IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 9, 2021, accepted May 25, 2021, date of publication June 3, 2021, date of current version June 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3085861

Generating Cryptographic S-Boxes Using the Reinforcement Learning

GIYOON KIM[1], HANGI KIM[1], YEACHAN HEO[2], YONGJIN JEON[1], AND JONGSUNG KIM[1,3]

[1]Department of Mathematics and Financial Information Security, Kookmin University, Seoul 02707, South Korea
[2]Gyeongsang National University High School (GNUHS), Jinju 52828, South Korea
[3]Department of Information Security, Cryptology and Mathematics, Kookmin University, Seoul 02707, South Korea

Corresponding author: Jongsung Kim (jskim@kookmin.ac.kr)

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC).

ABSTRACT Substitution boxes (S-boxes) are essential components of many cryptographic primitives. The Dijkstra algorithm, SAT solvers, and heuristic methods have been used to find bitsliced implementations of S-boxes. However, it is difficult to apply these methods for 8-bit S-boxes because of their size. Therefore, to implement these S-boxes so that the countermeasure of side-channel attack can be applied efficiently, using structures such as Feistel, Lai-Massey, and MISTY that can be bitsliced implemented with a small number of nonlinear operations has been widely used. Since S-boxes constructed with structures consist of small S-boxes and have specific designs, there are limitations to their cryptographic security and efficiency. In this paper, we propose a new method for generating S-boxes by stacking bitwise operations from the identity function, an approach that is different from existing methods. This method can be expressed in Markov decision process, and reinforcement learning is a suitable solver for Markov decision process. Our goal is to train this method to an agent through reinforcement learning to generate S-boxes to which the masking scheme, which is a countermeasure of side-channel attack, can be efficiently applied. In particular, our method provided various S-boxes superior or comparable to existing S-boxes. We produced 8-bit S-boxes with differential uniformity 16 (resp. 32) and linearity 128 (resp. 128), generated with nine (resp. eight) nonlinear operations, for the first time. To our best knowledge, this is the first study to construct cryptographic S-Box by incorporating reinforcement learning.

INDEX TERMS S-box, masking efficiency, reinforcement learning, bitsliced implementation, linearity, differential uniformity.

I. INTRODUCTION

To apply security applications to mobile and embedded platforms, lightweight and efficient cryptographic primitives are required. Substitution boxes (S-boxes) are representative nonlinear functions, giving cryptographic primitives Shannon's confusion property [1]. Therefore, finding S-boxes with sufficient security and efficiency is an important issue for the designers of cryptographic algorithms.

Side-channel attacks, first published by Kocher in 1996 [2], can draw secret information of cryptographic algorithms from side-channel leakages such as electromagnetic emissions and power consumption. Since mathematical cryptanalysis alone cannot guarantee the security of cryptographic primitives against side-channel attacks, various counter measures have been proposed. Techniques to randomize the intermediate values of ciphers are widely used, and among them, a higher-order Boolean masking technique is the most popular approach.

The method of implementing an S-box as Table Look Up (TLU) requires table storage space, and especially when applying the masking scheme, even more flip-flops are required. For this reason, in resource-constrained environments, it is common to implement the logic to build the S-box. When applying a higher-order Boolean masking scheme to an S-box, the number of nonlinear operations required for its implementation greatly affects the efficiency of the scheme. This is because, when implementing a higher-order Boolean

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

masking scheme, nonlinear operations have quadratic complexity, $O(d^2)$, whereas linear operations have linear complexity, $O(d)$, where d is the number of operations used. Since most of the nonlinear operations required to implement ciphers are used in S-box implementations [3]–[6], it is important to find S-boxes that have sufficient cryptographic security and can be implemented with a small number of nonlinear operations. There are limitations to the cryptographic security provided by S-boxes depending on the number of their nonlinear operations [7]–[9]. For this reason, studies on secure S-boxes that can be implemented with few nonlinear operations have been conducted [10]–[12]. We generalized this problem to generating S-box by stacking operations, and formulated this as Markov Decision Process (MDP). Deep reinforcement learning can be a good solution as a solver for MDP with high complexity and non-linearity. In this work, we focus on methods of generating cryptographic S-boxes with efficient side-channel masking implementations using reinforcement learning.

A. RELATED WORK

Exhaustive search, algebraic construction, and heuristic generation have been three popular methods of generating S-boxes. Since an exhaustive search requires the analysis of $(2^n)!$ S-boxes for n-bit S-boxes, it is very difficult to find large-sized cryptographic S-boxes. Algebraic construction is a widely used approach to constructing cryptographically secure S-boxes [4], [13]–[18]. In particular, there are studies to create a secure and bijective S-box through Cubic Polynomial Mapping and Cubic Fractional Transformation [19], [20].

Heuristic generation methods include gradient descent, genetic algorithms, and hill-climbing [21]–[23], and the application of these algorithms to neural networks [24].

In recent years, security has become ever more important, even in a lightweight environment in which it is burdensome to store S-boxes. Therefore, in the S-box design process, the efficiency of S-box implementation must be considered.

Bitslicing is an implementation technique that can parallelize bitwise operations in processors. It can work with registers larger than one bit and has proven to be very efficient for many S-boxes [25]–[27]. In the case of bijective 4-bit S-boxes, several studies that provide cryptanalytic results of all 16! of them through exhaustive search [28], [29]. Using tools such as Lighter [30], [31], one can find an efficient bitsliced implementation for a specific 4-bit S-box. Several lightweight block ciphers provide bitsliced implementations with minimal use of nonlinear operations [3], [27], [32]. However, no method has been proposed to find efficient bitsliced implementations for S-boxes with a size of more than 5-bit [33]. Stoffelen proposed a method to obtain an efficient S-box implementation using the SAT solver meeting various criteria, such as the number of nonlinear operations and the size of the gate, but is difficult to apply to large-sized S-boxes such as 8-bit S-boxes [34].

Even if an 8-bit S-box with a desired cryptographic security is selected, finding its efficient bitsliced implementation is challenging. Although many studies have investigated the efficient implementations of the Advanced Encryption Standard (AES), still more than 35 nonlinear operations are required to implement an 8-bit S-box of AES [25], [35], [36]. Lightweight ciphers to which higher-order Boolean masking is efficiently applied generally adopt 8-bit S-boxes that use less than 13 nonlinear operations [5], [12]. These S-boxes are often generated using S-box construction with structures such as the Lai-Massey and MISTY, which use 3-, 4-, or 5-bit S-boxes inside, for which an efficient bitsliced implementation of the generated S-box can be found.

However, the 8-bit S-box construction with structures has limitations with respect to security and efficiency, because the properties of the constructed 8-bit S-box are largely based on its employed construction structures and small S-boxes. It has been proven that the differential uniformity and linearity of many of 8-bit S-boxes generated using structures can not be optimal. For instance, the differential uniformity and linearity of the 8-bit S-box constructed with the MISTY structure should be 16 and 32 or more, respectively [37]. In particular, the construction with structures does not provide optimal differential uniformity or linearity for given number of nonlinear operations.

Reinforcement learning is often used in the problem of finding a new architecture by stacking operations or layers. Zoph *et al.* succeeded in searching for an efficient architecture by using reinforcement learning for neural architecture and neural optimizer search [38], [39]. Azalia *et al.* used reinforcement learning for chip placement and found unique architectures that were never discovered before [40]. These studies are similar to our studies to generate a secure S-box by stacking bitwised operations, and to reduce the number of nonlinear operations in the process.

B. CONTRIBUTIONS

In this paper, we propose a new method for generating S-boxes by stacking bitwise operations from the identity function with reinforcement learning, which is the opposite of the methods generally used to find bitsliced implementations of specific S-boxes.

Our S-box generation method repeats an action that stacks bitwise operations to the bitsliced implementation of the current S-box, deriving the bitsliced implementation of next S-box (Section II-B). Since this generation method can be expressed as a Markov model, it can be solved using reinforcement learning (RL). We trained this S-box generating procedure to the RL agents. With this approach, an S-box and its bitsliced implementation are generated simultaneously. Our methodology could easily generate 4-bit S-boxes corresponding to the state of the art. In the case of 8-bit S-boxes, our method generates an S-box with differential uniformity 16, but implemented with only 9 nonlinear operations, which can not be constructed in the existing structures; for differential uniformity 16, Feistel, unbalanced MISTY

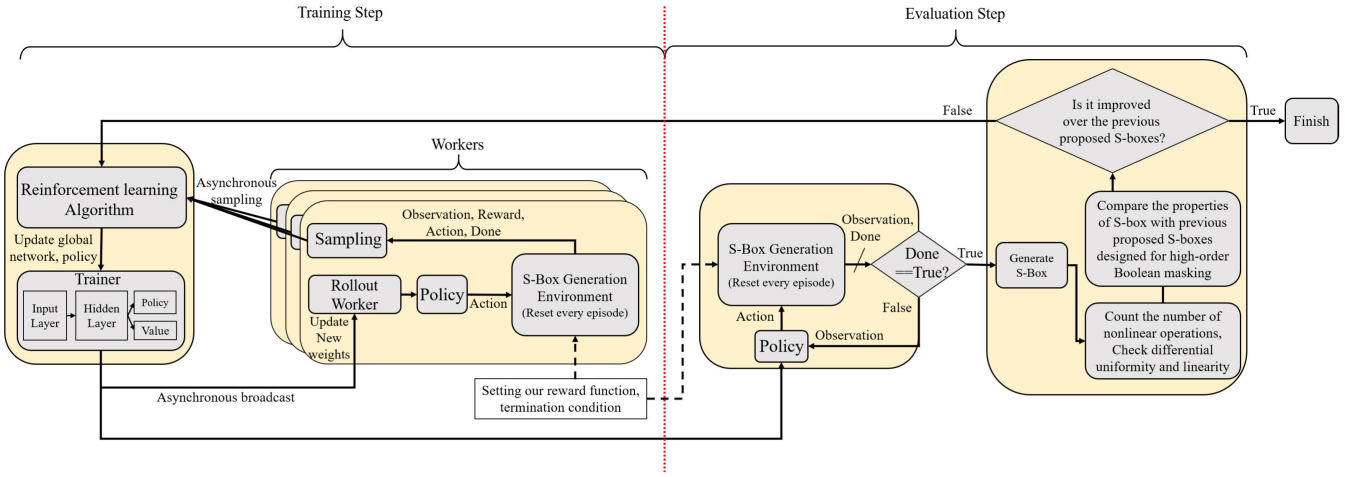


FIGURE 1. S-box generation workflow using reinforcement learning.

(and unbalanced Bridge [41]) and Lai-Massey (and MISTY, Bridge [41]) requires at least 10, 11 and 12 nonlinear operations, respectively. We could also generate S-boxes with differential uniformity 32, implemented with only 8 nonlinear operations. The SKINNY S-box, which is one of the most lightweight 8-bit S-boxes proposed, is also implemented using 8 nonlinear operations, but its differential uniformity is 64. Furthermore, 12 nonlinear operations were used to generate S-boxes having the same security properties as existing S-boxes such as Robin [12] and LITTLUN [5].

Our contributions are as follows:

1) We introduce a new S-box generation methodology using reinforcement learning.
2) We show that our methodology is valid by generating S-boxes superior or comparable to the state-of-the-art S-boxes.
3) We generate S-boxes with differential uniformity 16, implemented with 9 nonlinear operations, which is beyond the limit of S-box construction with structures.
4) We improve the differential uniformity over the SKINNY S-box implemented with the same number of nonlinear operations and the same linearity.

Fig. 1 illustrates the workflow of our S-box generation methodology.

II. REINFORCEMENT LEARNING FOR GENERATING S-BOXES AND THEIR BITSLICED IMPLEMENTATIONS

Reinforcement learning (RL) is a set of algorithms that solve problems expressed in the MDP. If a current state is probabilistically derived from a previous state, and the previous state was also derived in the same manner, this stochastic process is called a Markov chain. Such a model, in which the current state is expressed as a result generated by an action on the previous state, can be solved using RL.

Our S-box generation method repeatedly takes an action that adds bitwise operations, starting from the initial state (*IS*). It does not generate the bitsliced implementation of a specific given S-box, but finds a secure S-box using less than or equal to a set number of operations. We found that this S-box generation method can be expressed in MDP, and by applying RL, S-boxes with desired conditions can be generated. We aimed to generate S-boxes that can be implemented with minimum of nonlinear operations, while having certain security for differential and linear cryptanalysis. The main purpose of our RL agents is to generate a secure S-box, while reducing the number of nonlinear operations. In this section, we describe our detailed environment, reward functions, actions and state representations for the RL agents.

A. CONSTRUCTION OF THE ENVIRONMENT

S-boxes can be expressed in Algebraic Normal Form (ANF), so all S-boxes can be (bitsliced) implemented using only AND, XOR, NOT, and TEMP (MOV) operations.[1]

Among these operations, the NOT operation has the following properties.

$$(\sim b) \wedge c = (b \wedge c) \oplus b,$$
$$(\sim b) \wedge (\sim c) = \sim ((b \wedge c) \oplus b \oplus c).$$

With these properties, the NOT operation can be moved to the outermost I/O without changing the number of nonlinear operations. Even if the outermost NOT operation is removed, affine equivalence[2] is maintained. The NOT operation does not affect the differential uniformity and linearity. Thus, we could exclude NOT operations in our environment. However, if NOT operation is not used, input 0 becomes output 0, and as a result, at least one fixed point occurs with a probability of 1. Therefore, if necessary, in order to remove the fixed point, a NOT operation should be added on the bitsliced implementation of the generated S-box.

[1] All other operations are also expressed as logical gates of AND, XOR, and NOT. For example, OR can be created with AND and XOR ($A \vee B = (A \wedge B) \oplus A \oplus B$).

[2] The differential uniformity and linearity of an S-box do not change in the same affine equivalence class [28], [42]. In this paper, we consider differential uniformity and linearity as the security properties of an S-box.

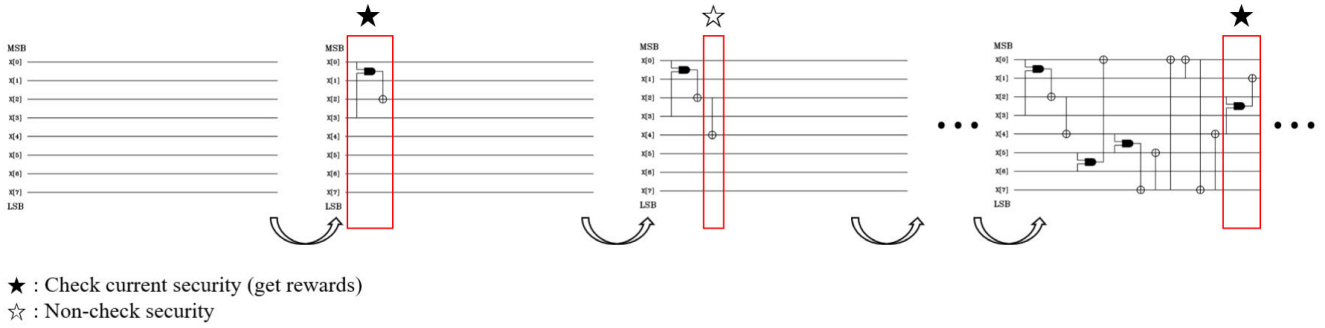


FIGURE 2. S-box generation process using RL.

We devised an environment in which an agent can stack AND and XOR operations to generate bijective S-boxes. Modern block ciphers proposed after DES [43] do not often use non-bijective S-boxes, because of the decrease in entropy and the existence of zero output difference even for a non-zero input difference of S-boxes. We induced RL to generate bijective S-boxes by limiting the actions that can occur in the environment.

There are several cases in which non-bijective S-boxes are generated. If the last operation of an S-box implementation is AND, only a non-bijective S-box can be generated. A non-bijective S-box can be also generated when AND operations are used consecutively, or when TEMP memory is used in the implementation. In our experiments, as the number of ANDs or TEMP memories used in the implementation increased, it became more difficult to generate a bijective S-box. Therefore, to make all S-boxes generated using RL bijective, we devised an environment that uses only XOR and AND-XOR operations. The definition of XOR operation and AND-XOR operation used in this environment are as follows.

$$XOR(a,b) := X[a] \leftarrow X[a] \oplus X[b] \quad a \neq b, a, b \in \mathbb{Z}_n$$
$$AND-XOR(a,b,c) := X[a] \leftarrow X[a] \oplus (X[b] \wedge X[c])$$
$$|\{a,b,c\}| = 3,\ a,b,c \in \mathbb{Z}_n$$
$$X[i] : i^{th}\text{position of bit tn the } n\text{-bit S-box}$$

The RL agents can stack the AND-XOR operation or the XOR operation as an action of every step. We represent the state of applying the i^{th} step to IS as $Stack_i$. The IS is $Stack_0$, and the last state, $Stack_{end}$, becomes an S-box when the episode is terminated.

B. PROBLEM STATEMENT

In the environment defined in Section II-A, the RL agents updates and generates an S-box by stacking AND-XOR or XOR operations onto IS. When the RL agents are using AND-XOR operations, the environment checks the cryptographic properties of the S-box, and provides rewards to the RL agents according to the reward setting of the environment (Fig. 2).

RL problems can be formulated as MDPs, consisting of four key elements: states, actions, state transitions, and rewards. The meaning of each element and the explanation of our case are as follows.

1) STATES

a: THE SET OF POSSIBLE OBSERVATION STATES OF THE ENVIRONMENT

We tried cryptographically meaningful deterministic or stochastic data as an observation state, such as the S-box differential distribution tables (DDTs) and linear approximation tables (LATs), the target cryptographic properties of an S-box, and the current cryptographic properties of the S-box, but most of the combinations produced similar performance. Therefore, we used only the S-box table as the observation state, as it was the most effective in terms of training speed and memory. With this observation state, the next state is deterministically chosen by the RL agents.

2) ACTIONS

a: THE SET OF ACTIONS THAT CAN BE TAKEN BY THE AGENTS

In our environment, the RL agents select AND-XOR or XOR, and select the bit positions to apply. The number of cases where 3 bits are selected to apply AND-XOR to n bits is $n \times (n-1) \times (n-2)$. Although the bits to apply AND-XOR can be swapped, the results are same. Thus, the number of cases has to be divided by two. The number of cases where 2 bits are selected to apply XOR to n bits is $n \times (n-1)$. For this reason, the RL agents choose one of $(n \times (n-1) \times (n-2))/2 + n \times (n-1)$ actions at each step.

3) STATE TRANSITION PROBABILITY

a: THE PROBABILITY OF BEING UPDATED TO NEXT STATE WHEN AN ACTION IS APPLIED TO THE CURRENT STATE

Since the state of our environment is deterministically generated, probability of transition from the current state to the next state is 1.

4) REWARDS

a: EXPECTED REWARD FOR AN ACTION IN THE CURRENT STATE

The environment gives a reward to the RL agents based on the differential uniformity or linearity of the updated state, or the number of operations used. Differential uniformity

and linearity can be derived from the DDT and the LAT, respectively, in the process of generating the S-box.

The DDT of n-bit S-box S (DDT_S) is defined as below, for $\Delta a, \Delta b \in \mathbb{Z}_2^n$,

$$DDT_S(\Delta a, \Delta b) = |\{x \in \mathbb{Z}_2^n | S(x) \oplus S(x \oplus \Delta a) = \Delta b\}|.$$

The differential uniformity S is the largest value among the entries of DDT_S, excluding the value of $DDT_S(0, 0)$.

The LAT of n-bit S-box S (LAT_S) is defined as below; for $\Lambda a, \Lambda b \in \mathbb{Z}_2^n$,

$$LAT_S(\Lambda a, \Lambda b) = |\{x \in \mathbb{Z}_2^n | x \cdot \Lambda a = S(x) \cdot \Lambda b\}|.$$

The largest value among the terms of LAT_S excluding the entry $(0, 0)$ is defined as linearity.

We experimented with various reward shapings to satisfy the cryptographic properties and minimize the number of nonlinear operations. We and selected three types of conditions with which to train the RL agents. All reward shaping was experimentally selected in consideration of the reward decay 0.99 to minimize the number of AND-XORs. The three types of rewards are as follows:

The first reward shaping below induces an increase in cryptographic security within an episode.

$$\textbf{reward}_1 = \begin{cases} \alpha & \text{When using AND-XOR and} \\ & \alpha \text{ is a positive number or zero} \\ \alpha - 1 & \text{When using AND-XOR and} \\ & \alpha \text{ is a negative number} \end{cases}$$

termination condition$_1$: N steps (We set $N = 500$.)

Here, $\alpha =$ (previous differential uniformity - current differential uniformity) + (previous linearity - current linearity).

In this reward shaping, when the RL agents use the AND-XOR operation as an action, a reward is given. The RL agents will receive a penalty whenever cryptographic security is decreased, *i.e.*, α is negative. This penalty prevents the RL agents from repeating the same action. As the termination condition in this reward, we set $N = 500$, which is a moderately large number experimentally selected.

The second reward shaping induces a reduction of the number of nonlinear operations while satisfying the specified cryptographic properties of the S-box.

$$\textbf{reward}_2 = \begin{cases} -0.001 & \text{When using XOR} \\ -1.001 & \text{When using AND-XOR} \\ -250 & \text{When reach maximum step} \end{cases}$$

termination condition$_2$: if satisfying some specific cryptographic properties, or N steps (we set $N = 250$.)

In this reward shaping, RL agents can receive a reward in various cases. The episode is terminated when some specified cryptographic properties (which will be explained in Section III) are satisfied, or the maximum length of the episode is reached. If the episode ends and the S-box does not satisfy the cryptographic properties, the RL agents get a large penalty. One of the methods used to achieve the goal and quickly terminate the episode is to give a small penalty for each step. When AND-XOR is used, an additional penalty is given. With these settings, the RL agents will try to end the episode quickly, using as few AND-XOR operations as possible.

Our RL process using reward 1 or 2 is implemented based on the OpenAI gym style interface as shown in the algorithm 1 [44].

Algorithm 1 Detailed Process of **reward$_{1\text{ or }2}$** Environment

```
agent ← Reinforcement learning algorithm
ENV ← S-box generation environment
ENV.reward_function ← reward_n
IS ← n bit identity function
mem=[]
while Evaluation do
    ENV.reset()
    /* Stack_0 ← IS                                   */
    done ← False
    i ← 0
    while Not done do
        act ← agent.select_action(Stack_i)
        /* act is derived from policy                 */
        Stack_{i+1}, reward, done ← Step(act)
        mem.append(Stack_i, Stack_{i+1}, reward, done, act)
        i+=1
    end while
    agent.train(mem, network, hyperparameters)
    mem.clear()
    sbox ← generates(agent)
    result ← Measure security and efficiency (sbox)
    if Does the result meet the criteria? then
        break
    end if
end while
```

The last reward shaping induces the RL agents to learn to improve the cryptographic properties within N AND-XORs.

$$\textbf{reward}_3 = \begin{cases} +1 & \text{When using AND-XOR} \\ & \text{improves security} \\ 0 & \text{When using AND-XOR} \\ & \text{maintains security} \\ -1.1 & \text{When using AND-XOR} \\ & \text{weakens security} \\ -100 & \text{When reach maximum step} \end{cases}$$

termination condition$_3$:N steps or using M AND-XORs (In our settings, $N = 100$ and $M \leq 4$.)

In this reward shaping, the episode is terminated when maximum length of the episode is reached, or the maximum available AND-XORs are all used. The environment measures the cryptographic properties of the S-box when

AND-XOR is used, and gives a reward of +1 if the security was improved, or a penalty of −1.1 if its security was worsened. This reward shaping is intended to force the RL agents to explore more. A detailed explanation of **reward$_3$** will be covered in Section III-B.

C. SELECTED ALGORITHMS FOR RL

In general, for sample-efficient methods, offline learning is effective, and for other methods, online learning is suitable. Even if all the efficient samples that can be implemented with a small number of nonlinear operations among S-boxes announced so far are used, sufficient learning cannot be achieved. In addition, while our S-box generation method is a general methodology, most of the efficient samples published so far are not suitable for learning because almost all of them are biased data generated through the structure. So, we have chosen online algorithms such as Asynchronous PPO, Impala without replay buffer. The best results we found is when we used Proximal Policy Optimization (PPO), Asynchronous Proximal Policy Optimization (APPO), and IMPortance weighted Actor-Learner Architecture (IMPALA) [45], [46]. These three algorithms are all designed for learning in a discrete action space.

PPO is a Policy-Based algorithm derived from Trust region policy optimization (TRPO) [47]. It is an algorithm that takes only the merits of TRPO and excludes the disadvantages. Unlike TRPO, which approximates the objective function using second-order approximation, only first-order approximation is used in PPO. Importance-Sampling, Ratio Clipping, and KL Penalty terms are used to perform stable weight updates, and have high convergence, but due to the nature of the Policy-Based algorithm, it is easy to fall into local minima.

APPO is the asynchronous version of PPO. It has advantages similar to those of A3C [48] which are obtained through an asynchronous update, and more stable and faster weight updates are possible by reducing the Time Related Correlation between samples. However, the algorithm has a disadvantage because it is not efficient, and it is still easy for it to fall into local minima.

IMPALA is an algorithm that can be used off-policy by using importance sampling in a representative form of policy-based RL. It is sample efficient, because Replay Buffer is used, and its training is also fast, because Multi-Actor and Multi-Learner are used. Due to the parallel structure, the gradient is not shared like A3C, and experiences are shared like APE-X, so the delay for weight update is also small. Stable update is also possible through the V-Trace formula.

III. EXPERIMENT AND RESULTS

Our experiments were conducted using the PPO, APPO, and IMPALA learning algorithms. We used Ray [49] as a universal API for building distributed applications, and RLlib [50] which is an open-source library for reinforcement learning, to ensure the accuracy of learning algorithms. We used a PC with an AMD Ryzen Threadripper CPU 2990WX Processor@ 3.00GHz, 128 GB RAM, NVIDIA GTX 1080Ti for our experiments, and it took about one month for each deep reinforcement learning agent to be trained. We used a basic architecture that returns policy and value after the fully connected layer for our RL agents's policy and value network (Fig. 3).

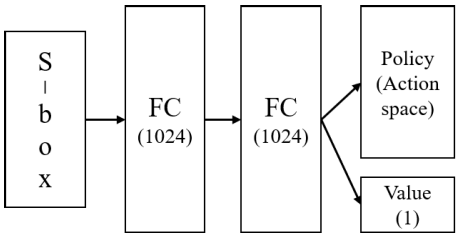


FIGURE 3. Policy and value network architecture (FC represents Fully Connected Layer.

A. 4-BIT S-BOXES

1) reward$_1$ EXPERIMENTS

We checked whether the task of generating the S-boxes having optimal[3] differential uniformity and linearity could be learned by RL agents. We conducted the experiment by setting *IS* as the identity function and the reward function as **reward$_1$**. It was possible to generate 4-bit S-boxes with optimal differential uniformity 4 and linearity 8, even when the actions were randomly selected. In order to clarify whether the training was producing the desired results, we calculated the average cryptographic properties for each episode. In this paper, we used 64 workers per global network. A lowered average value indicates that the proportion of S-boxes with optimal differential uniformity and linearity among the generated S-boxes increases, and also indicates that learning is continuously progressing. Fig. 4 shows moving average lines of the learning results.

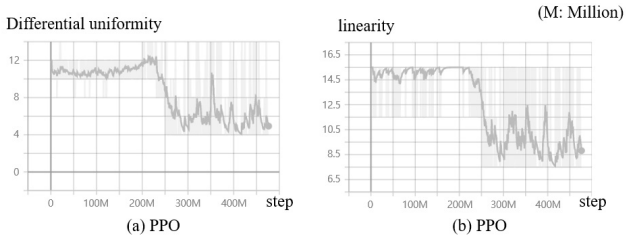


FIGURE 4. Results of learning with reward$_1$.

2) reward$_2$ EXPERIMENTS

Based on Section III-A1), we checked whether the task of optimizing the number of AND operations to generate S-boxes with an optimal security could be learned by RL agents or not. We conducted the experiment by setting *IS* as the identity function and the reward function as **reward$_2$**.

[3]One of the equivalence classes named *optimal* was established by Leander and Poschmann [42].

Fig. 5 shows moving average lines of the number of operations for each learning episode. According to [51], the minimum number of nonlinear operations to implement 4-bit S-box with optimal security is 4. Our RL agents that were trained by PPO, APPO, or IMPALA generated S-boxes having optimal security, implemented with 4 ANDs. Fig. 6 is one of the S-boxes that the RL agents generated with **reward$_2$**; its C implementation is given in Appendix . The results of these experiments demonstrate that it is possible to train the agents to generate S-boxes by considering their cryptographic properties and using a bitsliced implementation of S-box via RL.

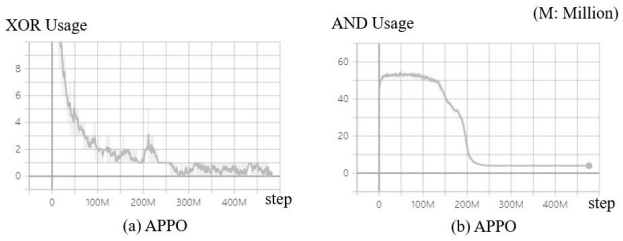


FIGURE 5. Results of learning with **reward$_2$**.

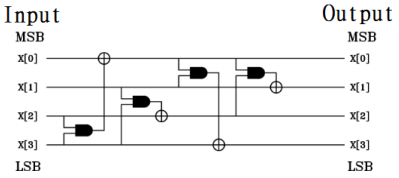


FIGURE 6. S-box logic circuit implemented with 4 ANDs generated by RL agents using **reward$_2$** (differential uniformity 4, linearity 8).

B. 8-BIT S-BOXES

1) reward$_1$ EXPERIMENTS

Unlike 4-bit S-boxes, no efficient tool has been proposed to produce a bitsliced implementation of 8-bit S-boxes. The tools to find bitsliced implementation using Dijkstra's algorithm or the SAT solvers, are efficient for 4-bit S-boxes, but they cannot be used for 8-bit S-boxes because of the large number of possible cases. Therefore, S-box construction structures such as Feistel, Lai-Massey and MISTY are usually used to generate 8-bit S-boxes to bitsliced implement them [5], [37]. We investigated various S-boxes with bitsliced implementations, and we set as our goal the generation of S-boxes with a differential uniformity 64 and a linearity 128 using 8 ANDs,[4] or with a differential uniformity 16 and a linearity 64 using 12 ANDs.[5]

Since 8-bit S-boxes are not easily generated if actions are taken randomly, we checked whether the task of generating the S-boxes with differential uniformity 16 and linearity 64 is possible for the RL agents. We conducted the experiment by setting *IS* as the identity function and the reward function as **reward$_1$**. All of the agents could generate secure S-boxes with differential uniformity 16 and linearity 64 within a short time.

2) reward$_2$ EXPERIMENTS

We confirmed that our RL improved the cryptographic properties of S-boxes using **reward$_1$**. We then used **reward$_2$** to reduce the number of AND-XOR operations to implement an S-box at the same time. We conducted this experiment by setting *IS* as the identity function and **reward$_2$** as the reward function. Fig. 7 shows the moving average lines of the usage of XORs (left) and AND-XORs (right) at the end of episode.

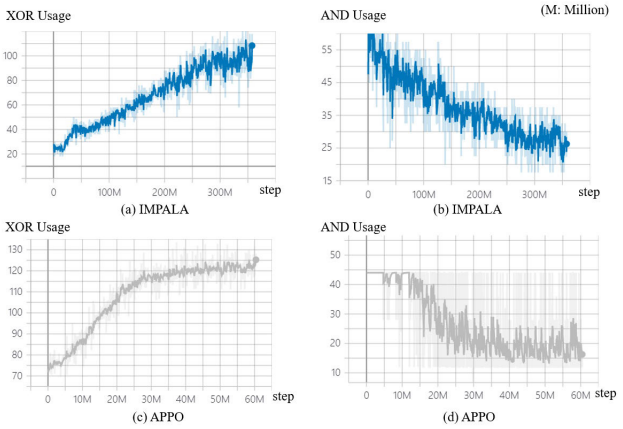


FIGURE 7. Results of learning with **reward$_2$**.

We can observe that the RL agents keep its cryptographic properties, but use XOR more, while reducing the use of AND-XOR. If the S-box is generated using only linear operations, only cryptographically weak S-boxes will be generated. Therefore, it is necessary to use nonlinear operations to achieve cryptographic security of the S-box, and the linear operations serve an auxiliary role in maximizing the effect of the nonlinear operation. The RL agents initially used many ANDs, but it was trained to maximize the effect of the nonlinear operations by increasing the number of XORs, and setting them in an appropriate position. This result was obtained because the penalty for increasing the number of nonlinear operations was relatively large compared to the penalty for increasing the number of linear operations. After continuous learning, the RL agents generated S-boxes with differential uniformity 16 and linearity 64, implemented with 12 ANDs. Fig. 8 represents one of these S-boxes; its C implementation is given in Appendix

However, the number of AND-XORs was not further reduced, even with additional learning. Therefore, we changed the environment, taking this issue into account. We created a new environment with **reward$_2$** that is designed to consider only one of the cryptographic properties at a time; differential uniformity, or linearity, not both. As a result, the RL agents generated S-boxes with a differential uniformity 16, implemented with 11 ANDs. However, the linearity

[4]Properties of S-box used in SKINNY [52].
[5]Properties of S-box used in Robin and FLY [5], [12].

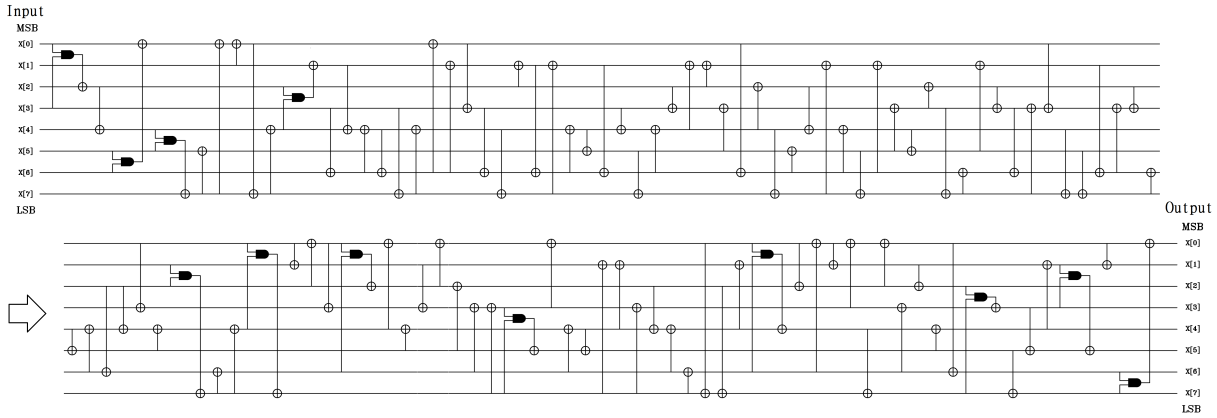


FIGURE 8. S-box logic circuit implemented with 12 ANDs generated by RL agents using $\textbf{reward}_2$ (differential uniformity of 16, linearity 64).

of the S-boxes generated by the RL agents still had a limit of 64, implemented with 12 ANDs, and 80 implemented with 11 ANDs. To further improve these results, we progressed the learning by tuning the hyper-parameters in various ways. However, the RL agents did not encounter any new cases during its exploration, and thus the entropy continued to decline; entropy generally rises when the RL agents encounters a new case. Based on these results, we determined that tuning of the hyperparameters alone does not allow the RL agents to explore sufficiently. To solve this exploration problem, we designed and applied a new environment with $\textbf{reward}_3$, as presented in Section III-B3.

3) $\textbf{reward}_3$ EXPERIMENTS

In order to increase the exploration carried out by the agents, we changed the *IS* of the environment from the identity function to a variable, and defined the reward function as $\textbf{reward}_3$. This aims to improve security with M AND-XOR operations from its current state. The RL agents is rewarded for improving security and penalized if not.

When using $\textbf{reward}_3$, the value of M needs to be adjusted appropriately. If M is small, the RL agents can never improve the security of the S-box, depending on the size of the S-box. However, if M is set too high, the RL agents should consider a sparse reward. Therefore, we devised a method of changing the *IS*, to address this consideration.

The environment pushes the S-box generated by workers in each episode into the queue. When the environment is reset, it sets the *IS* to an S-box randomly popped from the queue. These environments have multiple start points, allowing for better exploration and leading to positive rewards even if M is small. Algorithm 2 shows the details of the learning process.

After sufficient training of the RL agents, S-boxes with differential uniformity 32 and linearity 128, implemented with 8 ANDs were generated. We could also generate S-boxes with differential uniformity 16 and linearity 128, implemented with 9 ANDs. Fig. 9 shows one of the S-boxes that the RL agents generated with $\textbf{reward}_3$; its C implementation is given in Appendix .

Algorithm 2 Detailed Process of $\textbf{reward}_3$ Environment

agent ← Reinforcement learning algorithm
ENV ← S-box generation environment
ENV.reward_function ← $\textbf{reward}_3$
$N \leftarrow 100$; $M \leftarrow 2$
$I_n \leftarrow$ n bit identity function
$Q_0 \leftarrow \{I_n\}$
mem = []
for $i \in 1, \ldots, n$ **do**
    $Q_i \leftarrow$ Empty_Queue
end for
while Evaluation **do**
    $Q_k \overset{\$}{\leftarrow} \{Q_0, Q_1, \ldots, Q_n\}$
    if $Q_k ==$ Empty_Queue **then**
        continue
    end if
    $Stack_0 \overset{\$}{\leftarrow} Q_k$; i ← 0
    done ← False
    while Not done **do**
        act ← agent.select_action($Stack_i$)
        `/* act is derived from policy */`
        $Stack_{i+1}$, reward, done ← Step(act)
        mem.append($Stack_i$, $Stack_{i+1}$, reward, done, act)
    end while
    Q_{k+M}.append($Stack_{end}$)
    agent.train(mem, network, hyperparameters)
    mem.clear()
    sbox ← generates(agent)
    result ← Measure security and efficiency (sbox)
    if Does the result meet the criteria? **then**
        break
    end if
end while

However, in the case of linearity, there was still no significant improvement in performance, and it was still 64 when implemented with 12 ANDs.

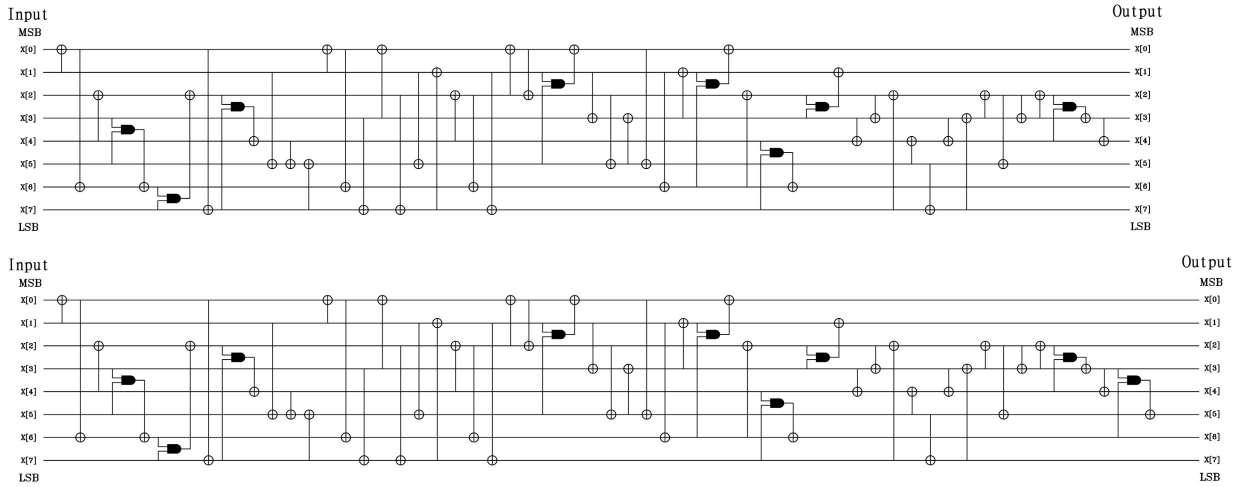


FIGURE 9. S-box logic circuits generated by RL agents implemented with 8 ANDs using reward_3 (differential uniformity 32, linearity 128, above) and implemented with 9 ANDs using reward_3 (differential uniformity 16, linearity 128, below).

a: LIMITATIONS OF THE EXISTING STRUCTURE-BASED 8-BIT S-BOX CONSTRUCTION WITH 9 ANDs

Feistel, Lai-Massey and MISTY structures require three small S-boxes, at least two of which are active. According to [37], when two S-boxes are active in each structure, the infimum of the products of the differential uniformity of each S-box becomes the differential uniformity of the generated S-box. 9 ANDs can be distributed as a combination of (3,3,3), (4,3,2), or (4,4,1) for the three small 4-bit S-boxes. A bijective 4-bit S-box can have differential uniformity 4 when 4 ANDs are used, and differential uniformity 8 when 3 ANDs are used [51]. Furthermore, based on our exhaustive search, a bijective 4-bit S-box can have differential uniformity 8 when 2 ANDs are used, and differential uniformity 16 when 1 AND is used. Thus, the infimum of differential uniformity for each combination is 64. In the case of Feistel, a non-bijective S-box can be partially used. In this case, the infimum is 32.

If the structure used is unbalanced, one 3-bit S-box and two 5-bit S-boxes are required. In the case of the 5-bit S-boxes, its infimum differential uniformity is 4 when 5 ANDs and is 2 when 7 ANDs are used [53]. The 3-bit S-box can be almost perfect nonlinear (APN[6]), when 3 ANDs are used. Then it is impossible to construct an 8-bit S-box with differential uniformity 16 using 9 ANDs, if the 8-bit S-box is constructed using one 3-bit S-box and two 5-bit S-boxes. The reason is as follows.

i) When the 3-bit S-box is constructed with 3 ANDs, remaining 6 ANDs can be divided into (6,0), (5,1), (4,2), (3,3) for two 5-bit S-boxes.
ii) When the 3-bit S-box is constructed with 2 ANDs, the remaining 7 ANDs can be divided into (7,0), (6,1), (5,2), (4,3) for two 5-bit S-boxes.
iii) When the 3-bit S-box is constructed with 1 AND, the remaining 8 ANDs can be divided into (8,0), (7,1), (6,2), (5,3), (4,4) for two 5-bit S-boxes.
iv) And when the 3-bit S-box is constructed with no ANDs, the remaining 9 ANDs can be divided into (9,0), (8,1), (7,2), (6,3), (5,4) for two 5-bit S-boxes.

In all of these combinations, the products of differential uniformity of two 5-bit S-boxes are each greater than 16. In addition, even if two 5-bit S-boxes have differential uniformity 4 when using 5 ANDs each, differential uniformity of constructed S-box cannot be 16 because 3-bit S-box will have differential uniformity 8. Therefore, to construct an 8-bit S-box with differential uniformity 16 in an unbalanced structure, at least 11 ANDs are required (3 for 3-bit S-box and 4 for 5-bit S-boxes) [11], [12].

Our S-box generation method overcomes the above limitations, as we have proposed an S-box with differential uniformity 16 that can be implemented with only 9 ANDs.

IV. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new S-box generation method by stacking bitwise operations from the identity function, which is a different approach from existing methods. Since our method can be expressed as a Markov model, we could utilize reinforcement learning (RL). We trained RL agents to stack XORs and AND-XORs, to generate secure and efficient S-boxes. The RL agents created secure S-boxes with fewer and fewer ANDs, as training proceeded. Finally, we succeeded in training the RL agents to generate meaningful 4-bit and 8-bit S-boxes. Our RL agents achieved 4-bit S-boxes with optimal security, implemented with 4 ANDs, which is proven to be the minimum number. They also achieved 8-bit S-boxes with differential uniformity 16 and linearity 64, implemented with 12 ANDs. Furthermore, our RL agents generated 8-bit S-boxes with differential uniformity 16, implemented with only 9 ANDs, and they generated 8-bit S-boxes with the same

[6]The differential uniformity of APN function is 2.

linearity 128 and the same number of nonlinear operations 8 as SKINNY, while providing differentail uniformity 32 better than that of SKINNY. Table 1 presents a comparison of various S-boxes.

TABLE 1. Comparison of 4- and 8-bit S-boxes.

S-box	S-box bit size	Differential uniformity	Linearity	#(Nonlinear operations)	Ref.
This paper	**4**	**4**	**8**	**4**	**Listing 1**
PRESENT	4	4	8	4	[54]
RECTANGLE	4	4	8	4	[6]
GIFT	4	6	8	4	[3]
Midori (Sb0)	4	4	8	4*	[55]
Midori (Sb1)	4	4	8	7*	[55]
This paper	**8**	**16**	**128**	**9**	**Listing 2**
This paper	**8**	**32**	**128**	**8**	**Listing 3**
SKINNY	8	64	128	8	[52]
Midori (SSb0∼SSb3)	8	64	128	14*	[55]
This paper	**8**	**16**	**64**	**12**	**Listing 4**
Fantomas	8	16	64	11	[12]
Robin	8	16	64	12	[12]
LITTLUN	8	16	64	12	[5]
LILLIPUT	8	8	64	12	[56]

*The number of nonlinear operations required to implement the S-boxes of Midori could be obtained using the tool Lighter [31].

In the papers proposing SKINNY and Midori, they ensure security against differential cryptanalysis (DC) and linear cryptanalysis (LC) by providing the minimum number of active S-boxes for each round. Since those figures are calculated based on the linear layer and structure of each block cipher, the figures do not change even if the S-box is replaced. More precisely, replaced S-boxes to SKINNY make the resistance of DC and LC higher than or equal to that of the original cipher, while requiring more nonlinear operations. As for the Midori cipher, they also provide higher or equal resistance of DC and LC as well as they reduce the number of required nonlinear operations. If the S-boxes in Listings 2, 3, and 4 are applied to SKINNY and Midori, the security against DC, LC and the number of nonlinear operations to implement the block cipher are as in Table 2.

TABLE 2. Changes in DC, LC security and number of nonlinear operations when replacing SKINNY and Midori's S-box.

		Original S-box	Listing2	Listing3	Listing4
SKINNY	15-round differential probability bounds (66 active S-boxes)	2^{-132}	2^{-264}	2^{-192}	2^{-264}
	15-round linear probability bounds (66 active S-boxes)	2^{-132}	2^{-132}	2^{-132}	2^{-264}
	Number of nonlinear operations to implementation of block cipher (Full-round)	288	324	288	432
Midori	7-round differential probability bounds (67 active S-boxes)	2^{-134}	2^{-268}	2^{-201}	2^{-268}
	7-round linear probability bounds (67 active S-boxes)	2^{-134}	2^{-134}	2^{-134}	2^{-268}
	Number of nonlinear operations to implementation of block cipher (Full-round)	280	180	160	240

In Fig. 7, we can see that the number of AND-XOR operations decreases, but the number of XOR operations increases according to the reward shaping of the 8-bit S-box. Reducing the number of ANDs is a very important task for efficient side-channel countermeasures, but the increased number of XORs increases the implementation cost in environments in which side-channel attacks are infeasible. Therefore, it is necessary to study reward shaping that decreases the number of linear operations, as well as the number of nonlinear operations. We also experimented the optimization of AND-depth [41] (data not shown). In this environment, we gave different rewards according to the cryptographic properties and AND-depth at the end of the episode. We could generate 4-bit S-boxes with optimal cryptographic properties and an AND-depth of 2, which is the best AND-depth. For 8-bit S-boxes, we could generate S-boxes with an AND-depth of 5,

```
// input MSB: X[0], LSB: X[3]
// output MSB: X[0], LSB: X[3]
X[0]^=(X[2]&X[3])
X[2]^=(X[1]&X[3])
X[3]^=(X[0]&X[1])
X[1]^=(X[0]&X[2])
//s[16] = {0,1,2,F,4,7,6,C,8,9,E,3,D,A,B,5};
```

LISTING 1. 4-bit S-box (Differential uniformity 4, Linearity 8).

```
// input MSB: X[0], LSB: X[7]
// output MSB: X[0], LSB: X[7]
X[0]^=X[1]; X[6]^=X[0]; X[2]^=X[4];
X[6]^=(X[3]&X[5]); X[2]^=(X[6]&X[7]);
X[7]^=X[0];         X[4]^=(X[2]&X[7]);
X[5]^=X[1]; X[5]^=X[4]; X[5]^=X[7];
X[0]^=X[1]; X[6]^=X[0]; X[7]^=X[3];
X[0]^=X[3]; X[7]^=X[2]; X[5]^=X[1];
X[1]^=X[7]; X[2]^=X[4]; X[6]^=X[2];
X[7]^=X[1]; X[0]^=X[2]; X[2]^=X[0];
X[0]^=(X[1]&X[5]);      X[3]^=X[1];
X[5]^=X[2]; X[3]^=X[5]; X[5]^=X[0];
X[6]^=X[1];             X[1]^=X[3];
X[0]^=(X[1]&X[6]);      X[2]^=X[6];
X[6]^=(X[4]&X[7]); X[1]^=(X[2]&X[3]);
X[4]^=X[3]; X[3]^=X[2]; X[2]^=X[7];
X[4]^=X[5]; X[7]^=X[5]; X[4]^=X[3];
X[3]^=X[7]; X[2]^=X[3]; X[5]^=X[2];
X[3]^=X[2];             X[2]^=X[3];
X[3]^=(X[2]&X[4]);      X[4]^=X[3];
X[5]^=(X[3]&X[6]);
/* s[256] =
    {0,6A,2A,3E,4D,52,BB,AF,7E,12,C8,
EF,5A,9F,34,D7,A6,BD,D4,DC,68,78,C2,A8,AD,D9,
47,11,8A,1C,B8,7D,8D,8,FB,C0,71,45,DF,24,1A,
85,F4,E4,97,61,A1,B1,37,B2,19,4F,14,82,E6,CA,
D1,23,63,77,7,F,6D,F2,86,90,6C,73,B,DD,39,4B,
E2,F0,50,7A,C6,CC,AC,2B,49,2E,FF,89,9B,84,F5,
56,35,27,DB,E9,16,95,20,36,13,1D,65,A2,EB,F9,
41,33,9E,8C,B0,5E,D3,1,25,BA,A9,BF,83,44,8E,
E0,7C,6E,3C,43,4E,D5,F6,18,58,E7,26,59,54,66,
42,FD,E8,B7,2,A,28,5C,CF,C7,3D,F8,31,3B,1F,
40,92,A4,64,AA,D,5,7B,17,5F,9A,F1,E1,9C,94,
72,F7,91,99,A3,8F,51,32,E3,38,ED,4A,87,2D,FE,
EA,48,CD,B4,80,DA,D2,2F,21,C5,6F,88,57,BE,74,
C9,96,7F,69,75,5B,1B,15,98,AE,B6,FC,55,D6,A7,
29,B3,9D,5D,53,10,1E,22,79,93,B9,E5,DE,C1,3A,
6B,EC,4,C3,EE,A0,9,67,3F,B5,BC,E,CE,C4,D8,3,
76,60,62,A5,D0,F3,2C,CB,46,30,AB,70,81,8B,C,
6,FA,4C};*/
```

LISTING 2. 8-bit S-box with 9 nonlinear operations (Differential uniformity 16, Linearity 128).

```
// input MSB: X[0], LSB: X[7]
// output MSB: X[0], LSB: X[7]
X[0]^=X[1]; X[6]^=X[0]; X[2]^=X[4];
X[6]^=(X[3]&X[5]); X[2]^=(X[6]&X[7]);
X[7]^=X[0];         X[4]^=(X[2]&X[7]);
X[5]^=X[1]; X[5]^=X[4]; X[5]^=X[7];
X[0]^=X[1]; X[6]^=X[0]; X[7]^=X[3];
X[0]^=X[3]; X[7]^=X[2]; X[5]^=X[1];
X[1]^=X[7]; X[2]^=X[4]; X[6]^=X[2];
X[7]^=X[1]; X[0]^=X[2]; X[2]^=X[0];
X[0]^=(X[1]&X[5]);      X[3]^=X[1];
X[5]^=X[2]; X[3]^=X[5]; X[5]^=X[0];
X[6]^=X[1];             X[1]^=X[3];
X[0]^=(X[1]&X[6]);      X[2]^=X[6];
X[6]^=(X[4]&X[7]); X[1]^=(X[2]&X[3]);
X[4]^=X[3]; X[3]^=X[2]; X[2]^=X[7];
X[4]^=X[5]; X[7]^=X[5]; X[4]^=X[3];
X[3]^=X[7]; X[2]^=X[3]; X[5]^=X[2];
X[3]^=X[2];             X[2]^=X[3];
X[3]^=(X[2]&X[4]);      X[4]^=X[3];
/* s[256] = { 0,6A,2A,3A,4D,56,BF,AF,7A,16,
C8,EF,5E,9B,34,D3,A6,BD,D4,DC,68,78,C2,A8,
AD,D9,47,11,8A,1C,B8,7D,8D,8,FF,C0,71,45,
DB,24,1E,85,F4,E4,93,61,A1,B1,33,B6,19,4F,
14,82,E6,CA,D1,23,63,73,7,F,6D,F6,86,90,6C,
77,B,DD,39,4B,E2,F0,50,7E,C6,CC,AC,2B,49,
2E,FB,89,9F,84,F5,52,35,27,DF,E9,12,95,20,
32,17,1D,65,A2,EB,F9,41,37,9A,8C,B0,5A,D7,
1,25,BE,A9,BB,83,44,8E,E0,7C,6E,3C,43,4E,
D5,F2,18,58,E7,26,59,54,66,42,FD,E8,B3,2,A,
28,5C,CF,C7,3D,F8,31,3F,1B,40,96,A4,64,AA,
D,5,7F,13,5B,9E,F1,E1,9C,94,76,F3,91,99,A3,
8F,51,36,E3,38,ED,4A,87,2D,FA,EA,48,CD,B4,
80,DE,D6,2F,21,C5,6F,88,53,BA,74,C9,92,7B,
69,75,5F,1F,15,98,AE,B2,FC,55,D2,A7,29,B7,
9D,5D,57,10,1A,22,79,97,B9,E5,DA,C1,3E,6B,
EC,4,C3,EE,A0,9,67,3B,B5,BC,E,CE,C4,D8,3,
72,60,62,A5,D0,F7,2C,CB,46,30,AB,70,81,8B,
C,6,FE,4C};*/
```

LISTING 3. 8-bit S-box with 8 nonlinear operations (Differential uniformity 32, Linearity 128).

differential uniformity 16 and linearity 64, while the best AND-depth of the 8-bit S-boxes was 2 [41].

We believe that there are still various methods which could be used to optimize our RL environment to generate better S-boxes. We also believe that S-box generation using RL agents can be used for the generation of S-boxes under different conditions. In future work, the following would be interesting research topics:

1) We used a basic network architecture. Which is the best network model for S-box generation through reinforcement learning?
2) How can we upgrade our S-box generation method to generate bijective S-boxes while using temp memory?
3) Can RL be used to generate super S-boxes such as 16- and 32-bit S-boxes?
4) We only used a model-free algorithm. What would be the results if a model-based algorithm is used?

```
// input MSB: X[0], LSB: X[7]
// output MSB: X[0], LSB: X[7]
X[2]^=(X[0]&X[3]);      X[4]^=X[2];
X[0]^=(X[5]&X[6]); X[7]^=(X[4]&X[5]);
X[5]^=X[7]; X[0]^=X[7]; X[0]^=X[1];
X[7]^=X[0];             X[4]^=X[7];
X[1]^=(X[2]&X[4]); X[6]^=X[3];
X[4]^=X[1]; X[4]^=X[6]; X[6]^=X[4];
X[7]^=X[3]; X[4]^=X[7]; X[0]^=X[6];
X[1]^=X[6]; X[3]^=X[0]; X[6]^=X[3];
X[7]^=X[4]; X[1]^=X[2]; X[6]^=X[1];
X[1]^=X[7]; X[4]^=X[6]; X[5]^=X[4];
X[6]^=X[1]; X[4]^=X[3]; X[7]^=X[5];
X[4]^=X[6]; X[3]^=X[2]; X[1]^=X[4];
X[1]^=X[2]; X[3]^=X[5]; X[6]^=X[0];
X[2]^=X[4]; X[7]^=X[4]; X[5]^=X[6];
X[4]^=X[2]; X[1]^=X[7]; X[4]^=X[6];
X[7]^=X[5]; X[1]^=X[6]; X[3]^=X[5];
X[5]^=X[4]; X[2]^=X[3]; X[7]^=X[3];
X[6]^=X[7]; X[1]^=X[5]; X[3]^=X[2];
X[6]^=X[3]; X[3]^=X[7]; X[3]^=X[0];
X[7]^=X[4]; X[7]^=X[5]; X[6]^=X[1];
X[3]^=X[6]; X[3]^=X[2]; X[6]^=X[7];
X[5]^=X[4]; X[4]^=X[6]; X[6]^=X[2];
X[4]^=X[2]; X[3]^=X[0]; X[4]^=X[5];
X[7]^=(X[1]&X[2]);      X[6]^=X[7];
X[4]^=X[7];         X[7]^=(X[0]&X[4]);
X[1]^=X[0]; X[0]^=X[2]; X[3]^=X[0];
X[2]^=(X[0]&X[6]);      X[0]^=X[4];
X[4]^=X[5]; X[3]^=X[1]; X[0]^=X[2];
X[2]^=X[5]; X[3]^=X[6]; X[3]^=X[7];
X[5]^=(X[3]&X[7]);      X[0]^=X[3];
X[4]^=X[6]; X[5]^=X[4]; X[1]^=X[7];
X[1]^=X[4]; X[3]^=X[7]; X[4]^=X[2];
X[4]^=X[6]; X[6]^=X[7]; X[7]^=X[0];
X[7]^=X[2];             X[1]^=X[4];
X[4]^=(X[0]&X[6]);      X[2]^=X[0];
X[0]^=X[6]; X[1]^=X[0]; X[0]^=X[3];
X[7]^=X[4]; X[0]^=X[2]; X[3]^=X[6];
X[2]^=X[1]; X[4]^=X[5]; X[6]^=X[0];
X[3]^=(X[2]&X[7]);      X[7]^=X[5];
X[5]^=X[3];             X[1]^=X[4];
X[5]^=(X[1]&X[3]);      X[1]^=X[0];
X[0]^=(X[6]&X[7]);
/* s[256] = {0,61,F1,CD,9F,37,4A,B0,89,2E,
73,40,7C,6,F2,88,5C,2D,72,51,C3,7F,9D,E7,53,
EB,6D,DE,60,15,71,5D,98,24,30,C1,F6,3,99,78,
A0,8,55,A4,3F,CE,84,6E,C0,68,33,10,BE,5B,8C,
79,A7,90,44,B1,3C,9B,1C,E9,7,7D,6B,1E,41,BF,
4B,A5,D5,E6,70,4C,34,CA,B3,D2,56,FE,EA,8F,DD
BC,DC,E0,CB,35,6C,DF,67,19,22,47,25,8D,C9,A3
2,F7,29,8E,49,23,F8,9,52,75,64,43,3D,57,1A,F0
11,36,31,D0,9E,74,B9,D3,C8,A2,FA,59,E4,81,18,
BB,B6,E,27,42,26,9A,87,F9,CC,A9,16,21,5A,AF,
2F,13,69,D1,D6,E1,62,93,DA,E2,ED,45,12,3A,
C2,A8,AE,82,EE,4,4E,66,3E,CF,8B,65,B5,1D,83,
AB,A1,96,86,E8,AA,50,20,17,7E,C6,1B,7A,94,
F5,C5,3B,E3,C4,D,BA,EC,5F,39,1,2C,4D,80,B8,
B,B7,8A,EF,E5,F,DB,F3,D9,6A,FB,5,92,77,28,
46,54,FC,7B,D8,FF,58,48,32,C7,F4,AC,1F,85,
6F,B2,91,A,BD,63,4F,FD,5E,D7,A6,76,C,38,14,
2B,9C,95,AD,97,B4,D4,2A};*/
```

LISTING 4. 8-bit S-box with 12 nonlinear operations (Differential uniformity 16, Linearity 64).

APPENDIX. BITSLICED IMPLEMENTATION CODES OF NEW S-BOXES

S-boxes generated by Listings 1,2,3 and 4 all have multiple fixed points. By adding the following operations to the end of Listing, one can reduce the number of fixed points while maintaining the DC and LC security.

1) Listing 1: $X[0] \wedge = NOT$ (FP:1)
2) Listing 2: $X[1] \wedge = NOT$ (FPs:0)
3) Listing 3: $X[0] \wedge = NOT$ (FPs:0)
4) Listing 4: $X[3] \wedge = NOT$ (FPs:0)

These operations of eliminating fixed points can also be defined in a linear layer rather than a nonlinear layer.

REFERENCES

[1] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[2] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 1109. Springer, 1996, pp. 104–113.

[3] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A small present—Towards reaching the limit of lightweight encryption," in *Proc. CHES*, in Lecture Notes in Computer Science, vol. 10529, W. Fischer and N. Homma, Eds. Cham, Switzerland: Springer, 2017, pp. 321–345.

[4] B. Gérard, V. Grosso, M. Naya-Plasencia, and F. Standaert, "Block ciphers that are easier to mask: How far can we go," in *Proc. 15th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)* in Lecture Notes in Computer Science, vol. 8086, G. Bertoni and J. Coron, Eds. Santa Barbara, CA, USA: Springer, Aug. 2013, 2013, pp. 383–399.

[5] P. Karpman and B. Grégoire, "The LITTLUN S-box and the fly block cipher," in *Proc. Lightweight Cryptogr. Workshop*, 2016, pp. 17–18.

[6] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, "RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms," *Sci. China Inf. Sci.*, vol. 58, no. 12, pp. 1–15, Dec. 2015.

[7] J. Boyar and M. G. Find, "Multiplicative complexity of vector valued Boolean functions," *Theor. Comput. Sci.*, vol. 720, pp. 36–46, Apr. 2018.

[8] J. Boyar *et al.*, "The relationship between multiplicative complexity and nonlinearity," in *Proc. Int. Symp. Math. Found. Comput. Sci.* Berlin, Germany: Springer, 2014, pp. 130–140.

[9] J. Boyar, M. G. Find, and R. Peralta, "On various nonlinearity measures for Boolean functions," *Cryptogr. Commun.*, vol. 8, no. 3, pp. 313–330, Jul. 2016.

[10] J. Bahrami, V. B. Dang, A. Abdulgadir, K. N. Khasawneh, J.-P. Kaps, and K. Gaj, "Lightweight implementation of the LowMC block cipher protected against side-channel attacks," in *Proc. 4th ACM Workshop Attacks Solutions Hardw. Secur.*, Nov. 2020, pp. 45–56.

[11] H. Kim *et al.*, "PIPO: A lightweight block cipher with efficient higher-order masking software implementations," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Cham, Switzerland: Springer, 2020, pp. 99–122.

[12] V. Grosso, G. Leurent, F. Standaert, and K. Varici, "Ls-designs: Bitslice encryption for efficient masked software implementations," in *Proc. 21st Int. Workshop*, in Lecture Notes in Computer Science, vol. 8540, C. Cid and C. Rechberger, Eds. London, U.K.: Springer, Mar. 2014 pp. 18–37.

[13] E. S. Abuelyman, A.-A. S. Alsehibani, and S. Arabia, "An optimized implementation of the s-box using residue of prime numbers," *Int. J. Comput. Sci. Netw. Secur.*, vol. 8, no. 4, pp. 304–309, 2008.

[14] L. Cui and Y. Cao, "A new S-box structure named affine-power-affine," *Int. J. Innov. Comput., Inf. Control*, vol. 3, no. 3, pp. 751–759, Jun. 2007.

[15] J. Kim* and R. C.-W. Phan, "Advanced differential-style cryptanalysis of the NSA's skipjack block cipher," *Cryptologia*, vol. 33, no. 3, pp. 246–270, Jul. 2009.

[16] K. Nyberg, "Differentially uniform mappings for cryptography," in *Proc. Workshop Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1993, pp. 55–64.

[17] M. T. Tran, D. K. Bui, and A. D. Duong, "Gray S-box for advanced encryption standard," in *Proc. Int. Conf. Comput. Intell. Secur.*, Dec. 2008, pp. 253–258.

[18] X. Yi, S. Xin Cheng, X. Hu You, and K. Yan Lam, "A method for obtaining cryptographically strong 8×8 S-boxes," in *Proc. GLOBECOM IEEE Global Telecommun. Conference. Conf. Rec.*, 1997, pp. 689–693.

[19] A. Zahid, M. Arshad, and M. Ahmad, "A novel construction of efficient substitution-boxes using cubic fractional transformation," *Entropy*, vol. 21, no. 3, p. 245, Mar. 2019.

[20] A. Zahid and M. Arshad, "An innovative design of substitution-boxes using cubic polynomial mapping," *Symmetry*, vol. 11, no. 3, p. 437, Mar. 2019.

[21] O. Kazymyrov, V. Kazymyrova, and R. Oliynykov, "A method for generation of high-nonlinear S-boxes based on gradient descent," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 578, Jun. 2013.

[22] K. Knezevic, "Combinatorial optimization in cryptography," in *Proc. 40th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2017, pp. 1324–1330.

[23] W. Millan, "How to improve the nonlinearity of bijective s-boxes," in *Proc. 3rd Australas. Conf., Inf. Secur. Privacy, (ACISP)* in Lecture Notes in Computer Science, vol. 1438, C. Boyd and E. Dawson, Eds. Brisbane, QLD, Australia: Springer, Jul. 1998, pp. 181–192.

[24] P. Kotlarz and Z. Kotulski, "On application of neural networks for S-boxes design," in *Proc. Int. Atlantic Web Intell. Conf. Adv. Web Intell. 3rd Int. Atlantic Web Intell. Conf., (AWIC)* in Lecture Notes in Computer Science, vol. 3528, P. S. Szczepaniak, J. Kacprzyk, and A. Niewiadomski, Eds. Lodz, Poland: Springer, Jun. 2005, pp. 243–248.

[25] M. Matsui and J. Nakajima, "On the power of bitslice implementation on intel Core2 processor," in *Proc. 9th Int. Workshop Cryptograph. Hardw. Embedded Syst.* in Lecture Notes in Computer Science, vol. 4727, P. Paillier and I. Verbauwhede, Eds. Vienna, Austria: Springer, Sep. 2007, pp. 121–134.

[26] D. A. Osvik, "Speeding up serpent," in *Proc. 3rd Adv. Encryption Standard Candidate Conf.*, New York, NY, USA: National Institute of Standards and Technology, Apr. 2000, pp. 317–329.

[27] T. B. S. Reis, D. F. Aranha, and J. L. Hernandez, "PRESENT runs fast—Efficient and secure implementation in software," in *Proc. CHES*, in Lecture Notes in Computer Science, vol. 10529, W. Fischer and N. Homma, Eds. Cham, Switzerland: Springer, 2017, pp. 644–664.

[28] M. O. Saarinen, "Cryptographic analysis of all 4×4-bit S-boxes," in *Proc. Int. Workshop Sel. Areas Cryptogr.* in Lecture Notes in Computer Science, vol. 7118, A. Miri and S. Vaudenay, Eds. Toronto, ON, Canada: Springer, Aug. 2011, pp. 118–133.

[29] M. Ullrich, C. De Canniere, S. Indesteege, O. Küçük, N. Mouha, and B. Preneel, "Finding optimal bitsliced implementations of 4× 4-bit S-boxes," in *Proc. SKEW Symmetric Key Encryption Workshop*, Copenhagen, Denmark, 2011, pp. 16–17.

[30] V. A. Dasu, A. Baksi, S. Sarkar, and A. Chattopadhyay, "LIGHTER-R: Optimized reversible circuit implementation for SBoxes," in *Proc. 32nd IEEE Int. Syst. Chip Conf. (SOCC)*, Sep. 2019, pp. 260–265.

[31] J. Jean, T. Peyrin, S. M. Sim, and J. Tourteaux, "Optimizing implementations of lightweight building blocks," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 4, pp. 130–168, Dec. 2017.

[32] A. Baysal and S. Sahin, "Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors," in *Proc. LightSec*, in Lecture Notes in Computer Science, vol. 9542, T. Güneysu, G. Leander, and A. Moradi, Eds. Cham, Switzerland: Springer, 2015, pp. 58–76.

[33] Z. Bao, J. Guo, S. Ling, and Y. Sasaki, "Sok: Peigen—A platform for evaluation, implementation, and generation of S-boxes," *IACR Cryptol. ePrint Arch.*, vol. 2019, no. 1, p. 209, 2019.

[34] K. Stoffelen, "Optimizing S-box implementations for several criteria using sat solvers," in *Proc. Int. Conf. Fast Softw. Encryption*. Berlin, Germany: Springer, 2016, pp. 140–160.

[35] E. Käsper and P. Schwabe, "Faster and timing-attack resistant AES-GCM," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* in Lecture Notes in Computer Science, vol. 5747, C. Clavier and K. Gaj, Eds. Lausanne, Switzerland: Springer, Sep. 2009, pp. 1–17.

[36] R. Könighofer, "A fast and cache-timing resistant implementation of the AES," in *Proc. Cryptographers Track RSA Conf.* Berlin, Germany: Springer, Apr. 2008, pp. 187–202.

[37] A. Canteaut, S. Duval, and G. Leurent, "Construction of lightweight S-boxes using feistel and MISTY structures," in *Proc. SAC*, in Lecture Notes in Computer Science, vol. 9566, O. Dunkelman and L. Keliher, Eds. Cham, Switzerland: Springer, 2015, pp. 373–393.

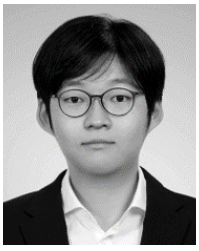
[38] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*. [Online]. Available: http://arxiv.org/abs/1611.01578

[39] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 459–468.
[40] A. Mirhoseini *et al.*, "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*. [Online]. Available: http://arxiv.org/abs/2004.10746
[41] B. Bilgin, L. De Meyer, S. Duval, I. Levi, and F.-X. Standaert, "Low and depth and efficient inverses: A guide on S-boxes for low-latency masking," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. 1, pp. 144–184, May 2020.
[42] G. Leander and A. Poschmann, "On the classification of 4 bit S-boxes," in *Proc. Int. Workshop Arithmetic Finite Fields*, in Lecture Notes in Computer Science, vol. 4547. Berlin, Germany: Springer, 2007, pp. 159–176.
[43] D. Coppersmith, "The data encryption standard (DES) and its strength against attacks," *IBM J. Res. Develop.*, vol. 38, no. 3, pp. 243–250, May 1994.
[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*. [Online]. Available: http://arxiv.org/abs/1606.01540
[45] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," 2018, *arXiv:1802.01561*. [Online]. Available: http://arxiv.org/abs/1802.01561
[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347
[47] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.
[48] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[49] P. Moritz, P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging $*AI*$ applications," in *Proc. 13th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, pp. 561–577, 2018.
[50] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3053–3062.
[51] P. Zajac and M. Jókay, "Multiplicative complexity of bijective 4×4 S-boxes," *Cryptogr. Commun.*, vol. 6, no. 3, pp. 255–277, Sep. 2014.
[52] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Proc. 36th Annu. Int. Cryptol. Conf.*, vol. 9815, 2016, pp. 123–153.
[53] D. Božilov, B. Bilgin, and H. A. Sahin, "A note on 5-bit quadratic permutations' classification," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 398–404, Mar. 2017.
[54] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science), vol. 4727. Berlin, Germany: Springer, 2007, pp. 450–466.
[55] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A block cipher for low energy," in *Proc. 21st Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, vol. 9453, 2015, pp. 411–436.
[56] A. Adomnicai, T. P. Berger, C. Clavier, J. Francq, P. Huynh, V. Lallemand, K. Le Gouguec, M. Minier, L. Reynaud, and G. Thomas, "Lilliput-AE: A new lightweight tweakable block cipher for authenticated encryption with associated data," in *Submission to the NIST Lightweight Cryptography Standardization Process*. NIST, 2019.

GIYOON KIM received the B.S. degree in information security, cryptology, and mathematics from Kookmin University, Seoul, South Korea, in 2019, where he is currently pursuing the joint M.S. and Ph.D. degrees in financial information security. His research interests include cryptographic primitives, cryptanalysis, artificial intelligence, and digital forensics.

HANGI KIM received the B.S. degree in mathematics and the M.S. degree in financial information security from Kookmin University, Seoul, South Korea, in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree in financial information security. His research interests include cryptographic primitives, cryptanalysis, and symmetric cryptosystems.

YEACHAN HEO is currently the Leader of the Reinforcement Learning Community. His research interests include deep reinforcement learning, information security, and financial optimization.



YONGJIN JEON received the B.S. degree in mathematics and the M.S. degree in financial information security from Kookmin University, Seoul, South Korea, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree in financial information security. His research interests include cryptographic primitives, cryptanalysis, and symmetric cryptosystems.

JONGSUNG KIM received the B.S. and M.S. degrees in mathematics from Korea University, South Korea, in 2000 and 2002, respectively, and the double Ph.D. degrees in combined differential, linear, and related-key attacks on block ciphers and MAC algorithms from the ESAT/COSIC Group, Katholieke Universiteit Leuven, Belgium, and in engineering in information security from Korea University, in 2006 and 2007, respectively. He is a Full Professor in cryptology and mathematics and of financial information security with the Department of Information Security, Kookmin University, South Korea. His research interests include cryptanalysis, symmetric cryptosystems, and digital forensics.