloud computing," *IEEE Access*, vol. 7, no. 1, pp. 42735-42744, 2019.

[6] H. Nashaat, N. Ashry, and R. Rizk, "Smart elastic scheduling algorithm for virtual machine migration in cloud computing," *Journal of Supercomputing*, Springer, vol. 75, no. 7, pp. 3842-3865, 2019.

[7] Y. Alkady, F. Farouk, and R. Rizk, "Fully homomorphic encryption with AES in cloud computing security," *Proc. of the International Conference on Advanced Intelligent Systems and Informatics*, vol. 845, pp. 370-382, 2018.

[8] S. Ganapathy, "A secured storage and privacy-preserving model using CRT for providing security on cloud and IoT-based applications," *Comput. Netw*., vol. 151, pp. 181–190, 2019.

[9] Y. Alkady, M. I. Habib, and R. Rizk, "A new security protocol using hybrid cryptography algorithms," *Proc. of 9th International Computer Engineering Conference (ICENCO)*, Egypt, pp. 109-115, 2013.

[10] D. Yuan, X. Song, Q. Xu, M. Zhao, X. Wei, H. Wang, and H. Jiang, "An ORAM-based privacy preserving data sharing scheme for cloud storage," *J. Inf. Secur. Appl*., vol. 39, no. C, pp. 1-9, 2018.

[11] S. H. El-sherif, *et al*., "Two-factor authentication scheme using one time password in cloud computing," *Proc. of the International Conference on Advanced Intelligent Systems and Informatics*, Egypt, vol. 845, pp. 425-434, 2018.

[12] Subasree and N. K. Sakthivel, "Design of a new security protocol using hybrid cryptography algorithms," *International Journal of Research and Reviews in Applied Science*, vol. 2, no. 2, pp. 95-103, 2010.

[13] S. K. Namini, "A secure communication wireless sensor networks through hybrid (AES+ECC) algorithm," LAP Lambert Academic Publishing, 2012.

[14] S. Zhu, "Research of hybrid cipher algorithm application to hydraulic information transmission," *International Conference on Electronics, Communications and Control (ICECC)*, Ningbo, China, 2011.

[15] R. Rizk, and Y. Alkady, "Two-phase hybrid cryptography algorithm for wireless sensor networks," *Journal of Electrical Systems and Information Technology*, vol. 2, no. 3, pp. 296-313, 2015.

[16] M. Pavani, and P. T. Rao, "Adaptive PSO with optimised firefly algorithms for secure cluster-based routing in wireless sensor networks," *IET Wireless Sensor Systems*, vol. 9, no. 5, pp. 274-283, 2019.

[17] A. Bansal and A. Agrawal, "Providing security, integrity and authentication using ECC algorithm in cloud storage," *International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, pp. 1-5, 2017.

[18] D. P. Timothy, and A. K. Santra, "A hybrid cryptography algorithm for cloud computing security," *International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, Vellore, India, pp. 1-5, 2017.
[19] J. Chueh, and M. Sun, "Design and implementation of security system for cloud storage," *19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Seoul, South Korea, pp. 129-134, 2017.
[20] K. Han, Q. Li, and Z. Deng, "Security and efficiency data sharing scheme for cloud storage," *Journal of Chaos, Solution and Fractals*, vol. 86, pp. 107-116, 2016.
[21] Y. Li, K. Gai, L. Qiu, M. Qiu, and H. Zhao, "Intelligent cryptography approach for secure distributed big data storage in cloud computing," *Journal of Information Science*, vol. 9, pp. 1-13, 2016.
[22] M.R. Rahardjo and G.F. Shidik, "Design and implementation of the self-encryption method on file security," *International Seminar on Application for Technology of Information and Communication (iSemantic)*, Semarang, Indonesia, pp. 181-186, 2017.
[23] R. Ye and W. Guo, "A chaos-based image encryption scheme using multimodal skew tent maps," *CIS*, vol. 4, no. 10, pp. 800-810, 2013.
[24] M. Aledhari, A. Marhoon, A. Hamad, and F. Saeed, "A new cryptography algorithm to protect cloud-based healthcare services," *IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Philadelphia, PA, USA, pp. 37-43, 2017.
[25] B. En-Jian, Z. Jun-Jie, and W. Liang-Cheng, "WSN message authentication code based on chaos and XOR-encryption, IFSA," S*ensors and Transducers*, vol. 156, no. 9, pp. 161-167, 2013.

BIOGRAPHIES OF AUTHORS

Rabab Farouk Abdel-Kader received her B.Sc. from the Electrical Engineering Department Suez Canal University in 1998. She received her Ph.D. degree from the department of Computer Science and Software Engineering at Auburn University, Auburn, AL in 2007 and the MS degree in Electrical Engineering from Tuskegee University with high honors in 2002. Since 2008 she is working as an Assistant Professor in the Electrical Engineering department, Faculty of Engineering, Port-Said University, Egypt. Her main research interests include image processing, parallel computing, network security, and software Engineering.

Samar H. El-Sherif received her B.Sc. from Electrical Engineering department, Computer and Control Section, Faculty of Engineering, Port-Said University in 2010 and her Master of Science in Electrical Engineering, Computer and Control Section, Faculty of Engineering, Port-Said University in (2019). Now she is working as an Assistant Lecturer, Technology and Information System Department, Faculty of Management Technology and Information System, Port-Said University.

Rawya Y. Rizk is a Professor of Computers and Control in Electrical Engineering Department, Port Said University, Egypt. She is the Head of Electrical Engineering Department, Port Said University, 2017 till now. She is the Chief Information Officer (CIO) of Port Said University (PSU), 2014 till now. She received her BSc, MSc and PhD in Computers and Control Engineering from Suez Canal University in 1991, 1996 and 2001, respectively. Her research interest is in computer networking, including mobile networking, wireless, ATM, Sensor Networks, Ad Hoc Networks, QoS, traffic and congestion control, handoffs and cloud computing. She is a reviewer in many of international communication and computer journals such as IEEE Access, IET communications, IET sensors, IET Networks, Journal of Supercomputing, Journal of Network and Computer Applications, Computers & Electrical Engineering, Mathematical Problems in Engineering, and IJACSA.