ISSN 2085-4552

Implementing the Chaotic Permutation Multicircular Cryptography Technique using Asymmetric Key

Aria Lesmana[1], Ruki Harwahyu[2], Yohan Suryanto[3]
[1,2,3] Department of Electrical Engineering, Universitas Indonesia, Depok, Indonesia
[1]aria.lesmana01@ui.ac.id, [2]ruki.h@ui.ac.id, [3]yohan.suryanto@ui.ac.id

Accepted 18 October 2022
Approved 30 November 2022

***Abstract*— In digital computing, cryptographic methods consider performance in both speed and security. This study aims to explore and improve a permutation-based symmetric chaotic cryptography technique called Chaotic Permutation Multicircular (CPMC). In this study, a method is proposed to implement asymmetric key system from CPMC technique by generating a reverser key for reverting the permutation result of CPMC encryption back to its original arrangement using the same function as the encryption process. The reverser key alongside the CPMC key act as encryption and decryption key pair. The pair key generation and cryptographic function utilizes the encryption function of CPMC technique dubbed CPMC Shrinking algorithm. Asymmetric implementation can simplify CPMC technique by also using CPMC Shrinking algorithm for decryption, therefore enabling it as a single function for encryption and decryption. The asymmetric implementation test showed improvement in total speed compared to initial implementation by average of 75.87% from tests using different block sizes.**

***Index Terms*—asymmetric cryptography; chaos cryptography; chaotic permutation; cryptography; permutation-based cryptography**.

I. INTRODUCTION

Digital communications are still vulnerable to security risks, such as eavesdropping or data leakage. Digital systems generally secure data by implementing a cryptographic system. Cryptographic systems in addition to requiring key confidentiality guarantees, also require high randomization for the confidentiality of the results and algorithms that are not heavy for encryption and decryption.

The application of Chaos Theory in mathematics for cryptographic methods gave rise Chaos Cryptography, through making use of mathematics functions with chaotic properties to be used as a pseudorandom number generator and can be combined with other functions or algorithms to become a cryptographic method. The widely used chaos mathematical function is in the form of a map, or dubbed chaotic map. Chaos cryptography methods is widely applied as an image encryption method [1-6], such algorithms also been developed using public key or as an asymmetric cryptography, especially the ones using Chebyshev Polynomial [15], [16], another method proposed by Silva-Garcia et. al. uses Elliptic Curve function alongside S-box Permutation generated by chaos resulting from Logistic Map function [17].

Permutation method in cryptography is widely applied alongside the substitution cipher method [7-14]. In this study, we explore the application of a permutation-based cryptography with a chaotic characteristics dubbed Chaotic Permutation Multicircular (CPMC). CPMC is a chaotic cryptographic technique based on permutations by Suryanto et al. [18], CPMC algorithm is suitable for data encryption due to its large keyspace of $(2^N)!$ [19]. CPMC algorithm applies rotational shift of a set elements, using a key generated with special modulus and LCM rules towards a set of natural number sequences to produce chaotic properties. CPMC algorithm has been implemented for image encryption [20-22] and audio [23], however both implementations are limited to symmetric block cipher algorithm.

This study proposes an asymmetric key cryptography implementation of the CPMC technique, by designing an algorithm that can generate a key pair from the CPMC encryption key. The current implementation of CPMC is a symmetric cryptography with a single key value, but the permutation functions are feasible to be utilized for an asymmetrical key values, so the proposed design expands the implementation to enable the usage of asymmetrical cryptography. For the implementation of cryptography, the encryption algorithm function of the CPMC technique, namely CPMC Shrinking, can be used as an encryption algorithm as well as decryption using the generated key pair. This works aims to:

- Explore an alternative implementation for the CPMC encryption technique as an asymmetric cryptography and its feasibility.
- Simplify the encryption-decryption processes by using a single algorithm function for running

both processes and analyzing its cryptographic performance improvements.

- Discover a different approach in implementing key generation and management for CPMC, by using different values of keys that can only encrypt and decrypt each other's messages, as opposed to the basic CPMC implementation as symmetric key algorithm.

II. PRELIMINARIES

A. *Asymmetric Key*

Asymmetric key cryptography is a cryptography algorithm in which the algorithms use a pair of different keys and use a different component of the pair for conducting encryption and decryption operations separately. Also widely known as Public Key Encryption [24]. For CPMC impelementation, both the encryption and decryption keys are presented as set of values, and both uses the same single function to conduct their respective operations. Here the usage of asymmetric pair of keys in CPMC algorithm is analyzed for feasibility of public key cryptography.

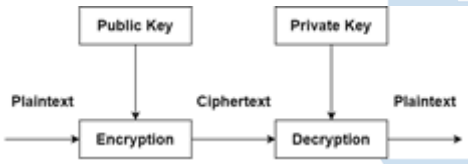

Fig. 1. Asymmetric Key Cryptography

B. *Chaotic Permutation Multicircular*

Chaotic Permutation Multicircular (CPMC) is a symmetric cryptographic technique utilizing permutation function with chaotic mathematical properties. The algorithm performs rotational permutations to the arrangement of array elements, accompanied by changes in the permutation space each iteration of the permutation process is complete. The CPMC technique consists of two cryptographic functions, CPMC Shrinking as the encryption function and CPMC Expanding as the decryption function. CPMC technique utilize its own key generator function dubbed Expanded Key Generator [18] [19], while the key value used by CPMC is an array of natural numbers with a length of N – 1 where N is the length of the plaintext. The value of the key element determines the shift distance of the permutation on the plaintext element, while the position of the key element value in the array determines which order the elements is permuted.

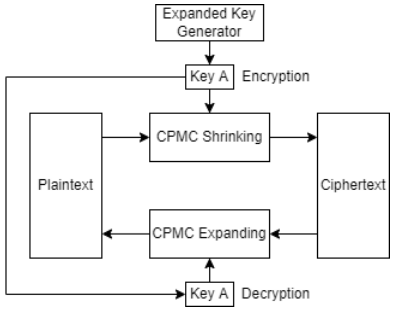

Fig. 2. Basic CPMC Encryption and Decryption Process

1) *CPMC Shrinking*

CPMC Shrinking (CPMCS) is a permutation function that acts as the encryption algorithm for CPMC. The permutation begins on all elements of the array, then the permutation is continued with the previous result with a range starting at the next index to the last index. The CPMCS permutation algorithm can be summarized in several steps [19]:

1. Determine the input set of plaintext X of size N, and the key array set Key of size N – 1.
2. Initiate the index loop iterator value n = 1.
3. Start of the (n)th Permutation by clockwise array rotation within range of (N – n + 1) on array X from its (n)th element to the last (X[n:N]), elements in the range are shifted as far as the (n)th element value of the key set array (Key[n]).
4. The first element of the result set of X is stored in Y[n], the stored element is not included in the next permutation.
5. Increment the loop iterator n = n + 1.
6. Check whether the value of n == N. If not, repeat step 4. If yes, continue to the next step.
7. The results of the remaining X permutations are stored at the last index Y (Y[N] = X[N]). Array Y is output as the result of CPMCS permutations.

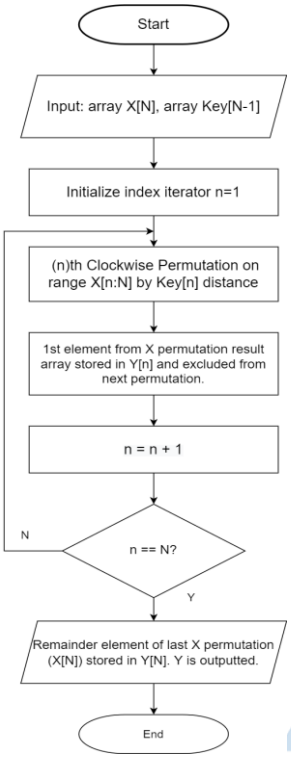

Fig. 3. CPMC Shrinking

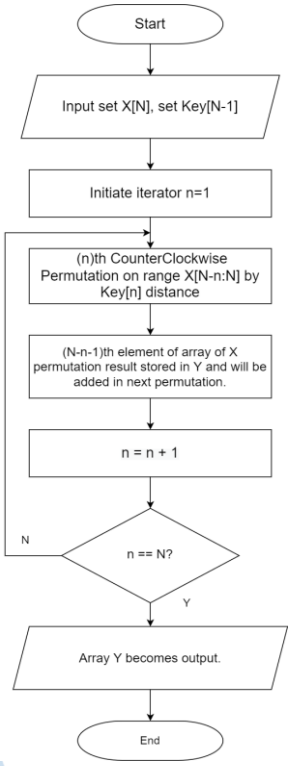

Fig. 4. CPMC Expanding

2) CPMC Expanding

CPMC Expanding (CPMCE) acts as a decryption function of the CPMC technique, the permutation process applied is the opposite of the CPMCS permutation. The CPMCE permutation algorithm can be summarized in several steps [19]:

1. Determine the input set of ciphertext X of size N, and the key array set Key of size N−1.
2. Initiate the index loop iterator value n = 1, with the last two elements X (X[N] and X[N – 1]).
3. Permuting the (n)th counterclockwise rotation with a range of elements of n + 1 on X from index N−n−1 to the last as far as the value of Key[n].
4. The (N − n − 1)th element of the result of the permutation X is stored in the Y set and becomes an additional element in the input of the next permutation.
5. Increment the loop iterator n = n + 1.
6. Check whether the value of n == N. If not, repeat step 4. If yes, continue to the next step.
7. Array Y results from the last iteration is output as the result of CPMCE permutation.

3) CPMC Expanded Key Generator

The expanded key generator algorithm for the CPMC technique serves as a generator for the permutation key value set for the CPMCS and CPMCE algorithms to determine the element shift in the permutation's multi-rotational movement process. This function uses three parameters of integer inputs, the first two are the initial key and sequence key as the main seed value parameters that determines how high the values of the key elements which determine the permutation shift distances. The third parameter is block size N to determine the key array size and the size limit of input that is eligible for the encryption and decryption process. The output of the function is an array of integers with the size of N−1 elements as the key. The keys generated from this function has a key space of (N!). The CPMC Expanded Key Generator algorithm can be summarized in several steps [19]:

1. Input the parameters values of 'Initial Key' and 'Sequence key' as the key seed, and N as the block size and the element permutation length.
2. Initiate iterator n = 1, and first element of array 'Temp' as Temp[n] = Initial Key.
3. Determine the value of the base modulus of the initial key 'Bi' as Bi[n] = (N – n + 1), and the base modulus of the sequence key 'Bs' as Bs[n] = (N – n + 1).
4. If initially Bi[n] is a prime number, then Bi[n] = 1, or if all prime factors of Bi[n] are the same value, then Bi[n]=Bi[n]/prime factor. If neither then the value of Bi[n] remains.

5. Calculate KeyI[n] as the modulus of Temp[n] with base Bi[n] (KeyI[n] = mod(Temp[n], Bi[n]))), and KeyS[n] as the modulus of the sequence key with base Bs[n] (KeyS[n] = mod(Sequence, Bi[n])).
6. Calculate Key[n] as the modulus of the sum KeyI[n] + KeyS[n] with base Bs[n].
7. Calculating the rounding down of the quotient Temp[n]/Bi[n], the result is added with KeyI[n] and stored as Temp[n].
8. Increment the iterator n = n + 1.
9. Check whether n == N, if not then the process returns to stage 3, if yes then the process continues to stage 10.
10. Array Key is outputted as the CPMC cryptographic key set.

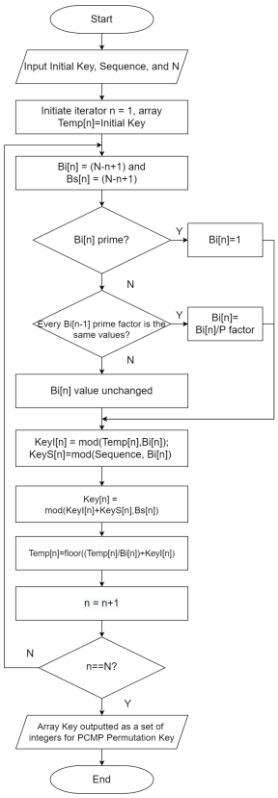

Fig. 5. CPMC Expanded Key Generator Function

III. Proposed Method

A. CPMC Asymmetric Implementation

The asymmetric implementation of CPMC can be described as such, if a CPMC function permute a set X using Key A into different arrangement resulting in set Y (Y=f(X,A)), then there is another set of values denoted as Key B that can revert Y to X using the same function. The encryption and decryption functions are implemented using the CPMCS function. From its permutative nature, the two keys in the pair can decrypt each other's permutations. The results of the encryption from the base key can be decrypted using the pair key, as well as the result of the pair key permutation can be decrypted using the base key (Figure 6).

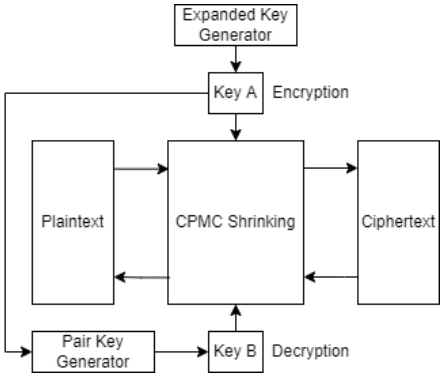

Fig. 6. CPMC Cryptography with Asymmetric Key

B. CPMC Asymmetric Pair Key Generation

The asymmetric key implementation of the CPMC technique uses a CPMC Permutation Key Finder algorithm as the pair key generator. The CPMC key finder algorithm is able to generate a CPMC key based on existing permutation patterns. If the CPMCS function permutes setA to setB using key KeyA, then the Key Finder function will return the reverser key dubbed KeyB that can permute setB back to setA using the CPMCS function as well. The CPMC Key Finder Algorithm generates an asymmetric pair key by using a set filled with unique values. The algorithm steps for the CPMC Key Pair Finder function, dubbed as "kFind", is described as such:

1. Get input setA as the initial set containing unique values of numbers up to the N length of block and setB as the permutation result of setA using KeyA. Declare iterator i as 0, and KeyB as an empty array initially for storing resulting key value.
2. Element of KeyB on KeyB[i] is incremented. Then KeyB is used to permute setB and the result is stored in Temp. (Temp=CPMCS(setB,KeyB)).
3. Check if the value of the Temp[i] element is equal to the setA[i] element. If not, repeat previous step until the element Temp[i] is equal to the value in setA[i].
4. If the element value Temp[i] == setA[i], then the iterator i is incremented to change the position index check of KeyB and setA to the next index. Then repeat step 3 until i reaches the last index of the key (i == N−1).
5. KeyB is outputted as the key pair of KeyA, which can permute setB back to setA.

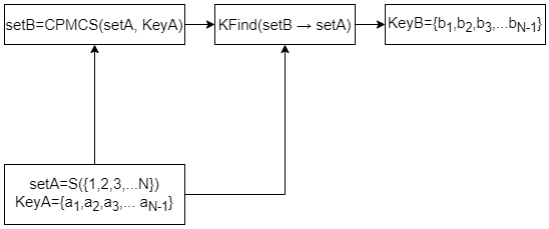

Fig. 7. CPMC Asymmetric Key Generation

IV. RESULTS

The algorithm design and test are applied using Matlab 2018b platform. The test is conducted in a computer with the specifications of i7-10750H 2.60GHz CPU, 16 GB of RAM and a 64-bit Windows 10 OS. The test consists of Pair Key Generation function, encryption and decryption test, and its runtime measurement to compare with the basic implementation of source method.

A. Pair Key Generation

The key generation algorithm is tested for an array length of 100 elements (N). For the test, the base key is generated using the PCMP Expanded Key Generator function with the input parameters of N=100, Initial Key = 12345678 and Sequence Key = 1234. The resulting base key contains the set of values: {12, 28, 14, 70, 53, 66, 79, 92, 13, 27, 41, 77, 69, 83, 11, 26, 41, 72, 71, 32, 49, 49, 1, 17, 33, 49, 65, 66, 25, 27, 59, 7, 25, 28, 61, 14, 33, 52, 9, 14, 49, 54, 31, 52, 17, 39, 7, 15, 1, 25, 49, 10, 37, 12, 41, 22, 5, 30, 19, 4, 37, 28, 21, 13, 13, 12, 13, 16, 21, 25, 7, 16, 5, 22, 15, 12, 13, 15, 5, 19, 17, 18, 13, 10, 5, 7, 5, 12, 1, 2, 7, 1, 3, 2, 5, 4, 3, 1, 0}.

Using Pair Key Finder, the base key is inputted and the resulting pair key is: { 19, 23, 36, 62, 67, 19, 26, 88, 67, 9, 23, 55, 61, 84, 0, 11, 34, 21, 53, 43, 41, 22, 74, 36, 23, 52, 45, 20, 65, 45, 54, 11, 9, 11, 2, 11, 32, 26, 61, 12, 6, 37, 47, 8, 55, 26, 46, 50, 15, 33, 11, 21, 26, 29, 45, 37, 5, 15, 41, 24, 11, 36, 20, 17, 20, 21, 17, 24, 27, 11, 27, 24, 15, 18, 22, 12, 7, 2, 16, 6, 13, 10, 17, 1, 15, 2, 11, 12, 4, 5, 8, 3, 6, 1, 3, 1, 2, 2, 1}.

B. Encryption & Decryption Function Test

The key pair generated is used for permutation testing of the encryption function against a text input string of 93 characters long including spaces, which has been padded with additional spaces to achieve a match with a key block size of 100 characters. The input text is stored in the 'plaintext' variable. Since CPMC uses permutation as its method for encryption and decryption, the encryption and decryption test using the asymmetric key pair shows the interchangeability of each key usage for either encryption or decryption function, with one key encryption can be decrypted using the other in the pair.

Encryption is applied using both keys in the pair, the encryption result of the base key (key1) is stored in the variable 'ciphertext1' (Figure 10), while the encryption result of the pair key (key2) is stored in the variable 'ciphertext2' (Figure 11). The decryption function test is carried out using the ciphertext generated from the encryption experiment, namely 'ciphertext1' and 'ciphertext2'. Decryption is performed using the PCMP key as opposed to the key used for encryption. 'ciphertext1' is decrypted using the key pair 'key2' (Figure 12), while 'ciphertext2' is decrypted using the base key 'key1'.

```
>> plaintext=pad(plaintext,length(key1)+1)
plaintext =
    'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut.      '
>> plaintext-0
ans =
  Columns 1 through 17
    76   111   114   101   109    32   105   112   115   117   109    32   100   111   108   111   114
  Columns 18 through 34
    32   115   105   116    32    97   109   101   116    44    32    99   111   110   115   101    99
  Columns 35 through 51
   116   101   116   117   114    32    97   100   105   112   105   115    99   105   110   103    32
  Columns 52 through 68
   101   108   105   116    44    32   115   101   100    32   100   111    32   101   105   117   115
  Columns 69 through 85
   109   111   100    32   116   101   109   112   111   114    32   105   110    99   105   100   105
  Columns 86 through 100
   100   117   110   116    32   117   116    46    32    32    32    32    32    32    32
```

Fig. 8. Plaintext Input as String and ASCII Number

```
>> ciphertext2=PCMPK(plaintext,key2)
ciphertext2 =
  Columns 1 through 17
   105   112   110   115   108   101   101   100   115   105   100   111    76    32    32    32   116
  Columns 18 through 34
   105    99    32   101    32   109    32   105    97   100   109   101   115   105   111   111    46
  Columns 35 through 51
    32   111   105   112   105    32   117   111   100    32    32   105   101    44   115   109   110
  Columns 52 through 68
   110   100   115    32   117   114   114   116   117   116   115    32   116   101    97   105    32
  Columns 69 through 85
    44   105   112   100    32   116   111    99    32   117    32   116   116    32   116   117   109
  Columns 86 through 100
   114    32    99    99   109   101   111   114   101   108   100    32   103   110    32
>> sprintf(char(ciphertext2))
ans =
  'ipnsleedsidoL   tic e m iadmesico. oipi uod  ie,smnnds urrtuts teai ,ipd toc u tt tumr ccmeoreld gn '
```

Fig. 9. Pair Key Encryption

```
>> decrypted2=PCMPK(ciphertext2,key1)
decrypted2 =
  Columns 1 through 17
    76   111   114   101   109    32   105   112   115   117   109    32   100   111   108   111   114
  Columns 18 through 34
    32   115   105   116    32    97   109   101   116    44    32    99   111   110   115   101    99
  Columns 35 through 51
   116   101   116   117   114    32    97   100   105   112   105   115    99   105   110   103    32
  Columns 52 through 68
   101   108   105   116    44    32   115   101   100    32   100   111    32   101   105   117   115
  Columns 69 through 85
   109   111   100    32   116   101   109   112   111   114    32   105   110    99   105   100   105
  Columns 86 through 100
   100   117   110   116    32   117   116    46    32    32    32    32    32    32    32
>> sprintf(char(decrypted2))
ans =
  'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut.      '
```

Fig. 10. Base Key as Description Key

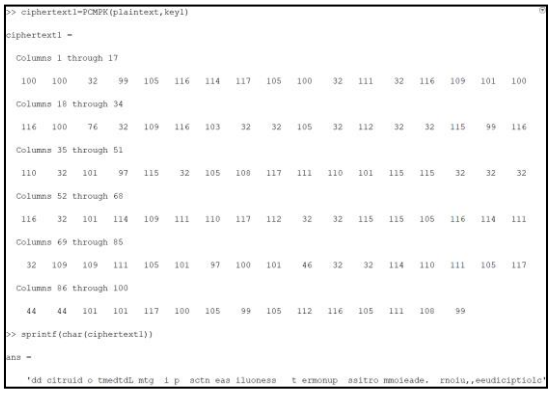

Fig. 11. Base Key Encryption

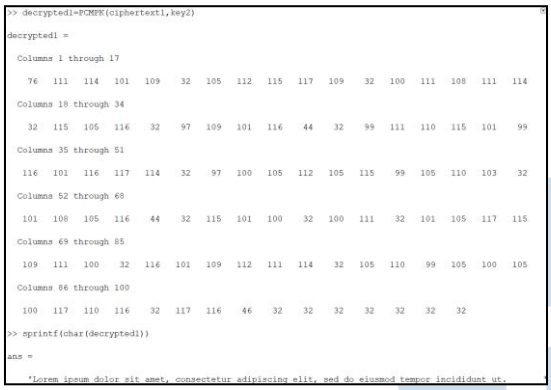

Fig. 12. Pair Key Description

C. *Encryption & Decryption Runtime*

In testing the CPMC cryptographic algorithm, runtime measurements of the encryption and decryption functions were also carried out in Matlab. Measurements were made during the running of the proposed asymmetric implementation algorithm program and the basic implementation as symmetric cryptography, both of which were also run with different input lengths (N) starting from 100 to 1000 elements in size. The basic implementation process uses the CPMC Shrinking and Expanding permutation algorithm separately as encryption and decryption functions, respectively, using the key generated by the Expanded Key Generator function, while the asymmetric implementation uses CPMC Shrinking as the encryption and decryption function, where the encryption process uses the same key (Expanded Key) as in the basic implementation, and the decryption process uses the key (Pair Key) generated by the KFind function, so that the runtime measured is one time encryption process using CPMCS with two decryption processes using the CPMCE and CPMCS functions respectively along with their difference ratio percentage.

TABLE I. RUNTIME OF ENCRYPTION AND DECRYPTION TEST BY PERMUTATION LENGTH (N)

N	Runtime (s)			*Runtime Difference (%)*
	CPMCS Encryption	***CPMCE***	***CPMCS Decryption***	
100	1.11E-04	5.13E-04	1.12E-04	78.18
200	2.22E-04	9.76E-04	2.13E-04	78.2
300	2.87E-04	1.28E-03	2.69E-04	76.98
400	4.44E-04	1.78E-03	4.41E-04	75.21
500	4.91E-04	2.28E-03	4.95E-04	78.29
600	6.41E-04	2.56E-03	6.42E-04	74.94
700	7.84E-04	3.02E-03	7.57E-04	74.94
800	9.72E-04	3.48E-03	8.75E-04	74.85
900	1.13E-03	3.94E-03	1.01E-03	74.32
1000	1.14E-03	4.19E-03	1.14E-03	72.81
Average				75.87

TABLE II. RUNTIME COMPARISON WITH SOURCE DATA AT N=300

Test	Runtime (s)		
	Encryption	***Decryption***	***Total***
CPMCS + CPMCE Test	2.87E-04	1.28E-03	1.57E-03
Asymmetric Key CPMCS	2.87E-04	2.69E-04	5.57E-04
CPMC Source [19]	3.08E-03	3.22E-03	6.30E-03

The runtime measurement results in Table I show that the speed of the CPMCS function both as an encryption and decryption function runs faster than CPMCE, where the use of CPMCS as a decryption function increases the decryption runtime by an average of 75.87%. Then the measured runtimes were compared with data from literature sources that recorded measurements at N=300 (Table 2). This comparison is used to show the difference between the results of the basic implementation test and the proposed method with the source data, in which the Matlab test version runtime are found to be faster for both basic and proposed implementation compared to the source data. The basic implementation CPMC encryption and decryption runtime on the running test in Matlab give results of a total of 1.57E-03 seconds, while the source data gives a total of 6.30E-03 seconds. The proposed asymmetric implementation results in a total runtime of 5.57E-04 seconds, which is an increase of 75.16% compared to basic CPMC from the Matlab test and by 91.17% from the source data.

D. *Public Key Feasibility*

The permutation key and CPMC Shrinking algorithm properties made it so that a certain key set values are only pairable with a specific key set value so that one key's encryption result can only be decrypted using the other key using the same permutation

function. However, since the base permutation algorithms were meant as a symmetric cryptographic function, the leakage of one pair of keys can compromise the secrecy of the encrypted data content. A modification on the permutation step and encryption process while without requiring to modify the key value can add better secrecy.

V. CONCLUSIONS

CPMC technique can apply cryptography asymmetrically using key pairs that can perform decryption using the same algorithm for encryption, namely CPMC Expanding, where the results of one key encryption permutation can be decrypted using another key in the pair. The implementation of asymmetric keys in the CPMC technique can simplify the work of the algorithm by using the same function to perform encryption and decryption, and produce better performance at the decryption speed than the basic CPMC technique. This means the speed of the CPMC Shrinking algorithm is found to be faster than the CPMC Expanding, so the implementation of asymmetric key cryptography for CPMC can be used as a more efficient alternative for fast encryption and decryption method using a pre-generated key. The use of CPMC Shrinking as a decryption function results in an average decryption speed increase of 75.87% from testing with different iterations of block size.

REFERENCES

[1] M. Kumar, A. Saxena, and S. S. Vuppala, "A Survey on Chaos Based Image Encryption Techniques," in Multimedia Security Using Chaotic Maps: Principles and Methodologies, K. M. Hosny Ed. Cham: Springer International Publishing, pp. 1-26, 2020.

[2] M. T. Suryadi, Y. Satria, and M. Fauzi, "Implementation of digital image encryption algorithm using logistic function and DNA encoding," Journal of Physics: Conference Series, vol. 974, p. 012028, 2018/03 2018, doi: 10.1088/1742-6596/974/1/012028.

[3] M. Wang, X. Wang, Y. Zhang, S. Zhou, T. Zhao, and N. Yao, "A novel chaotic system and its application in a color image cryptosystem," Optics and Lasers in Engineering, vol. 121, pp. 479-494, 2019, doi: https://doi.org/10.1016/j.optlaseng.2019.05.013

[4] A. M. Elshamy, A. I. Hussein, H. F. A. Hamed, M. A. Abdelghany, and H. M. Kelash, "Color Image Encryption Technique Based on Chaos," Procedia Computer Science, vol. 163, pp. 49-53, 2019/01/01/ 2019, doi: https://doi.org/10.1016/j.procs.2019.12.085.

[5] Y. Liu, J. Zhang, D. Han, P. Wu, Y. Sun, and Y. S. Moon, "A multidimensional chaotic image encryption algorithm based on the region of interest," Multimedia Tools and Applications, vol. 79, no. 25, pp. 17669-17705, 2020/07/01 2020, doi: 10.1007/s11042-020-08645-8.

[6] S. Zhou, X. Wang, M. Wang, and Y. Zhang, "Simple colour image cryptosystem with very high level of security," Chaos, Solitons & Fractals, vol. 141, p. 110225, 2020/12/01/ 2020, doi: https://doi.org/10.1016/j.chaos.2020.110225.

[7] Y. Naseer, T. Shah, and D. Shah, "A novel hybrid permutation substitution base colored image encryption scheme for multimedia data," Journal of Information Security and Applications, vol. 59, p. 102829, 2021/06/01/ 2021, doi: https://doi.org/10.1016/j.jisa.2021.102829.

[8] L. Teng, X. Wang, and Y. Xian, "Image encryption algorithm based on a 2D-CLSS hyperchaotic map using simultaneous permutation and diffusion," Information Sciences, vol. 605, pp. 71-85, 2022/08/01/ 2022, doi: https://doi.org/10.1016/j.ins.2022.05.032.

[9] E. A. Albahrani and T. K. Alshekly, "New Chaotic Substation and Permutation Method for Image Encryption," International Journal of Applied Information Systems, vol. 12, pp. 33-39, 2017.

[10] J. I. Moreira Bezerra, V. Valduga de Almeida Camargo, and A. Molter, "A new efficient permutation-diffusion encryption algorithm based on a chaotic map," Chaos, Solitons & Fractals, vol. 151, p. 111235, 2021/10/01/ 2021, doi: https://doi.org/10.1016/j.chaos.2021.111235.

[11] R. S. Devi, K. Thenmozhi, R. Amirtharajan, and P. Padmapriya, "A Novel Multiple Segmented Image Encryption," in 2019 International Conference on Computer Communication and Informatics (ICCCI), 23-25 Jan. 2019 2019, pp. 1-5, doi: 10.1109/ICCCI.2019.8822155.

[12] M. Wang, X. Wang, Y. Zhang, and Z. Gao, "A novel chaotic encryption scheme based on image segmentation and multiple diffusion models," Optics & Laser Technology, vol. 108, pp. 558-573, 2018/12/01/ 2018, doi: https://doi.org/10.1016/j.optlastec.2018.07.052.

[13] S. Sun, Y. Guo, and R. Wu, "A Novel Image Encryption Scheme Based on 7D Hyperchaotic System and Row-column Simultaneous Swapping," IEEE Access, vol. 7, pp. 28539-28547, 2019, doi: 10.1109/ACCESS.2019.2901870.

[14] A. A. Karawia and Y. A. Elmasry, "New Encryption Algorithm Using Bit-Level Permutation and Non-Invertible Chaotic Map," IEEE Access, vol. 9, pp. 101357-101368, 2021, doi: 10.1109/ACCESS.2021.3096995.

[15] I. Mishkovski and L. Kocarev, "Chaos-Based Public-Key Cryptography," in Chaos-Based Cryptography: Theory, Algorithms and Applications, L. Kocarev and S. Lian Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 27-65, 2011.

[16] D. Yoshioka, "Security of Public-Key Cryptosystems Based on Chebyshev Polynomials Over Z/pk Z," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 10, pp. 2204-2208, 2020, doi: 10.1109/TCSII.2019.2954855.

[17] V. M. Silva-García, R. Flores-Carapia, M. D. González-Ramírez, E. Vega-Alvarado, and M. G. Villarreal-Cervantes, "Cryptosystem Based on the Elliptic Curve With a High Degree of Resistance to Damage on the Encrypted Images," IEEE Access, vol. 8, pp. 218777-218792, 2020, doi: 10.1109/ACCESS.2020.3042475.

[18] Y. Suryanto, Suryadi and K. Ramli, "Chaos properties of the Chaotic Permutation generated by Multi Circular Shrinking and Expanding Movement," 2015 International Conference on Quality in Research (QiR), 2015, pp. 65-68, doi: 10.1109/QiR.2015.7374896.

[19] Y. Suryanto, "Pengembangan dan Analisis Metode Permutasi Chaotic Baru Berbasis Multiputaran Mengecil dan Membesar untuk Enkripsi Citra dengan Tingkat Keamanan Tinggi, Cepat dan Tahan Terhadap Gangguan," Program Doktor [Dissertation], Universitas Indonesia, 2016.

[20] Y. Suryanto, Suryadi, and K. Ramli, "A Secure and Robust Image Encryption Based on Chaotic Permutation Multiple Circular Shrinking and Expanding," J. Inf. Hiding Multim. Signal Process., vol. 7, pp. 697-713, 2016.

[21] Y. Suryanto, Suryadi, and K. Ramli, "A new image encryption using color scrambling based on chaotic permutation multiple circular shrinking and expanding," Multimedia Tools and Applications, vol. 76, no. 15, pp. 16831-16854, 2017, doi: 10.1007/s11042-016-3954-5.

[22] K. Ramli, Y. Suryanto, Magfirawaty, and N. Hayati, "Novel Image Encryption Using a Pseudoset Generated by Chaotic Permutation Multicircular Shrinking With a Gradual Deletion of the Input Set," IEEE Access, vol. 8, pp. 110351-110361, 2020, doi: 10.1109/ACCESS.2020.3001949.

[23] N. Hayati, Y. Suryanto, K. Ramli, and M. Suryanegara, "End-to-End Voice Encryption Based on Multiple Circular Chaotic Permutation," in 2019 2nd International Conference on Communication Engineering and Technology (ICCET), 12-15 April 2019 2019, pp. 101-106, doi: 10.1109/ICCET.2019.8726890.

[24] R. Shirey, "Internet Security Glossary, Version 2", FYI 36, RFC 4949, doi: 10.17487/RFC4949, August 2007. Network Working Group [Internet]. Available: https://www.rfc-editor.org/info/rfc4949.