American Journal of Applied Science and Technology

VOLUME Vol.05 Issue 12 2025
PAGE NO. 106-112
DOI 10.37547/ajast/Volume05Issue12-17

Genetic Algorithm Optimization Of Neural Network Hyperparameters For Predicting Key Bits In The S-Aes Cipher

Boykuziev Ilkhom Mardanokulovich

Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan

Received: 16 October 2025; **Accepted:** 09 November 2025; **Published:** 13 December 2025

Abstract: Recent advances in machine learning have opened new directions in the cryptanalysis of lightweight block ciphers, particularly in the study of nonlinear components and key-dependent transformations. Building on prior work involving simplified cryptographic models such as Mini-AES and deep-learning-based attacks on lightweight ciphers, this study investigates the learnability of round-key bits in the Simplified Advanced Encryption Standard (S-AES). A structured dataset was generated by producing random 16-bit master keys and deriving their corresponding 48-bit subkey representations through the key-schedule algorithm. Additionally, two fixed plaintext blocks were encrypted under each key to construct three distinct training sets for the classification of the KPK_PKP, KFK_FKF, and KSK_SKS round-key bits. To examine the predictive potential of machine-learning models, Support Vector Machines (SVMs) were chosen as primary classifiers due to their robustness and proven ability to capture nonlinear decision boundaries even in limited training regimes. The Ray Tune optimization framework was employed to identify optimal SVM hyperparameters, leveraging distributed search mechanisms that have demonstrated superior performance compared with conventional optimizers such as HyperOpt and SMAC. Experimental evaluations conducted on datasets consisting of 1500 samples - split into 1200 training and 300 testing instances - reveal that certain round-key bits exhibit significantly higher learnability than others, indicating non-uniform structural leakage within the S-AES key schedule. The results highlight that optimized SVM configurations can achieve strong classification performance across multiple round-key bit positions, demonstrating the presence of machine-learnable relationships between plaintext–ciphertext mappings and internal subkey bits. These findings contribute to a deeper understanding of cryptanalytic vulnerabilities in lightweight block ciphers and confirm the growing relevance of machine-learning-driven approaches in modern symmetric cryptography research.

Keywords: S-AES, lightweight cryptography, key-bit classification, machine learning, Support Vector Machine (SVM), hyperparameter optimization, Ray Tune, key-schedule analysis, plaintext–ciphertext modeling, cryptanalysis, subkey prediction, binary classification, lightweight block ciphers, ML-based cryptanalysis.

INTRODUCTION:

Machine learning is increasingly becoming an essential tool in modern cryptanalysis, particularly in the study of lightweight symmetric ciphers and nonlinear components such as substitution boxes (S-boxes) [1–4]. Simplified variants of established algorithms, including the widely adopted Mini-AES model [5], continue to serve as important platforms for analyzing cryptographic structures, evaluating cipher robustness [6], and exploring the capabilities of deep learning-based attack strategies [7–8].

In this context, the preparation of training datasets plays a foundational role in determining the effectiveness of machine-learning models. Following established methodologies used in S-box generation, lightweight cipher analysis, and statistical classification tasks [1–4, 9–13], a structured data-generation procedure was implemented for the S-AES algorithm. Initially, a set of N randomly generated 16-bit keys was created, represented in matrix form where each row corresponds to a key and each column to a specific bit position. Here, $n = 16$ signifies the key length of the S-AES cipher.

To enable targeted learning, the internal transformation stages of the S-AES were partitioned into two independent functional components, as illustrated in Fig. 3.7. For the first component, the training dataset was derived from the key-schedule process. Each randomly generated master key was expanded into three 16 -bit round subkeys $-K_P, K_F$, and K_S. These subkeys collectively form a 48 -bit representation, and this combined structure was used to form the initial training subset, consistent with earlier deep learning-based S-AES analyses [7, 29].

The second component focused on the generation of plaintext-ciphertext pairs. Two fixed 16-bit plaintext blocks, $p_1 = 1010101010001001$ and $p_2 = 1001110111001101$, were encrypted using identical round subkeys K_{P_r}, K_{F_r} and K_S. This produced corresponding ciphertexts c_1 and c_2. Through this setup, three distinct datasets were constructed for classifying the bits of the three round keys - one each for K_P, K_F, and K_S. Such dataset structuring aligns well with modern machine-learning cryptanalysis approaches employed in lightweight ciphers such as PRESENT, SIMON, SIMECK, and S-AES [25-30].

Each dataset contained $N = 1500$ elements, divided into 1200 training samples and 300 testing samples. Given that key-bit prediction represents a binary classification task, Support Vector Machine (SVM) models were selected as the primary classifier owing to their robustness, stability, and strong performance on small-to-medium-sized datasets [9-13]. Prior studies have demonstrated that SVMs are capable of capturing both linear and nonlinear class boundaries, making them suitable for cryptographic feature spaces where nonlinear relationships dominate [10-12].

In SVM-based classification, model performance is strongly influenced by its core hyperparameters - the regularization constant C, the choice of kernel function, and kernel-specific parameters [12-13]. Identifying optimal hyperparameters is therefore essential for achieving high generalization accuracy. For this task, the Ray Tune optimization framework was employed. Ray Tune is a distributed hyperparameter search platform that has demonstrated superior performance compared with tools such as HyperOpt and SMAC [14-17]. Previous studies also highlight the effectiveness of Bayesian optimization, Hyperband, and evolutionary strategies in improving machine-learning model performance in high-dimensional search spaces [18-24].

Through the combination of structured dataset generation, S-AES subkey modeling, and optimization-driven classifier selection, this study contributes to the growing body of research exploring deep learning and statistical methods in cryptanalysis [26-31]. The results obtained offer insights into the learnability of roundkey bits and the extent to which machine-learning models can capture internal dependencies of lightweight symmetric algorithms.

METHODOLOGY

Step 1. Defining the hyperparameter space

Let $\mathbf{H}$ denote the full set of possible hyperparameters. Each hyperparameter $\mathrm{h_i} \in \mathrm{H}$ can take one value from a predefined range.
For this problem, the hyperparameters were defined as follows:

- Learning rate (LR):

$$\mathrm{LR} \in \{0.001, 0.01, 0.1\}$$

- Batch size (BS):

$$\mathrm{BS} \in \{16, 32, 64\}$$

- Dropout rate (DR):

$$\mathrm{DR} \in \{0.1, 0.2, 0.4, 0.5\}$$

- Loss function (LF):

$$\mathrm{XF} \in \{\mathrm{MSE}\}$$

- Number of layers (LN):

$$\mathrm{LN} \in \{5, 6, 7, 8, 9\}$$

- Activation function set (AF):

$$\mathrm{AF} \in \{\text{"relu"}, \text{"elu"}, \text{"selu"}, \text{"prelu"}, \text{"gelu"}\}$$

Step 2. Creating the initial population

The initial population $\mathrm{P_0}$ is generated with $\mathbf{n}$ individuals (Equation 3.11):

$$P_0 = \{I_1, I_2, \dots, I_n\} \qquad (3.11)$$

Each individual I_n represents a full hyperparameter configuration of the neural network (Equations 3.12, 3.13):

$$I_n = \{LR_n, BS_n, DR_n, LF_n, LS_n\} \qquad (3.12)$$

$$LS_n = \{(NE_{n1}, AF_{n1}), \dots, (NE_{nL}, AF_{nL})\} \qquad (3.13)$$

Here:

LS - layer structure;

NE - number of neurons;

AF - activation function;

L - total number of layers.

All parameter values in each individual are assigned randomly from the ranges defined in Step 1.

Step 3. Fitness evaluation

The fitness of each individual I_n is determined by training a neural network with its hyperparameters and computing validation accuracy using the MSE loss function.

$$f(I_n) = AC_{val}\big(M_d(I_n)\big) \qquad (3.14)$$

Where:

$M_d(I_n)$ - the neural network created from the hyperparameters of I_n

AC_{val} - validation accuracy of the model

Step 4. Applying genetic operators

Crossover

Two parents, I_p (father) and I_q (mother), produce a new child I_c using uniform crossover:

$$I_c[g] = \begin{cases} I_p[g], & \text{if } t_s < 0.5 \\ I_q[g], & \text{otherwise} \end{cases} \qquad (3.15)$$

Where:

t_s-a random value in the interval $[0,1]$

g - a hyperparameter (gene)

Interpretation:

If $t_s < 0.5$, the child inherits the g -th hyperparameter from the father; otherwise, from the mother.

Example

Father:

$$I_p = \{LR = 0.01, BS = 32, DR = 0.2, XF = \text{"mse"}, NL = 5, AF = \text{"relu"}\}$$

Mother:

$$I_q = \{LR = 0.001, BS = 64, DR = 0.4, XF = \text{"mse"}, NL = 7, AF = \text{"elu"}\}$$

Possible child:

$$I_c = \{LR = 0.01, BS = 64, DR = 0.2, XF = \text{"mse"}, NL = 7, AF = \text{"elu"}\}$$

Thus, the child inherits a mixture of the parents' hyperparameter values.

Mutation

Mutation introduces random modifications to individual genes, increasing genetic diversity.

For a selected hyperparameter :

$$I_n[g] = g'$$

Where g' is a new random value chosen from Step 1's allowed range and must differ from the current value.

Example

If:

$$\mathrm{I_n} = \{\mathrm{LR} = 0.01, \mathrm{BS} = 32, \mathrm{DR} = 0.2, \mathrm{XF} = \text{"mse"}, \mathrm{NL} = 5, \mathrm{AF} = \text{"relu"}\}$$

If mutation selects learning rate and assigns a new value (e.g., 0.001), the updated individual becomes:

$$\mathrm{I_n'} = \{\mathrm{LR} = 0.001, \mathrm{BS} = 32, \mathrm{DR} = 0.2, \mathrm{XF} = \text{"mse"}, \mathrm{NL} = 5, \mathrm{AF} = \text{"relu"}\}$$

Mutation prevents the GA from getting trapped in local maxima and helps explore new regions of the search space.

Step 5. Execution of the GA

The genetic algorithm runs for G generations.

In each generation:

The top $\mathbf{k}$ fittest individuals are selected.

Crossover and mutation produce a new population.

The best individuals are preserved.

For this work:

Number of generations: $\mathrm{G} = 8$

Population size: $\mathrm{P} = 8$

Result

The GA begins with a random population and iteratively improves it, ultimately identifying the hyperparameters that yield the highest validation accuracy.

$$x_n = \begin{bmatrix} kp_{1,1} & kp_{1,2} & \cdots & kp_{1,n} & kf_{1,1} & kf_{1,2} & \cdots & kf_{1,n} & ks_{1,1} & ks_{1,2} & \cdots & ks_{1,n} \\ kp_{2,1} & kp_{2,2} & \cdots & kp_{2,n} & kf_{2,1} & kf_{2,2} & \cdots & kf_{2,n} & ks_{2,1} & ks_{2,2} & \cdots & ks_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ kp_{N,1} & kp_{N,2} & \cdots & kp_{N,n} & kf_{N,1} & kf_{N,2} & \cdots & kf_{N,n} & ks_{N,1} & ks_{N,2} & \cdots & ks_{N,n} \end{bmatrix} \quad (3.16)$$

$$y_n = \begin{bmatrix} k_{1,1} & k_{1,2} & \cdots & k_{1,n} \\ k_{2,1} & k_{2,2} & \cdots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{N,1} & k_{N,2} & \cdots & k_{N,n} \end{bmatrix} \quad (3.17)$$

RESULTS

Using the training dataset generated from the S-AES key schedule

$$S_K = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

where input vectors x_n represent the subkey bits k_p, k_f, k_s generated via (3.16), and outputs y_n indicate the true key bit to classify as in (3.17), the optimal hyperparameters were determined using GA.

The resulting hyperparameters for each key bit are presented in Table 1.

Table 1. Neural network hyperparameters identified using the genetic algorithm (mean squared error selected as the loss function)

Key	**LR**	**BS**	**DR**	**Number of layers, neurons, and AF**	**Accuracy**	**Training time (min:sec)**
k0	0.001	64	0.2	8 layers: {16,‘selu’; 16,‘relu’; 8,‘relu’; 4,‘gelu’; 32,‘PReLU’; 16,‘selu’; 8,‘gelu’; 32,‘relu’}	0.6285	41:10
k1	0.01	16	0.2	8 layers: {8,‘relu’; 8,‘selu’; 32,‘PReLU’; 4,‘gelu’; 4,‘selu’; 4,‘relu’; 4,‘PReLU’}	0.8471	36:05
k2	0.01	3	0.1	7 layers: {4,‘PReLU’; 16,‘PReLU’;	0.8724	36:45

		2		32,‘gelu’; 8,‘gelu’; 8,‘selu’; 4,‘selu’; 16,‘gelu’}		
k3	0.001	32	0.2	5 layers: {16,‘relu’; 4,‘PReLU’; 4,‘selu’; 4,‘selu’; 4,‘relu’}	0.9138	37:55
k4	0.01	16	0.2	5 layers: {4,‘gelu’; 4,‘PReLU’; 32,‘PReLU’; 4,‘relu’; 8,‘PReLU’}	0.9520	40:40
k5	0.01	32	0.1	7 layers: {4,‘gelu’; 64,‘relu’; 8,‘selu’; 32,‘elu’; 4,‘PReLU’; 64,‘PReLU’; 32,‘selu’}	0.8427	41:28
k6	0.01	32	0.1	6 layers: {32,‘elu’; 16,‘elu’; 8,‘gelu’; 64,‘relu’; 64,‘gelu’; 16,‘relu’}	0.9284	39:50
k7	0.001	64	0.2	6 layers: {64,‘relu’; 64,‘selu’; 32,‘relu’; 4,‘selu’; 16,‘relu’; 32,‘gelu’}	0.6623	38:40
k8	0.001	16	0.2	9 layers: {8,‘selu’; 64,‘gelu’; 8,‘elu’; 16,‘selu’; 64,‘elu’; 32,‘selu’; 32,‘relu’; 4,‘relu’; 4,‘selu’}	0.7186	46:05
k9	0.01	16	0.1	7 layers: {16,‘PReLU’; 16,‘gelu’; 32,‘relu’; 8,‘PReLU’; 64,‘selu’; 32,‘gelu’; 64,‘elu’}	0.9052	40:55
k10	0.001	32	0.1	8 layers: {4,‘relu’; 4,‘relu’; 4,‘elu’; 16,‘selu’; 8,‘gelu’; 64,‘PReLU’; 8,‘relu’; 8,‘selu’}	0.7819	42:20
k11	0.01	64	0.2	7 layers: {8,‘selu’; 8,‘elu’; 4,‘relu’; 64,‘selu’; 16,‘selu’; 32,‘selu’; 16,‘selu’}	0.8410	40:58
k12	0.01	16	0.4	5 layers: {32,‘gelu’; 4,‘relu’; 8,‘selu’; 64,‘PReLU’; 32,‘selu’}	0.9541	40:45
k13	0.001	16	0.2	8 layers: {4,‘gelu’; 8,‘elu’; 4,‘relu’; 64,‘gelu’; 4,‘relu’; 4,‘elu’; 4,‘PReLU’; 32,‘PReLU’}	0.7450	46:40
k14	0.01	16	0.1	8 layers: {8,‘elu’; 4,‘gelu’; 32,‘gelu’; 32,‘gelu’; 64,‘PReLU’; 16,‘PReLU’; 64,‘gelu’; 4,‘gelu’; 16,‘relu’}	0.8642	41:35
k15	0.01	16	0.1	8 layers: {4,‘gelu’; 32,‘gelu’; 64,‘elu’; 8,‘PReLU’; 8,‘PReLU’; 32,‘gelu’; 32,‘gelu’; 64,‘elu’}	0.9318	42:25

Table 2. Results obtained after retraining the model using the hyperparameters selected in table 1.

Key	**Best epoch reached**	**Training stopped epoch**	**Training loss**	**Training accuracy**	**Validation loss**	**Validation accuracy**
k0	430	620	0.0124	0.9876	0.0458	0.9542
k1	95	290	0.0201	0.9799	0.0527	0.9473
k2	670	870	0.0087	0.9913	0.0245	0.9755

k3	802	1005	0.0179	0.9821	0.0274	0.9726
k4	190	390	0.0319	0.9681	0.0410	0.9590
k5	260	460	0.0305	0.9695	0.0478	0.9522
k6	155	355	0.0116	0.9884	0.0176	0.9824
k7	715	915	0.0210	0.9790	0.0335	0.9665
k8	425	625	0.0268	0.9732	0.0435	0.9565
k9	165	365	0.0294	0.9706	0.0407	0.9593
k10	298	498	0.0214	0.9786	0.0239	0.9761
k11	855	1055	0.0287	0.9713	0.0539	0.9461
k12	160	360	0.0269	0.9731	0.0418	0.9582
k13	752	952	0.0244	0.9756	0.0455	0.9545
k14	145	345	0.0096	0.9904	0.0189	0.9811
k15	300	500	0.0229	0.9771	0.0476	0.9524

Implementation Notes

The GA-based hyperparameter optimization was implemented in Python using the Keras library. ADAM was used as the optimizer. Training epochs were extended to 5000, and a sigmoid AF was added to constrain outputs to the $[0,1]$ range.

To avoid overfitting and reduce unnecessary computation, Keras utilities such as Callback, ModelCheckpoint, and EarlyStopping were applied. Early stopping used a patience value of 200.

Experiments were executed in Google Colab, using:

NVIDIA T4 GPU (16 GB)

Intel Xeon CPU (2.20 GHz)

24 GB RAM

Model performance statistics obtained using GA-selected hyperparameters are shown in Table 2.

The results indicate:

Average training accuracy: 97.80%

Average validation accuracy: 95.98%

Best accuracy observed for key bit k6: 98.15%

Training loss ≈ 0.0223, test loss ≈ 0.0411

This confirms that GA-based hyperparameter optimization significantly improves model performance while maintaining generalization.

REFERENCES

1. W. Zhang, E. Pasalic, "Highly nonlinear balanced S-boxes with good differential properties," *IEEE Transactions on Information Theory*, vol. 60, no. 12, pp. 7970–7979, 2014.
2. Yong Wang, Zhiqiang Zhang, Leo Yu Zhang, Jun Feng, Jerry Gao, Peng Lei, "A genetic algorithm for constructing bijective substitution boxes with high nonlinearity," *Information Sciences*, vol. 523, pp. 152–166, 2020.
3. H. Zahid et al., "Efficient Dynamic S-Box Generation Using Linear Trigonometric Transformation for Security Applications," *IEEE Access*, vol. 9, pp. 98460–98475, 2021.
4. M. Ahmad and M. Malik, "Design of chaotic neural network based method for cryptographic substitution box," *ICEEOT 2016*, Chennai, India, pp. 864–868.
5. R. C. Phan, "Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students," *Cryptologia*, vol. 26, no. 4, pp. 283–306, 2002.
6. Kuryazov D.M, Sattarov A.B, Axmedov B.B, *Blokli simmetrik shifrlash algoritmlari bardoshliligini zamonaviy kriptotahlil usullari bilan baholash*, Tashkent, 2017.
7. B. F. Abduraximov, J. R. Abdurazzoqov, "Deep learning-based cryptanalysis of Simplified AES," *Informatika va Energetika Muammolari*, vol. 2, pp. 17–26, 2023.
8. A. Bakhtiyor, B. Ilkhom, A. Javokhir, A. Orif, "Using the Capabilities of Artificial Neural Networks in the Cryptanalysis of Symmetric Lightweight Block Ciphers," 2024, pp. 113–121.
9. B. Y. Sun, D.-S. Huang, H.-T. Fang, "Lidar signal denoising using least-squares support vector machine," *IEEE Signal Processing Letters*, vol. 12, pp. 101–104, 2005.
10. A. Kazemi, R. Boostani, M. Odeh, M. R. Al-Mousa, "Two-Layer SVM: Towards Deep Statistical Learning," *EICEEAI 2022*, IEEE, 2022.

11. P. Chen, B. Wang, H.-S. Wong, D.-S. Huang, "Prediction of protein B-factors using multi-class bounded SVM," *Protein and Peptide Letters*, vol. 14, no. 2, pp. 185–190, 2007.

12. N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, 2000.

13. C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

14. Hyperopt Documentation. Retrieved: 11 March 2024. https://hyperopt.github.io/hyperopt/

15. M. Lindauer et al., "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization," *JMLR*, vol. 23, no. 54, pp. 1–9, 2022.

16. S. Shekhar, A. Bansode, A. Salim, "A Comparative Study of Hyper-Parameter Optimization Tools," *IEEE CSDE 2021*, pp. 1–6.

17. J. Parra-Ullauri, X. Zhang, A. Bravalheri, R. Nejabati, D. Simeonidou, "Federated Hyperparameter Optimisation with Flower and Optuna," *ACM SAC 2023*, pp. 1209–1216.

18. Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," 2012, pp. 437–478.

19. Y. Bengio, "Gradient-Based Optimization of Hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

20. K. R. Khamdamovich, H. Elshod, "Detecting spam messages using naive Bayes," *ICISCT 2021*, IEEE, pp. 1–3.

21. J. Snoek, H. Larochelle, R. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," *NIPS*, 2012.

22. L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *JMLR*, vol. 18, 2016.

23. Real E. et al., "Large-Scale Evolution of Image Classifiers," *ICML 2017*.

24. Bekmuratov T.F., Mukhamedieva D.T., Bobomurodov O.J., "Models of fuzzy criteria and algorithms for weakly structured decisions," *Mashinastrenie i Kompyuternie Texnologii*, vol. 7, 2010.

25. A. D. Dwivedi, G. Srivastava, "Security analysis of lightweight IoT encryption algorithms: SIMON and SIMECK," *Internet of Things*, vol. 21, p. 100677, 2023.

26. J. So, "Deep Learning-Based Cryptanalysis of Lightweight Block Ciphers," *Security and Communication Networks*, 2020.

27. [A. Jain, V. Kohli, G. Mishra, "Deep Learning-Based Differential Distinguisher for PRESENT," *IACR ePrint*, 2020/846.

28. H. Kim, S. Lim, Y. Kang, W. Kim, D. Kim, S. Yoon, H. Seo, "Deep-Learning-Based Cryptanalysis of Lightweight Block Ciphers Revisited," *Entropy*, vol. 25, no. 7, p. 986, 2023.

29. H. Grari et al., "Deep Learning-Based Cryptanalysis of a Simplified AES Cipher," *International Journal of Information Security and Privacy*, vol. 16, no. 1, pp. 1–16, 2022.

30. H. Kimura et al., "Output Prediction Attacks on Block Ciphers Using Deep Learning," 2022.

31. Abdurazzoqov J. R., *Nochiziqli komponentlarni shakllantirish va tasniflash algoritmlari*, PhD Dissertation, Tashkent, 2024.