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
salyut2017@gmail.com

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.
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.

Methodology

Step 1. Defining the hyperparameter space

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

- Learning rate (LR):

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

- Batch size (BS):

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

- Dropout rate (DR):

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

- Loss function (LF):

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

- Number of layers (LN):

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

- Activation function set (AF):

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

Step 2. Creating the initial population

The initial population P_0 is generated with **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.

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

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.

Step 5. Execution of the GA

The genetic algorithm runs for G generations.

In each generation:

The top **k** fittest individuals are selected.

Crossover and mutation produce a new population.

The best individuals are preserved.

For this work:

Number of generations: $G = 8$

Population size: $P = 8$

Results

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

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.