Received: October 18, 2020. Revised: November 11, 2020.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

The Dynamic Symmetric Four-Key-Generators System for Securing Data Transmission in the Industrial Control System

Eko Hadiyono Riyadi[1,2] **Tri Kuntoro Priyambodo**[1]* **Agfianto Eko Putra**[1]

[1]*Computer Science and Electronics Department, Universitas Gadjah Mada, Yogyakarta, Indonesia*
[2]*Assessment Centre for Nuclear Installation and Material, Nuclear Energy Regulatory Agency, Jakarta, Indonesia*
* Corresponding author's Email: mastri@ugm.ac.id

Abstract: Most of the communication protocols in the Industrial Control System (ICS) are vulnerable to cyber-attacks. Initially, the network protocol was designed for reliable performance, and thus did not incorporate data transmission security features. Therefore, ICS requires adequate data transmission security. This paper suggests improving the security of data transmission through a dynamic symmetric four-key-generators system, wherein the system anticipates cyberattacks by generating four keys before encryption. It involves four generators: a random initial key generator, a keystream generator, a key scheduling algorithm generator, and a pseudo-random number algorithm generator. In the receiver section, the system generates three keys before decryption to ensure data confidentiality and to avoid cyber-attacks. The test results show that the proposed system keyspace is ≈22048 bits, meaning that the key is more secure from brute force attacks. As a result, the cipher data have a correlation value of 0.00007. The entropy value is 7.99, indicating that the cipher data is more secure. Also, speed tests show that the processing time still qualifies as real-time.

Keywords: Super encryption, ICS security, Protocol security.

1. Introduction

The industry needs a control system to implement an automation process. This system, the collection of individual control systems and other hardware that automates or operates industrial processes, is identified as the Industrial Control System (ICS refers to [1]).

Some industries carry out production activities in several processes at the same time, but in different areas. Such a process requires a communication protocol, with several commonly included ones being Modbus, Profibus, DNP3, and OPC. Initially, the network protocol was designed for reliable communication and real-time processing and used on relatively secure local networks. As such, it does not incorporate adequate security features [2].

As communication and network technologies have advanced, TCP/IP network protocols began to be developed to communicate between ICS devices via the internet. However, this has brought a new problem: the risk of cyberattacks [3]. Experiences in the last decade show that cyberattacks are becoming increasingly common, and the damage and losses they cause are increasing.

Several critical industries have been the main targets of these cyberattacks, including power generation, telecommunication, oil and gas, and chemical. All of these industries have reported cyber-attacks. Many have been due to protocol vulnerabilities, Modbus and DNP3 vulnerabilities, cryptographic attacks, replay attacks, communication stack attacks, and flooding. Industrial systems have also been vulnerable to such internet-facing threats as man-in-the-middle (MITM) attacks, eavesdropping, false command and control communications, and TCP/lP stack exploits. Others have been vulnerable to malware attacks, i.e. Stuxnet worm-altered Programmable Logic Controller (PLC) operations, or the injection of false data. Still others have been attacked through automatic payload generation, wherein the form of PLCs are exploited [4, 5].

All such attacks cause damage and losses, ranging from the minor damage to the severe. Cyber-attacks can paralyze a country, as they result in the cessation of community services (as seen in the 2007 DDOS attack in Estonia [6]). Such attacks can also occur due to human factors, both internal and external [7].

Unencrypted data transmission is thus vulnerable to cyberattacks, and it is necessary to secure data transmissions in ICS. This issue can be overcome by encrypting data transmissions with complex key generators. In cryptography, Kilinc [8] showed communication protocols' intrinsic shortcomings in controlling network processes. They introduced a distributed key generation model for ICS as a representation of Petri nets. Bernardinello [9] subsequently introduced a Petri nets protocol model for id-based private key generation by compiling two network models: one modelling the interaction between the private key generator nodes, another modelling the client from the main generator. Rajkumar [10] introduced a lightweight technique to ensure the confidentiality and integrity of the messages. Meanwhile, Ouaissa [11] presented an efficient and secure authentication and key agreement protocol for the IoT system.

There are, however, several issues with encryption key management: first, difficulty generating symmetric keys that cannot easily be predicted by cryptanalysts; second, maintaining the confidentiality of symmetrical keys; third, distributing symmetrical keys [12]. With these issues, insecurity in data transmission remains.

This study introduces a new method of improving data transmission security using a four-key-generator in BRC4 super encryption, i.e., a random initial key generator (K1), a keystream generator (K2), a key scheduling algorithm generator (K3), and a pseudo-random number algorithm generator (K4). This four-key-generator aims to increase the data transmission security on ICS.

For security purposes, the system inserts K1 as a means of avoiding cyber-attacks. Even if an attacker succeeds in getting a ciphertext, this ciphertext cannot be read. Even if an attacker can separate K1 from the ciphertext, the ciphertext data cannot be read because the decryption process requires K2, K3, and K4 (whereas the attacker only has K1).

2. Related works

Some ICS observers have continued to develop security features in order to anticipate cyberattacks and improve communication between ICS devices, especially to data transmission security.

Mohamed [13] stated that communication devices must provide a strong communication capacity to secure data transmission of various data types to prevent cyber-attacks, including implementing cryptographic methods. As data transmission security, each cryptographic scheme is built with its strengths. However, the application of a single cryptographic technique to the system has several weaknesses. For example, the symmetric encryption method is cost-effective to secure data without compromising security. Unfortunately, how to share the secret key is a vital issue.

Meanwhile, the asymmetric scheme solves the secret key distribution problem. However, the processing speed is slower and consumes more computer resources compared to symmetric encryption. As an alternative to overcome each scheme's security weaknesses, the integration of several cryptographic methods is being proposed to offer efficient data security and solve key distribution problems.

Several previous studies utilize symmetric cryptography. For example, the Advanced Encryption Standard (AES) was conducted by Altigani [14], Xin [15], and Harba [16]. The Data Encryption Standard (DES) is implemented by Z. Hong [17], who combines with Rivest Code 4 (RC4). Then, Singh [18] uses symmetric encipherment. Meanwhile, some studies that apply asymmetric cryptography, i.e., Rivest Shamir Adleman (RSA) is utilized by Purevjav [19] and Harba [16]. Furthermore, Elliptic Curve Cryptography (ECC) was being used by N. Hong [20] and Xin [15].

N. Hong [20] presents a data transmission security framework based on the ECC cipher algorithm and SM2 handshake agreement to solve security problems between the client and the receptor in the information transmission process. However, the study did not provide a performance evaluation. Altigani [14] proposes a new approach that provides an additional layer of protection for messages transmitted over communication networks. This approach combines the symmetric encryption algorithm (AES) and the Word Shift Coding Protocol steganography protocol. The resulting model has a better impact on the confidentiality of messages sent and the overall system computing security. Xin [15] proposed mixed encryption using AES and ECC. MD5 is integrated with ECC and AES to form a hybrid approach. Unfortunately, the study did not evaluate the performance results.

Singh [18] proposes two different encryption techniques. The first proposed technique focuses on compressing the data by half. The second technique justifies Shallon's idea of diffusion by generating

Received: October 18, 2020. Revised: November 11, 2020.

different ciphertext characters for their distinct appearance in plaintext. The combinatorial effect results in a Hybrid Encryption scheme that makes it difficult for adversaries to learn any information from messages sent over an insecure transmission medium. Purevjav [19] presents a new protocol design for securing email communication on Android OS using a hybrid cryptosystem. It is a combination of a public key encryption system and a symmetrical hash function. The new protocol design uses the RSA asymmetric cipher with the MD5 hash function. Messages encrypted with the public key can only be decrypted for a reasonable time using the private key. Z. Hong [17] proposed a new concept of the fusion encryption algorithm for monitoring equipment. A hybrid security encryption algorithm encrypts the communication data based on DES, and the RC4 fusion encryption algorithm. This study, however, does not present a performance evaluation. Harba [16] proposes a method to protect data transfer with a hybrid encryption technique. A symmetric AES algorithm is used to encrypt files; an asymmetric RSA is used to encrypt AES passwords; and an HMAC is used to encrypt symmetrical passwords and data to ensure secure transmission. The ciphertext size and encryption time results show that the overall encryption results in low computational requirements and high security. D'souza [21] proposes the AES algorithm with a hybrid approach to Dynamic Key Generation and Dynamic S-box Generation. This method adds more complexity in the data to increase confusion and diffusion in the ciphertext using Dynamic Key Generation.

3. Proposed method

To anticipate the vulnerability of data transmissions in the ICS, we introduce a dynamic symmetrical four-key cryptographic method. It involves four generators: a random 256-byte key generator, a key-stream generator, a key scheduling algorithm generator, and a pseudo-random number algorithm generator.

Dynamic symmetric key generation is the process of generating keys through symmetric data encryption; in other words, encryption uses the same key as decryption. Such a process can be completed quickly, with processing time being negligible compared to previous cycles.

We simulate our proposed method using Matlab software, with an instruction list (IL) from the PLC of an industrial machine as simulation data. This data consisted of 4,571 lines of sequentially executed logical commands representing input and output. Each line contained approximately 15 characters. The IL record contained a total of 33,046 characters. The four-key-generators model can be explained as follows:

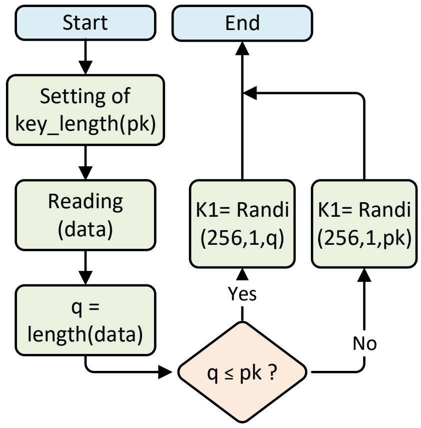


Figure. 1 *K1* generation

3.1. Random initial key generation

This generator produces a random initial key by providing key-length setting options according to data transmission requirements. The longer the key, the more secure it is from brute-force attacks.

Fig. 1 explains the *K1* generation process. The system starts with *pk* key-length setting, then reads the *IL* data while calculating the character length of *the IL* data and storing it in the *q* variable. The system compares the character size of the *IL* data and *pk* key-length. If $q \leq pk$, the system generates a random *K1* as long as the *q* variable. When $q > pk$, the system generates a random key as long as the key *pk*.
This process generates a random *K1* with a range of values between 1 and 256.

3.2. Key-stream generation

This section explains the process of key-stream generation. This generator aims to form keys with certain equations and with the same length as the *IL*-data. Fig. 2 explains the *K2* generation process. It begins with the computation of the character length of the IL data, stored in variable q. It then calculates character length of *K1*, stored in variable r.

The system then compares the values of variables q and r. If $q > r$, it adds the 1 to the variable r and stores it to variable *s*. If $q \leq r$, it stores the value of *r* in variable *s*, then keeps data *K1* in *K2*.

The system compares the values of *q* and *s*. If $q > s$, the length of the IL data is more than the length of the key, and thus the system generates a keystream. Keystream generation follows the equation:

$$k_s = (k_{s-r} + k_{s-1}) \; mod \; 256 \qquad (1)$$

Received: October 18, 2020. Revised: November 11, 2020.

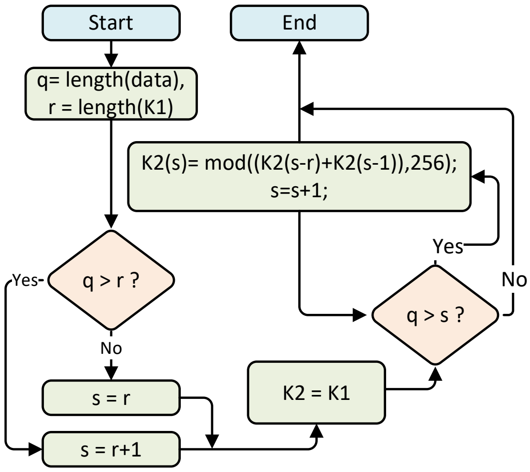


Figure. 2 *K2* generation

Where k_s is the s^{th} key, k_{s-r} is the s^{th} key minus the r^{th} key, and k_{s-1} is the s^{th} key minus *1*. If $q \leq s$, the character length of the IL-data is less than or equal to the key-length, and thus the process is complete; *K2* is equal to *K1*.

3.3. Key scheduling algorithms generation

This section explains the generation of *K3*, i.e. the key scheduling algorithm. This generator aims to generate a random initial array for use in the next key generation process.

The third generator builds an *S*-array and a *K*-array. It initializes permutations in the *S*-array. The *S*-array is then processed for up to 256 iterations. Key length is defined as the number of bytes in the key, and ranges from 1 to 256.

Fig. 3 details the *K3* generation process. It begins by calculating the character-length of the *IL*-data, stored in variable *q*, and calculating the key length of *K2*, stored in variable *r*.

The system checks the value of *q*. If $q<256$, the system adds *1* to the *q* value and stores it in variable *t*. If $q \geq 256$, the system stores the value of *q* in variable *t*, then checks the value of *t*; if $t<256$, the system generates a *KG* keystream of character-length equal to the *IL*-data and key. If $t \geq 256$, it proceeds to the formation of *S* and *K*-arrays.

The system builds an *S*- and a *K*-array up to *256* bytes long. It then generates a key scheduling algorithm using permutations from the sum of the *j*, *Si*, and *Ki* values in Modulo 256. This permutation produces a *K3*-key and an *S* array that is randomized by *256* bytes.

3.4. Pseudo-random algorithm generation

This section explains the generation of pseudo-random algorithms to produce *K4*. After getting a randomized *S*-array from the previous approach, the

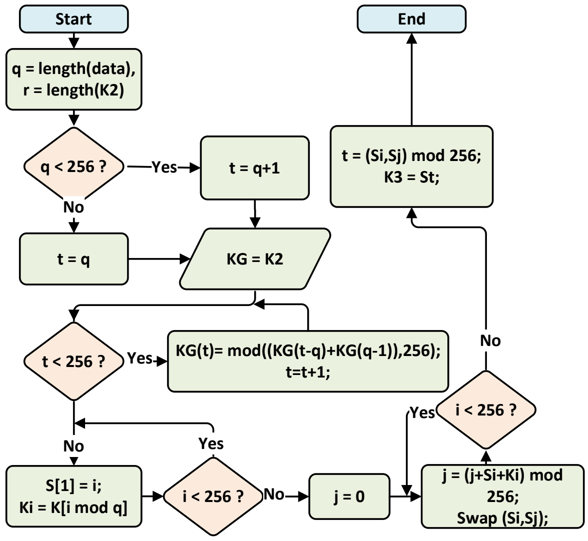


Figure. 3 *K3* generation

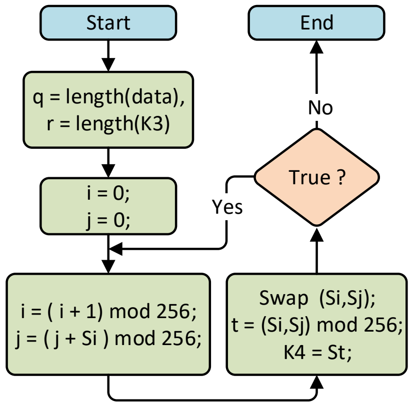


Figure. 4 *K4* generation

system continues to re-initialize the values of *i* and *j* to zero. It then adds *1* to the value of *i* in Modulo 256. Furthermore, it adds the value of *j* to *Si* in Modulo *256*, then adds the value of *S[i]* and *S[j]* and swaps both. *S* values with the same index as the *S[i]* and *S[j]* value is modulated to *256*, producing *K4* (see Fig. 4).

4. Experiment

This section explains the simulation process using *IL*-data from a PLC program. In many lines, *IL*-data is converted into *one*-line by providing a sign (;) to separate lines.

Next, the *IL*-data is converted from string to numeric format. This is aimed to facilitate arithmetic operations in the encryption and decryption processes. After decryption, the system reconverts numeric data into string data. The simulation process for the four-key-generator model is detailed below:

4.1 *K1* generation

This subsection explains the random key-generation process. A simulation was conducted using Matlab software, with the following hardware specifications: i7-6500U processor, 16GB RAM, Windows-10 64-bit operating system.

For *K1* generation, we set a key-length of *256* bytes. The system thus generated a *K1* with a length of 256 characters. Some of the results are shown in Fig. 5.

4.2 *K2* generation

K2 generation begins with a reading of the plaintext *IL*-data. As this data has a length of 33,046 characters, while *K1* has a length of 256 characters, the system generates *K2* for characters 257–33,046 (*K-257th* through *K-33,046th*) using the keystream generator in Eq. (1) below. For example,

K-257th = (K(257-256) + K(257-1)) mod 256
= (K1 + K256) mod 256
= (140 + 212) mod 256
= 352 mod 256
= 96

K-258th = (K2 + K257) mod 256
= (185 + 96) mod 256
= 281 mod 256
= 25

This continues until K-33,046th. Fig. 6 shows the partial results of K2 generation, starting with K-257th.

4.3 *K3* Generation

K3 is generated through key scheduling algorithm. Here, the system forms *S* and *K*-arrays for initial array initiation. Key length is defined as the number of bytes in the key, ranging from *1* to *256*.

Command Window

K1 =

140	185	134	255	56	28	29	17	104	115
94	196	161	198	239	250	50	36	179	25
135	136	221	125	101	172	190	134	90	39
151	68	12	194	63	114	177	92	189	102
175	181	114	6	85	109	70	51	211	111
228	101	197	102	207	194	97	56	203	244
84	172	113	214	197	43	221	254	132	227
151	40	52	105	192	212	203	82	137	24
29	35	174	127	49	127	38	15	218	144
238	179	150	209	226	254	1	222	157	254
136	123	206	59	128	231	148	217	190	151
64	171	22	161	170	187	229	252	197	149
238	149	5	31	221	124	217	54	142	162
9	158	93	13	126	50	32	53	38	49
11	163	73	138	178	128	138	114	32	126
219	224	70	54	145	164	107	53	243	22
28	37	43	159	147	14	239	187	189	17
221	240	253	220	202	132	46	103	35	8
241	78	76	86	120	166	7	216	144	219
90	115	14	46	170	85	231	31	254	139
181	256	74	107	119	196	210	26	46	93
15	134	86	45	54	232	173	120	234	27
191	189	144	48	153	77	35	55	230	19

Figure. 5 Partial result

Command Window

K2 =

Columns 1 through 11

96	25	159	158	214	242	15	32	136	251
89	29	190	132	115	109	159	195	118	143
22	158	123	248	93	9	199	77	167	206
101	169	181	119	182	40	217	53	242	88
7	188	46	52	137	246	60	111	66	177
149	250	191	37	244	182	23	79	26	14
98	14	127	85	26	69	34	32	164	135
30	70	122	227	163	119	66	148	29	53
82	117	35	162	211	82	120	135	97	241
223	146	40	249	219	217	218	184	85	83
219	86	36	95	223	198	90	51	241	136
200	115	137	42	212	143	116	112	53	202
184	77	82	113	78	202	163	217	103	9
18	176	13	26	152	202	234	31	69	118
129	36	109	247	169	41	179	37	69	195
158	126	196	250	139	47	154	207	194	216
244	25	68	227	118	132	115	46	235	252
217	201	198	162	108	240	30	133	168	176
161	239	59	145	9	175	182	142	30	249
83	198	212	2	172	1	232	7	5	144

Figure. 6 Some results of K2 generation, starting with K-257th

The *S*-array is initialized to the permutation of the identity by giving it a value of *0* to *255*. The *K*-array, meanwhile, contains the *K2*-key. The *S* and *K*-arrays are permuted through *256* iterations to randomize key positions. The generation of the key scheduling algorithm is presented below:

1) Initialize an S-array with a length of 256 bytes to form an array S[0]=0, S[1]=1, S[2]=2, S[3]=3,…, S[255]=255.

Index	0	1	2	3	4	5	...	254	255
S	0	1	2	3	4	5	...	254	255

2) Initialize a *K*-key array to form an array: K[0]=140, K[1]=185, K[2]=134, K[3]=255, K[4]=56, K[5]=28, … , K[254]=137, K[255]=28.

Index	0	1	2	3	4	5	...	254	255
K	140	185	134	255	56	28	...	137	28

3) Permutation of the *S*-array value, and swapping the *S[i]* and *S[j]* arrays:

i=0;
j=0;
for i=0 to 255;
j= (j + S[j] + K[i]) mod 256;
swap value of *S[i]* and *S[j]*;

According to the algorithm, the value of i=0 to i=255 is obtained using the following *S*-array value.

4) Iteration-1, for i=0, j=0.

j= (j + S[j] + K[i]) mod 256;
= (j + S[0] + K[0]) mod 256;
= (0 + 0 + 140) mod 256;
= 140.

Received: October 18, 2020. Revised: November 11, 2020.

Swap the array values of S[0] and S[140], and the S-array becomes:

Index	0	1	2	3	4	5	...	139	140
S	140	1	2	3	4	5	...	139	0

5) Iteration-2, for i=1, j=140 (obtained from Iteration-1)

j= (j + S[j] + K[i]) mod 256;
= (j + S[140] + K[1]) mod 256;
= (140 + 140 + 185) mod 256;
= 209.

Swap the array values of S[1] and S[209], and the S-array becomes:

Index	0	1	2	3	4	5	...	208	209
S	140	209	2	3	4	5	...	208	1

6) Iteration-3, for i=2, j=209 (obtained from Iteration-2)

j= (j + S[j] + K[i]) mod 256;
= (j + S[209] + K[2]) mod 256;
= (209 + 209 + 134) mod 256;
= 40.

Swap the array values of S[2] and S[40], and the S-array becomes:

Index	0	1	2	3	4	5	...	40	...
S	140	209	40	3	4	5	...	2	...

7) Iteration-4, for i=3, j=40 (obtained from Iteration-3)

j= (j + S[j] + K[i]) mod 256;
= (j + S[40] + K[3]) mod 256;
= (40 + 40 + 255) mod 256;
= 79.

Swap the array values of S[3] dan S[79], and the S-array becomes:

Index	0	1	2	3	4	5	...	79	...
S	140	209	40	79	4	5	...	3	...

The system performs permutation until the 256th iteration, resulting in a random S-array. If the *IL*-data is more than 256 characters, or multiples of 256, the system generates block permutations in multiples of 256. This process produces a random *S*-array block.

4.4 *K4* Generation

After obtaining a random S-array (assuming that the array produced by Section 5.3 is the last array), the system continues to re-initialize the i and j values to zero. Here, the system generates pseudo-random algorithms using the following algorithm:

i=0;
j=0;
i= (i + 1) mod 256;
j= (j + S[i]) mod 256;
swap values of *S[i]* and *S[j]*;
t= (S[i] + S[j]) mod 256;
K= S[t];

Based on the previous block *S*-array results, the process of generating pseudo-random algorithms to produce *K4* is:

1) Iteration-1, for i=0, j=0,

i= (i + 1) mod 256;
= (0 + 1) mod 256
= 1
j= (j + S[i]) mod 256;
= (0 + S[1]) mod 256
= (0 + 209) mod 256
= 209

Swap the array values of S[1] and S[209], and the S-array becomes:

Index	0	1	2	3	4	5	...	209	210
S	140	1	40	79	4	5	...	209	210

K-1

t= (S[i] + S[j]) mod 256;
= (S[1] + S[209]) mod 256;
= (1 + 209) mod 256
= 210
K4 = S[t] = S[210] = 210;

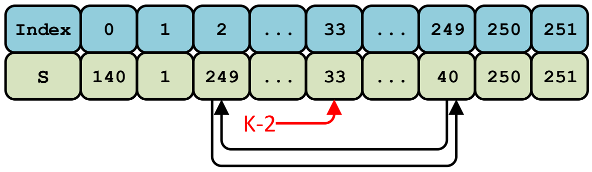
The first key character of *K4* is 210.

2) Iteration-2, for i=1, j=209 (obtained from Iteration-1):

i= (i + 1) mod 256;
= (1 + 1) mod 256
= 2
j= (j + S[i]) mod 256;
= (209 + S[2]) mod 256
= (209 + 40) mod 256
= 249

Swap the array values of S[2] and S[249] , and the S-array becomes:

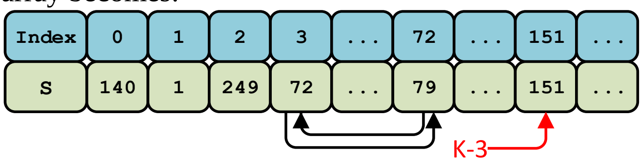
Received: October 18, 2020. Revised: November 11, 2020.



t= (S[i] + S[j]) mod 256;
= (S[2] + S[249]) mod 256;
= (249 + 40) mod 256
= 33
K4(2) = S[t] = S[33] = 33;
The second key character of *K4* is 33.

3) Iteration-3, for i=2, j=249, (obtained from Iteration-2):

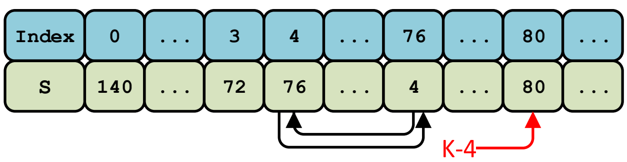
i= (i + 1) mod 256;
= (2 + 1) mod 256
= 3
j= (j + S[i]) mod 256;
= (249 + S[3]) mod 256
= (249 + 79) mod 256
= 72
Swap the array values of S[3] and S[72], and the S-array becomes:



t= (S[i] + S[j]) mod 256;
= (S[3] + S[72]) mod 256;
= (79 + 72) mod 256
= 151
K4(3) = S[t] = S[151] = 151;
The third key character of *K4* is 151.

4) Iteration-4, for i=3, j=72, (obtained from Iteration-3):

i= (i + 1) mod 256;
= (3 + 1) mod 256
= 4
j= (j + S[i]) mod 256;
= (72 + S[4]) mod 256
= (72 + 4) mod 256
= 76
Swap the array values of S[4] and S[76], and the S-array becomes:



t= (S[i] + S[j]) mod 256;
= (S[4] + S[76]) mod 256;
= (4 + 76) mod 256
= 80
K4(4) = S[t] = S[80] = 80;
The fourth key character of *K4* is 80.

The system continuously generates *K4*-keys until the keys' length is equal to the plaintext *IL*-data size (33,046 characters).

After four-key-generation is complete, the system continues the encryption of *IL*-data. The system thus inserts the key and key-length information behind the ciphertext.

5. Result and discussion

This simulation produces a K4-key with a length of 33,046 characters, which is subsequently used to encrypt the transmission data. This cipher data is analyzed below to ensure security.

5.1 Visual analysis

Visual analysis aims to compare the distribution of plain and cipher data using histograms. Fig. 7 shows a histogram of the plaintext in the red chart and the ciphertext in the blue chart for the first 500 of the 33,046 characters.

The plaintext distribution ranges between 10 and 99, while the resulting ciphertext has a numerical distribution from 1 to 256. This shows that the ciphertext is more secure from cyberattacks as it has a larger range than the plain data.

5.2 Keyspace analysis

Keyspace analysis aims to ascertain the means of securing the keyspace and anticipating brute-force attacks. The keyspace must be sufficiently large—greater than 2^{100}—to anticipate brute-force attacks [22].

In this simulation, we introduce a dynamic symmetric four-key-generators system. The system randomly generates a *K1*, with a length of *256* bytes, every session. Each session, the system generates a different key, as this is intended to increase confidentiality and integrity. Keys with a numerical distribution of *1* to *256*, and a length of *256* characters, have a keyspace of $(256)^{256}$ (equivalent to $\approx 2^{2048}$). As such, *K1* is secure from brute-force attacks.

5.3 Entropy analysis

Entropy analysis aims to determine the randomness of the amount of information in a message. The ciphered data is declared secure if it has an entropy value close to ≈8.00 [23]. Randomness is positively correlated with security. The entropy value follows the following equation [24]:

$$H = -\sum_{k=0}^{n} P(k)\, Log_2(P(k)) \qquad (2)$$

H is the entropy value, *n* is the number of different symbols in the message, and *P(k)* is the probability of ciphertext symbol occurrence.

Based on Eq. (2), the entropy value of *IL*-data = 7.99. The ciphertext is thus secure, as the entropy value is very close to 8.00.

5.4 Correlation analysis

Correlation analysis aims to determine the correlation between the plaintext and the ciphertext. A correlation value of ***r = 1*** means that both data are the same, while a correlation value approaching zero (***r ≈ 0***) indicates increased inequality. The less the correlation between the plaintext and the ciphertext, the greater the randomness, and thus the greater the security.

Correlation is analyzed via the following formula:

$$\boldsymbol{r} = \frac{\boldsymbol{n}\,(\sum \boldsymbol{xy}) - (\sum \boldsymbol{x})\,(\sum \boldsymbol{y})}{\sqrt{[\boldsymbol{n}\sum \boldsymbol{x}^2 - (\sum \boldsymbol{x})^2]\,[\boldsymbol{n}\sum \boldsymbol{y}^2 - (\sum \boldsymbol{y})^2]}} \qquad (3)$$

Where *r* is the correlation value, *x* is the plaintext data, and *y* is the ciphertext data. Based on Eq. (3), it can be seen that the *IL*-data correlation value is ***0.00007***, meaning that the plaintext and ciphertext are very random and have no correlation (see Table 1). That shows that data transmissions are secure from cyberattacks.

5.5 Speed test analysis

Speed test analysis aims to measure the proposed model's processing speed and determine whether it still meets the criteria for real-time processing [25]. The encryption and decryption times for keys of several sizes are shown in Table 2. The result shows that the processing time still qualifies as real-time, as it is less than one second.

Table 2. Speed test value in seconds

IL (Key sizes)	16 chars	32 chars	64 chars	128 chars	256 chars
Encryption time	0.239	0.205	0.205	0.213	0.208
Decryption time	0.492	0.466	0.447	0.524	0.464
Total of time	0.731	0.671	0.652	0.737	0.672

Mohamed [13] reviewed some of the hybrid cryptographic approach to data transmission security. As for comparison to the proposed method, as shown in Table 3. He identified several studies, such as those conducted by [15, 16, 19], as combining multiple cryptographic schemes to strengthen security.

From Table 3, it can be seen that encryption and decryption time are significant metrics for evaluating the performance of data transmission.

However, some studies have not provided performance evaluations and security analyses.

This study uses these metrics to measure performance and present security in-depth to data transmission. It presents a security analysis based on visual analysis, keyspace, entropy, correlation, and speed testing.

6. Conclusion

This research introduces a dynamic symmetric four-key-generators system to secure data transmission. The results of the keyspace test ($\approx 2^{2048}$bits), entropy test (7.99), correlation test (0.00007), and histogram test show that data transmissions secured through this system are better protected from cyber-attacks.

We propose that other complex key generators may potentially be applied in ICS communication for future works.

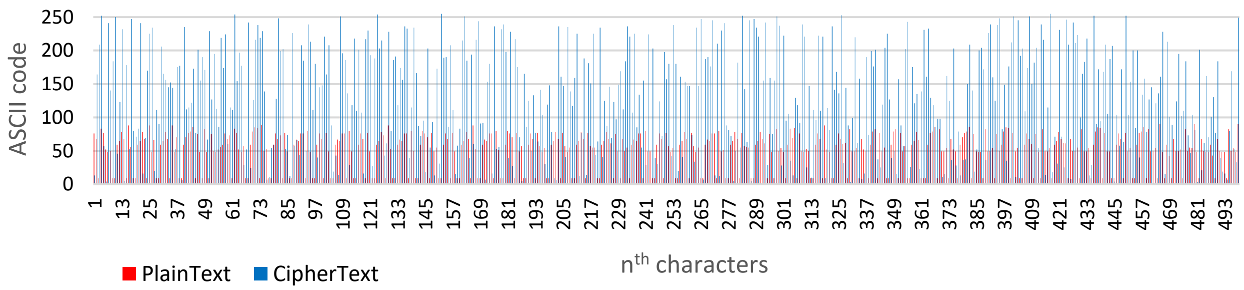


Figure. 7 Histogram of plaintext and ciphertext data for the first 500 characters (of 33,046 characters)

Received: October 18, 2020. Revised: November 11, 2020.

Table 1. Pearson correlation coefficient

Degree of correlation (r)								
Perfect	High	Moderate	Low	**No correlation**	Low	Moderate	High	Perfect
-1	≤ -0.90	≤ -0.50	≤ -0.30	**-0.29 ≤ r ≤ +0.29**	≥ +0.30	≥ +0.50	≥ +0.90	1

Table 3. Hybrid security approach to data transmission [13]

Study	Method	Performance Measuring	Presenting security analysis
[20]	Handshake agreement (SM2) and ECC.	No performance evaluation.	No
[14]	AES and steganography Word Shift Coding.	Encryption time and extraction time.	No
[15]	MD5, AES and ECDH.	Key exchange time, number of time, key length, time of signature, number of signature, verification time.	No
[18]	Symmetric encipherment and middle value algorithm.	Encryption and decryption test	No
[19]	Symmetric cipher Ping Pong-128, RSA and hash function MD5.	Encryption and decryption test.	No
[17]	DES and RC4.	No evaluation.	No
[16]	AES, RSA and HMAC.	Ciphertext size, encryption time	No
[21]	AES and Dynamic Key Generation, Dynamic S-box Generation.	Encryption and decryption test.	No
Proposed Method	Super Encryption BRC4, Four-key generators	Visual analysis, Keyspace, Entropy, Correlation, Speed Test.	Yes

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

The contributions of each author are described as follows: "Conceptualization, Riyadi, Priyambodo, Putra; methodology, Riyadi; software, Riyadi; validation, Priyambodo, Putra; formal analysis, Riyadi; investigation, Priyambodo; resources, Priyambodo, Putra; data curation, Riyadi; writing—original draft preparation, Riyadi; writing—review and editing, Priyambodo, Putra; visualization, Riyadi; supervision, Priyambodo, Putra; funding acquisition, Putra.

Acknowledgments

This research is supported by Rekognisi Tugas Akhir (RTA) from the Research Directorate of Universitas Gadjah Mada through contract number 2488/UN1.P.III/DIT-LIT/PT/2020.

References

[1] T. C. Pramod, G. S. Thejas, S. S. Iyengar, and N. R. Sunitha, "CKMI: Comprehensive key management infrastructure design for Industrial Automation and Control Systems", *Futur. Internet*, Vol. 11, No. 6, pp. 1–25, 2019, doi: 10.3390/fi11060126.

[2] B. Genge, P. Haller, A.-V. Duka, and H. Sándor, "A lightweight key generation scheme for end-to-end data authentication in Industrial Control Systems", *- Autom.*, Vol. 67, No. 5, pp. 417–428, 2019, doi: 10.1515/auto-2019-0017.

[3] P. Jain and P. Tripathi, "SCADA security: a review and enhancement for DNP3 based systems", *CSI Trans. ICT*, 2013, doi: 10.1007/s40012-013-0024-2.

[4] M. MacKintosh, G. Epiphaniou, H. Al-Khateeb, K. Burnham, P. Pillai, and M. Hammoudeh, "Preliminaries of orthogonal layered defence using functional and assurance controls in industrial control systems", *J. Sens. Actuator*

Received: October 18, 2020. Revised: November 11, 2020.

Networks, Vol. 8, No. 1, 2019, doi: 10.3390/jsan8010014.

[5] D. Puthal, S. Nepal, R. Ranjan, and J. Chen, "A Dynamic Key Length Based Approach for Real-Time Security Verification of Big Sensing Data Stream", *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, Vol. 9419, pp. 93–108, 2015, doi: 10.1007/978-3-319-26187-4.

[6] A. Schmidt, "The Estonian Cyberattacks", *Fierce Domain – Conflicts Cybersp. 1986-2012*, No. August, pp. 1–28, 2013.

[7] Z. Shouran, T. Priyambodo, and A. Ashari, "Information System Security : Human Aspects Information System Security : Human Aspects", No. March, 2019.

[8] G. Kilinc, I. N. Fovino, C. Ferigato, and A. Koltuksuz, "A model of distributed key generation for industrial control systems", *IFAC Proc. Vol.*, Vol. 45, No. 29, pp. 356–363, 2012, doi: 10.3182/20121003-3-MX-4033.00057.

[9] B. Luca, G. Kilinc, E. Mangioni, and L. Pomello, "Modeling Distributed Private Key Generation by Composing Petri Nets", *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, Vol. 8910, pp. 19–40, 2014, doi: 10.1007/978-3-662-45730-6.

[10] B. Rajkumar and G. Narsimha, "Secure light weight encryption protocol for MANET", *Int. J. Intell. Eng. Syst.*, Vol. 10, No. 3, pp. 58–65, 2017, doi: 10.22266/ijies2017.0630.07.

[11] M. Ouaissa, M. Ouaissa, and A. Rhattoy, "An efficient and secure authentication and key agreement protocol of LTE mobile network for an IoT system", *Int. J. Intell. Eng. Syst.*, Vol. 12, No. 4, pp. 212–222, 2019, doi: 10.22266/ijies2019.0831.20.

[12] E. Setyaningsih, R. Wardoyo, and A. K. Sari, "Securing color image transmission using compression-encryption model with dynamic key generator and efficient symmetric key distribution", *Digit. Commun. Networks*, 2020, doi: 10.1016/j.dcan.2020.02.001.

[13] N. N. Mohamed, Y. M. Yussoff, M. A. Saleh, and H. Hashim, "Hybrid cryptographic approach for internet of things applications: A review", *J. Inf. Commun. Technol.*, Vol. 19, No. 3, pp. 279–319, 2020, doi: 10.32890/jict2020.19.3.1.

[14] A. Altigani and B. Barry, "A hybrid approach to secure transmitted messages using advanced encryption standard (AES) and Word Shift Coding Protocol", In: *Proc. of- 2013 Int. Conf. Comput. Electr. Electron. Eng. 'Research Makes a Differ. ICCEEE 2013*, pp. 134–139, 2013, doi: 10.1109/ICCEEE.2013.6633920.

[15] M. Xin, "A Mixed Encryption Algorithm Used in Internet of Things Security Transmission System", In: *Proc. of - 2015 Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discov. CyberC 2015*, pp. 62–65, 2015, doi: 10.1109/CyberC.2015.9.

[16] E. Harba, "Secure Data Encryption Through a Combination of AES, RSA and HMAC", *Eng. Technol. Appl. Sci. Res.*, Vol. 7, No. 4, pp. 1781–1785, 2017, doi: 10.5281/zenodo.844291.

[17] Z. Y. Hong, Z. P. Qiu, S. L. Zeng, S. De Wang, and M. Sandrine, "Research on fusion encryption algorithm for internet of things monitoring equipment", In: *Proc. of - 14th Int. Symp. Pervasive Syst. Algorithms Networks, I-SPAN 2017, 11th Int. Conf. Front. Comput. Sci. Technol. FCST 2017 3rd Int. Symp. Creat. Comput. ISCC 2017*, Vol. 2017-Novem, pp. 425–429, 2017, doi: 10.1109/ISPAN-FCST-ISCC.2017.49.

[18] R. Singh, I. Panchbhaiya, A. Pandey, and R. H. Goudar, "Hybrid Encryption Scheme (HES): An approach for transmitting secure data over internet", *Procedia Comput. Sci.*, Vol. 48, No. C, pp. 51–57, 2015, doi: 10.1016/j.procs.2015.04.109.

[19] S. Purevjav, T. Kim, and H. Lee, "Email encryption using hybrid cryptosystem based on Android", *Int. Conf. Adv. Commun. Technol. ICACT*, Vol. 2016, pp. 426–429, 2016, doi: 10.1109/ICACT.2016.7423418.

[20] N. Hong and Z. Xuefeng, "A security framework for internet of things based on SM2 cipher algorithm", In: *Proc. of - 2013 Int. Conf. Comput. Inf. Sci. ICCIS 2013*, pp. 13–16, 2013, doi: 10.1109/ICCIS.2013.12.

[21] F. J. D'souza and D. Panchal, "Advanced encryption standard (AES) security enhancement using hybrid approach", In: *Proc. of- IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017*, Vol. 2017-Janua, pp. 647–652, 2017, doi: 10.1109/CCAA.2017.8229881.

[22] R. Bellazreg, N. Boudriga, and M. Hamdi, "A dynamic distributed key tunneling protocol for heterogeneous wireless sensor networks", 2012, doi: 10.1109/TrustCom.2012.26.

[23] E. Setyaningsih and R. Wardoyo, "Review of Image Compression and Encryption Techniques", Vol. 8, No. 2, pp. 83–94, 2017.

[24] A. Shukla and S. Kumar, "Analysis of secure watermarking based on DWT-SVD technique for piracy", In: *Proc. of- IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2016*, pp. 1110–1115, 2017, doi: 10.1109/CCAA.2016.7813882.

[25] E. H. Riyadi, T. K. Priyambodo, and A. E. Putra, "Real-time Testing on Improved Data Transmission Security in the Industrial Control System", In: *Proc. of- IEEE Int. Semin. Res. Inf. Technol. Intell. Syst.*, 2020.