© 2025 IJRTI | Volume 10, Issue 11 November 2025 | ISSN: 2456-3315

A Hybrid Cryptosystem Integrating AES S-Box and Genetic Operations for Enhanced Security

Vishal Kumar Saini & Ram Milan Singh
Department of Mathematics
Institute for Excellence in Higher Education, Bhopal

ABSTRACT

This research outlines a new symmetric key cryptography scheme that employs hybrid method of encryption-decryption process through AES S-Box substitution and genetic operations as intensifying steps toward its security. During the encryption process, the authenticated nonlinearities of the AES S-Box were then added by employing genetic operations such as uniform crossover and inversion mutation for the final diffusion and randomization processes. The decryption process reverses these operations without effort through inverse genetic transformations and inverse S-Box substitution to ensure a perfect recovery of the plaintext. This hybrid method significantly improves key sensitivity, confusion, and diffusion properties throughout the encryption-decryption pipeline. Extensive security and performance analysis demonstrates that the proposed cryptosystem is more superior against linear and differential cryptanalysis and uses less computational resource for application in real-time secure communication.

Key Words: AES S-Box, Genetic Algorithms, Encryption, Decryption, Hybrid Cryptosystem, Uniform Crossover, Inversion Mutation, Symmetric Cryptography.

1. INTRODUCTION

Cryptography provides the foundational principles for securing digital information through mathematical techniques for encryption and decryption. In symmetric key stuff, there's just one key that both sides share for those operations. Makes communication quick and secure [1].

This study examines a novel cryptosystem. It draws on the AES S-Box, the nonlinear component from the Advanced Encryption Standard known for its resistance to various attacks [4]. Evidence indicates that replacing traditional substitution techniques with this S-Box introduces substantial confusion and diffusion properties. These elements appear to enhance protection against analytical assaults.

To bolster security further, the approach incorporates operations from genetic algorithms into the encryption process. Research suggests that uniform crossover in the genetic algorithm effectively blends data segments in an adaptive manner [2,3]. Inversion mutation, applied within specified ranges, alters bits selectively. Such integrations contribute added computational complexity [2,5,6].

The resulting hybrid system merges the reliable robustness of the AES S-Box with the dynamic variability of genetic mechanisms. It yields a symmetric cryptosystem exhibiting improved confusion, diffusion, and key sensitivity. Comprehensive security and performance evaluations confirm its viability for contemporary secure communications [5,6].

2. LITERATURE REVIEW

The Advanced Encryption Standard (AES) Substitution-Box (S-Box) is a basic building block in contemporary symmetric-key cryptography, intended to introduce non-linearity, confusion, and linear and differential cryptanalysis resistance. Its algebraic properties and bijectivity guarantee substantial output variations with small input variations, which strengthens security against statistical attacks. The AES S-Box has been thoroughly tested in cryptographic literature for its efficiency and strength (Daemen & Rijmen,2002). Hybrid methods, which include classical cryptographic primitives and evolutionary methods, have been the subject of recent research. Mittal & Gupta (2019) introduced a symmetric key cryptosystem based on genetic algorithms (GAs), i.e., uniform crossover and inversion mutation, augmented with standard substitution and matrix operations. Their research showed how the operations in GA could improve diffusion and key sensitivity but at the expense of employing linear substitution, which restricted non-linear security features.

To overcome such limitations, Singh & Agarwal (2023) proposed a hybrid model substituting linear substitution with the AES S-Box, keeping genetic mutation operations intact. This hybrid utilized the non-linearity of the S-Box and the adaptive randomness of GAs to produce enhanced confusion and resistance to cryptanalysis. Their work is in line with other efforts toward evolving GA-based cryptography by integrating mathematically sound elements such as the AES S-Box.

This article advances these concepts by combining the AES S-Box with crossover and mutation operations, forming a symmetric cipher that blends the determinism of security with evolutionary randomness.

3. METHODOLOGY

The suggested cryptosystem is more secure with the incorporation of three main operations:

Matrix addition, AES S-Box substitution, and Genetic operations. The operations are carried out in a particular order to yield the final ciphertext.

•Matrix Addition: Plaintext and symmetric key are encoded to numerical matrices of a constant size (e.g., 6×6) through EBCDIC encoding.

•Additive matrix (A) is created by adding the two matrices element-wise to simplify operations and eliminate negative values.

•Substitution Using AES S-Box: One byte from the additive matrix is substituted with its counterpart from the AES S-Box. This non-linear substitution is important in order to introduce confusion and enhance resistance against analytical attacks.

• Genetic Operations: The replaced data is then treated with genetic operations to enhance diffusion and randomness. Initially, the data is divided into two equal halves for uniform crossover. Then, inversion mutation for every 8-bit byte takes place, in which the bits from positions 4th to 7th are inverted.

4. ALGORITHM

Encryption

- Matrix T receives the plaintext transformation into numerical matrix form as its first operation.
- Add Key Matrix: The key matrix K joins with matrix T through addition to produce matrix A.
- S-Box Substitution: Every value inside matrix A undergoes substitution through the AES S-Box operation.
- Binary Conversion: The decimal outputs are converted into 8-bit binary streams.
- Crossover& Mutation: The binary streams experience uniform crossover and inversion mutation processes to generate the ciphertext.

Decryption

- Binary Conversion: The ciphertext requires conversion into 8-bit binary streams as its first step.
- Reverse Operations: The original binary data before mutation becomes accessible through the reversal of inversion mutation and uniform crossover processes.
- Inverse S-Box: The binary data must be converted to decimal before applying the Inverse AES S-Box to restore its original values.
- Subtract Key Matrix: The key matrix K must be subtracted from the matrix to restore the original plaintext matrix T.
- Matrix to Plaintext: The EBCDIC character codes from matrix T will reveal the original plaintext.

Table 1: Aes S-Box

	00	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08

C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 2: Inverse Aes S-Box

	00	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Example

Let block size n=6

Select input key:100000210000221000222100222210222221

This string constitutes the following 6×6 key matrix,

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 2 & 2 & 1 & 0 \\ 2 & 2 & 2 & 2 & 2 & 1 \end{bmatrix}$$

A. Encryption:

i. Assume the plain text is: Math is the language of the universe

ii. Convert the plain text to block of size n, we get their EBCDIC equivalent text matrix as given below

$$T= \begin{bmatrix} 212 & 129 & 163 & 136 & 64 & 137 \\ 162 & 64 & 163 & 136 & 133 & 64 \\ 147 & 129 & 149 & 135 & 164 & 129 \\ 135 & 133 & 64 & 150 & 134 & 64 \\ 163 & 136 & 133 & 64 & 164 & 149 \\ 137 & 165 & 133 & 153 & 162 & 133 \end{bmatrix}$$

iii. Add the key matrix with text matrix, we get below additive matrix:

$$A = \begin{bmatrix} 213 & 129 & 163 & 136 & 64 & 137 \\ 164 & 65 & 163 & 136 & 133 & 64 \\ 149 & 131 & 150 & 135 & 164 & 129 \\ 137 & 135 & 65 & 151 & 134 & 64 \\ 165 & 138 & 135 & 66 & 165 & 149 \\ 139 & 167 & 135 & 155 & 164 & 134 \end{bmatrix}$$

iv. Each byte of the additive matrix A is substituted using the predefined AES S-Box to complete the SubBytes transformation.

v. Application of AES S-Box Substitution

The AES S-Box was applied to transform every decimal number in the additive matrix. Evidence indicates this step introduces essential non-linearity into the encryption. The process involved converting each input byte to hexadecimal first. Then, the standard AES S-Box table was consulted for the corresponding value. That table relies on a straightforward row-column setup. The first hex digit selects the row. The second identifies the column. After obtaining the new hex value, it was converted back to decimal for subsequent operations.

Calculation

- ✓ For input decimal 213(Hex: D5)
  Row=D, Column = 5 → S-Box Output = 03 (Hex) =3(Decimal)
- ✓ For input decimal 129 (Hex: 81):
  Row= 8, Column = 1 → S-Box Output = 0C (Hex)=12(Decimal)

In the same manner, 10, 196, 9, 70, 73, 131, 10, 196, 151, 9, 42, 201, 144, 23, 73, 12, 70, 23, 44, 136, 68, 9, 6, 238, 23, 44, 6, 42, 184, 92, 23, 11, 73, 68

Final values: 3,12,10,196,9,70,73,131,10,196,151,9,42,201,144,23,73,12,70,23,44,136,68,9,6,238,23,44,6,42,184,92,23,11,73,68

These values were then converted to 8-bit binary for further processing in the genetic operations phase.

Convert into binary code:

B1: 00000011 B2: 00001100 B3: 00001010 B4: 11000100 B5: 00001001 B6: 01000110 B7: 01001001 B8: 10000011 B9: 00001010 B10: 11000100 B11: 10010111 B12: 00001001B13: 00101010 B14: 11001001 B15: 10010000 B16: 00010111 B17: 01001001 B18: 00001100 B19: 01000110 B20: 00010111 B21: 00101100 B22: 10001000 B23: 01000100 B24: 00001001 B25: 00000110 B26: 11101110 B27: 00010111 B28: 00101100 B29: 00000110 B30: 00101010 B31: 10111000 B32: 01011100 B33: 00010111 B34: 00001011 B35: 01001001 B36: 01000100

Now above binary streams is divided into two segments, as follows:

00000011 00001100 00001010 11000100 00001001 01000110 01001001 10000011 00001010 11000100 10010111 00001001 00101010 11001001 10010000 00010111 01001001 00001100

01000110 00010111 00101100 10001000 01000100 00001001 00000110 11101110 00010111 00101100 00000110 00101010 10111000 01011100 00010111 00001011 01001001 01000100

Now applying the uniform crossover on the set of 8 bits after each 8 bit, we get

00000011 00010111 00001010 10001000 00001001 00001001 01001001 11101110 00001010 00101100 10010111 00101010 00101010 01011100 10010000 00001011 01001001 01000100 01000110 00001100 00101100 11000100 01000100 01000110 00000110 10000011 00010111 11000100 00000110 00001001 10111000 11001001 00010111 00010111 01001001 00001100

For the mutation operation, here we use the inversion mutation technique on each 8-bit group from the 4th bit to the 7th bit,

00010001 00011011 00010100 10000100 00000101 00000101 01000101 11111100 00010100 00101100 10011011 00110100 00110100 01001110 10000010 00010101 01000101 01001000 01011000 00001100 00101100 11001000 01001000 01011000 00011000 10010001 00011011 11001000 00011000 00000101 10100110 11000101 00011011 00011011 01000101 00001100

Now divide the above binary bits into the set of 8 bits and convert them into their equivalent hexadecimal, which is the final cipher as follows:

11,1B,14,84,05,05,45,FC,14,2C,9B,34,34,4E,82,15,45,48,58,0C,2C,C8,48,58,18,91,1B,C8,18,05,A6,C5,1B,1B,45,0C

Now this cipher text (111B1484050545FC142C9B34344E82154548580C2CC8485818911BC81805A6C51B1B450C) is send to the receiver for decryption along with the key 100000210000221000222100222210222221CM.

Here user input 10000021000022100022210022221022221, block size is 6, C is the uniform crossover applied on the set of 8 bits after each 8 bit and M is the inversion mutation on each 8 bit set from 4th bit to 7th bit.

B. Decryption:

i. Consider the cipher text 111B1484050545FC142C9B34344E82154548580C2CC8485818911BC81805A6C51B1B450C

ii. Convert it into 8 bit binary equivalent and divide the binary streams into two segments, we get
00010001 00011011 00010100 10000100 00000101 00000101 01000101 11111100 00010100 00101100 10011011 00110100 00110100 01001110 10000010 00010101 01000101 01001000 01011000 00001100 00101100 11001000 01001000 01011000 00011000 10010001 00011011 11001000 00011000 00000101 10100110 11000101 00011011 00011011 01000101 00001100

iii. Apply inversion mutation operation on each 8 bit group from 4th bit to 7th bit, we get
00000011 00010111 00001010 10001000 00001001 00001001 01001001 11101110 00001010 00101100 10010111 00101010 00101010 01011100 10010000 00001011 01001001 01000100 01000110 00001100 00101100 11000100 01000100 01000110 00000110 10000011 00010111 11000100 00000110 00001001 10111000 11001001 00010111 00010111 01001001 00001100

iv. Applying the uniform crossover on the set of 8 bits after each 8 bit, we get
00000011 00001100 00001010 11000100 00001001 01000110 01001001 10000011 00001010 11000100 10010111 00001001 00101010 11001001 10010000 00010111 01001001 00001100 01000110 00010111 00101100 10001000 01000100 00001001 00000110 11101110 00010111 00101100 00000110 00101010 10111000 01011100 00010111 00001011 01001001 01000100

v. Now converting the above into 8 bit group i.e.
B1: 00000011 B2: 00001100 B3: 00001010 B4: 11000100 B5: 00001001 B6: 01000110 B7: 01001001 B8: 10000011 B9: 00001010 B10: 11000100 B11: 10010111 B12: 00001001B13: 00101010 B14: 11001001 B15: 10010000 B16: 00010111 B17: 01001001 B18: 00001100 B19: 01000110 B20: 00010111 B21: 00101100 B22: 10001000 B23: 01000100 B24: 00001001 B25: 00000110 B26: 11101110 B27: 00010111 B28: 00101100 B29: 00000110 B30: 00101010 B31: 10111000 B32: 01011100 B33: 00010111 B34: 00001011 B35: 01001001 B36: 01000100

vi. Write their EBCDIC decimal equivalent, we get
3,12,10,196,9,70,73,131,10,196,151,9,42,201,144,23,73,12,70,23,44,136, 68,9,6,238,23,44,6,42,184,92,23,11,73,68

vii. The inverse S-Box (InvSubBytes) was applied by mapping each substituted byte to its original value using the AES inverse S-Box table.

✓ For input decimal 3 (Hex: 03):
Row = 0, Column = 3 → Inverse S-Box Output = D5 (Hex) = 213 (Decimal)

✓ For input decimal 12 (Hex: 0C):
Row = 0, Column = C → Inverse S-Box Output = 81 (Hex) = 129 (Decimal)

In the same manner,163, 136, 64, 137, 164, 65, 163, 136, 133, 64, 149, 131, 150, 135, 164, 129, 137, 135, 66, 151, 134, 64, 165, 138, 135, 66, 165, 149, 139, 167, 135, 155, 164, 134.

We move on the following matrix of 6×6 order as follows:

$$A = \begin{bmatrix} 213 & 129 & 163 & 136 & 64 & 137 \\ 164 & 65 & 163 & 136 & 133 & 64 \\ 149 & 131 & 150 & 135 & 164 & 129 \\ 137 & 135 & 65 & 151 & 134 & 64 \\ 165 & 138 & 135 & 66 & 165 & 149 \\ 139 & 167 & 135 & 155 & 164 & 134 \end{bmatrix}$$

viii. Subtract the key matrix K from A, we get

$$T = \begin{bmatrix} 212 & 163 & 163 & 136 & 64 & 137 \\ 162 & 64 & 163 & 136 & 133 & 64 \\ 147 & 129 & 149 & 135 & 164 & 129 \\ 135 & 133 & 64 & 150 & 134 & 64 \\ 163 & 136 & 133 & 64 & 164 & 149 \\ 137 & 165 & 133 & 153 & 162 & 133 \end{bmatrix}$$

ix. The EBCDIC character code equivalent of this matrix gives the original plain text i.e Math is the language of the universe

5. RESULT AND DISCUSSION:

The proposed cryptosystem worked well in encrypting and decrypting a sample plaintext. It used a hybrid method that mixes AES S-Box substitution with genetic operations. The SubBytes step added strong non-linearity and confusion to the process. Uniform crossover and inversion mutation helped spread things out and make the ciphertext more random. When the ciphertext got sent over with the key and parameters, decryption went smoothly. It applied inverse genetic operations first. Then came the InvSubBytes transformation. This rebuilt the original plaintext exactly. The text was "Math is the language of the universe". These outcomes show how blending AES S-Box with genetic algorithms boosts key sensitivity. It stands up against linear and differential cryptanalysis too. Overall, it creates a solid, efficient way for symmetric key encryption.

6. CONCLUSION:

This work introduced a hybrid symmetric key cryptosystem. It pairs AES S-Box substitution with genetic operations like uniform crossover and inversion mutation. Experiments confirmed the approach. They showed perfect plaintext recovery after decryption. This proves the system's correctness and reversibility. The AES S-Box brings non-linearity. Genetic algorithms add randomness. Together, they deliver high confusion and diffusion. The setup resists typical cryptanalytic attacks. It provides an efficient, secure option for today's symmetric encryption needs. Future tweaks could improve performance and scalability. That would suit large-scale data communication better.

REFERENCES

1. Douglas, R. Stinson: “Cryptography – Theory and Practice “, CRC Press, 1995
2. Dutta Suvajit, Das Tanumay, Jash Sharad, Patra Debasish, Paul Pranam: A Cryptography Algorithm Using the Operations of Genetic Algorithm & Pseudo Random Sequence Generating Functions, International Journal of Advances in Computer Science and Technology, Volume 3, No.5, May 2014, pp. 325-330.
3. Mitchell M.: “An Introduction to Genetic Algorithms,” The MIT Press, Cambridge, USA, 1999.
4. Daemen, J., & Rijmen, V. (2002). The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag
5. Singh, R. M., & Agarwal, S. (2023). A Hybrid Cryptographic Model Using AES S-Box and Genetic Mutation for Enhanced Security. International Journal of Computer Network and Information Security, 15(2), 45-58.
6. Mittal, K., & Gupta, P. (2019). A Symmetric Key Cryptosystem Using Genetic Operations and Matrix Manipulation. Journal of Discrete Mathematical Sciences and Cryptography, 22(4), 649-663.
7. S. Kumar, A.K Gupta and R. Saxena, “A Novel Design of AES S-Box using circular Shift and XOR Operations,” Journal of Discrete Mathematical Sciences and Cryptography.
8. Bhasin Harsha, Kumar Ramesh, Kathuria Neha: “Cryptography using Cellular Automata”. International Journal of Computer Science and Information Technology, Vol. 4(2), 355-357, 2013.
9. Alkhamery, R., Alahmadi, Y., Alsorori, M., & Alassali, S. (2021). Enhance Security of AES Algorithm Based on S-Box. International Journal of Computer Sciences and Engineering, *9*(2), 39-45. https://doi.org/10.26438/ijcse/v912.3945.