Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming*

Nicky Mouha[1,3,**], Qingju Wang[1,2,3], Dawu Gu[2], and Bart Preneel[1,3]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
[2] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[3] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium
{Nicky.Mouha,Qingju.Wang}@esat.kuleuven.be

Abstract. Differential and linear cryptanalysis are two of the most powerful techniques to analyze symmetric-key primitives. For modern ciphers, resistance against these attacks is therefore a mandatory design criterion. In this paper, we propose a novel technique to prove security bounds against both differential and linear cryptanalysis. We use mixed-integer linear programming (MILP), a method that is frequently used in business and economics to solve optimization problems. Our technique significantly reduces the workload of designers and cryptanalysts, because it only involves writing out simple equations that are input into an MILP solver. As very little programming is required, both the time spent on cryptanalysis and the possibility of human errors are greatly reduced. Our method is used to analyze Enocoro-128v2, a stream cipher that consists of 96 rounds. We prove that 38 rounds are sufficient for security against differential cryptanalysis, and 61 rounds for security against linear cryptanalysis. We also illustrate our technique by calculating the number of active S-boxes for AES.

Keywords: Differential cryptanalysis, Linear Cryptanalysis, Mixed-Integer Linear Programming, MILP, Enocoro, AES, CPLEX.

1 Introduction

Differential cryptanalysis [1] and linear cryptanalysis [19] have shown to be two of the most important techniques in the analysis of symmetric-key cryptographic

* This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II, and is funded by the National Natural Science Foundation of China (No. 61073150).
** This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

C.-K. Wu, M. Yung, and D. Lin (Eds.): Inscrypt 2011, LNCS 7537, pp. 57–76, 2012.

primitives. For block ciphers, differential cryptanalysis analyzes how input differences in the plaintext lead to output differences in the ciphertext. Linear cryptanalysis studies probabilistic linear relations between plaintext, ciphertext and key. If a cipher behaves differently from a random cipher for differential or linear cryptanalysis, this can be used to build a distinguisher or even a key-recovery attack.

For stream ciphers, differential cryptanalysis can be used in the context of a resynchronization attack [11]. In one possible setting, the same data is encrypted several times with the same key, but using a different initial value (IV). This is referred to as the standard (non-related-key) model, where the IV value is assumed to be under control of the attacker. An even stronger attack model is the related-key setting, where the same data is encrypted with different IVs *and* different keys. Not only the IV values, but also the differences between the keys are assumed to be under control of the attacker. Similar to differential cryptanalysis, linear cryptanalysis can also be used to attack stream ciphers in both the standard and related-key model. In the case of stream ciphers, linear cryptanalysis amounts to a known-IV attack instead of a chosen-IV attack.

Resistance against linear and differential cryptanalysis is a standard design criterion for new ciphers. For the block cipher AES [13], provable security against linear and differential cryptanalysis follows from the wide trail design strategy [12]. In this work, we apply a similar strategy. After proving a lower bound on the number of active S-boxes for both differential and linear cryptanalysis, we use the maximum differential probability (MDP) of the S-boxes to derive an upper bound for the probability of the best characteristic. We assume (as is commonly done) that the probability of the differential can accurately be estimated by the probability of the best characteristic. Several works focus on calculating the minimum number of active S-boxes for both Substitution-Permutation Networks (SPNs) [12] and (Generalized) Feistel Structures (GFSs) [5, 6, 16, 24]. Unfortunately, it seems that a lot of time and effort in programming is required to apply those techniques. This may explain why many related constructions have not yet been thoroughly analyzed. In this paper, we introduce a novel technique using mixed-integer linear programming in order to overcome these problems.

Linear programming (LP) is the study of optimizing (minimizing or maximizing) a linear objective function $f(x_1, x_2, \ldots, x_n)$, subject to linear inequalities involving decision variables x_i, $1 \leq i \leq n$. For many such optimization problems, it is necessary to restrict certain decision variables to integer values, i.e. for some values of i, we require $x_i \in \mathbb{Z}$. Methods to formulate and solve such programs are called mixed-integer linear programming (MILP). If all decision variables x_i must be integer, the term (pure) integer linear programming (ILP) is used. MILP techniques have found many practical applications in the fields of economy and business, but their application in cryptography has so far been limited. For a good introductory level text on LP and (M)ILP, we refer to Schrage [23].

In [7], Borghoff *et al.* transformed the quadratic equations describing the stream cipher Bivium into a MILP problem. The IBM ILOG CPLEX Optimizer[1] was then used to solve the resulting MILP problem, which corresponds to recovering the internal state of Bivium. In the case of Bivium A, solving this MILP problem takes less than 4.5 hours, which is faster than Raddum's approach (about a day) [22], but much slower than using MiniSAT (21 seconds) [9].

For the hash function SIMD, Bouillaguet *et al.* [8] used an ILP solver to find a differential characteristic based on local collisions. Using the SYMPHONY solver[2], they could not find the optimal solution, but found lower bounds for both SIMD-256 and SIMD-512. The computation for SIMD-512 took one month on a dual quad-core computer.

In [5, 6], Bogdanov calculated the minimum number of linearly and differentially active S-boxes of unbalanced Feistel networks with contracting MDS diffusion. He proved that some truncated difference weight distributions are impossible or equivalent to others. For the remaining truncated difference weight distributions, he constructed an ILP program which he then solved using the MAGMA[3] Computational Algebra System [4]. Compared to Bogdanov's technique, the fully automated method in this paper is much simpler to apply: Bogdanov's approach requires a significant amount of manual work, and the construction of not one but several ILP programs. We will show how this can be avoided by introducing extra dummy variables into the MILP program.

While this paper was under submission, Wu and Wang released a paper on ePrint [28] that also uses integer linear programming to count the number of active S-boxes for both linear and differential cryptanalysis. Just as in Bogdanov's approach, their algorithms require a large number of ILP programs to be solved, instead of only one as in the technique of this paper.

We apply our technique to the stream cipher Enocoro-128v2 [26, 27], in order to obtain bounds against differential and linear cryptanalysis. We consider both the standard and related-key model. All MILP programs are solved using CPLEX. There are 96 initialization rounds in Enocoro-128v2. We prove that 38 rounds are sufficient for security against differential cryptanalysis, and 61 rounds against linear cryptanalysis. These security bounds are obtained after 52.68 and 228.94 seconds respectively. We also calculate the minimum number of active S-boxes for up to 14 rounds of AES, which takes at most 0.40 seconds for each optimization program. Our experiments are performed on a 24-core Intel Xeon X5670 Processor, with 16 GB of RAM.

This paper is organized as follows. Sect. 2 explains how to find the minimum number of active S-boxes for a cryptographic primitive by solving an MILP program. A brief description of Enocoro-128v2 is given in Sect. 3. In Sect. 4 and Sect. 5, we construct an MILP program to prove that Enocoro-128v2 is secure against differential cryptanalysis and linear cryptanalysis respectively. We provide some ideas for future work in Sect. 6, and conclude the paper in

[1] `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`
[2] `http://projects.coin-or.org/SYMPHONY`
[3] `http://magma.maths.usyd.edu.au/`

Sect. 7. In App. A, we calculate the minimum number of active S-boxes for AES using our technique, and provide the full source code of our program.

2 Constructing an MILP Program to Calculate the Minimum Number of Active S-Boxes

We now explain a technique to easily prove the security of many ciphers against differential and linear cryptanalysis. Our method is based on counting the minimum number of active S-boxes. To illustrate our technique, we use Enocoro-128v2 and AES as test cases in this paper. The equations we describe are not specific to these ciphers, but can easily be applied to any cipher constructed using S-box operations, linear permutation layers, three-forked branches and/or XOR operations.

2.1 Differential Cryptanalysis

We consider truncated differences, that is, every byte in our analysis can have either a zero or a non-zero difference. More formally, we define the following difference vector:

Definition 1 *Consider a string Δ consisting of n bytes $\Delta = (\Delta_0, \Delta_1, \ldots, \Delta_{n-1})$. Then, the difference vector $x = (x_0, x_1, \ldots, x_{n-1})$ corresponding to Δ is defined as*

$$x_i = \begin{cases} 0 & \text{if } \Delta_i = 0 \ , \\ 1 & \text{otherwise} \ . \end{cases}$$

Equations Describing the XOR Operation. Let the input difference vector for the XOR operation be $(x^{\oplus}_{in_1}, x^{\oplus}_{in_2})$ and the corresponding output difference vector be $x^{\oplus}_{out}$. The differential branch number is defined as the minimum number of input and output bytes that contain differences, excluding the case where there are no differences in inputs nor outputs. For XOR, the differential branch number is 2. In order to express this branch number in equations, we need to introduce a new binary dummy variable $d^{\oplus}$.[4] If and only if all of the three variables $x^{\oplus}_{in_1}$,$x^{\oplus}_{in_2}$ and $x^{\oplus}_{out}$ are zero, $d^{\oplus}$ is zero, otherwise it should be one. Therefore we obtain the following linear equations (in binary variables) to describe the relation between the input and output difference vectors:

$$\begin{aligned} x^{\oplus}_{in_1} + x^{\oplus}_{in_2} + x^{\oplus}_{out} &\geq 2d^{\oplus} \ , \\ d^{\oplus} &\geq x^{\oplus}_{in_1} \ , \\ d^{\oplus} &\geq x^{\oplus}_{in_2} \ , \\ d^{\oplus} &\geq x^{\oplus}_{out} \ . \end{aligned}$$

[4] Note that this extra variable was not added in [5,6], which is why Bogdanov had to solve several ILP programs instead of only one.

Equations Describing the Linear Transformation. The equations for a linear transformation L can be described as follows. Assume L transforms the input difference vector $(x^L_{in_1}, x^L_{in_2}, \cdots, x^L_{in_\mathcal{M}})$ to the output difference vector $(x^L_{out_1}, x^L_{out_2}, \cdots, x^L_{out_\mathcal{M}})$. Given the differential branch number $\mathcal{B}_D$, a binary dummy variable d^L is again needed to describe the relation between the input and output difference vectors. The variable d^L is equal to 0 if all variables $x^L_{in_1}, x^L_{in_2}, \cdots, x^L_{in_\mathcal{M}}, x^L_{out_1}, x^L_{out_2}, \cdots, x^L_{out_\mathcal{M}}$ are 0, and 1 otherwise. Therefore the linear transformation L can be constrained by the following linear equations:

$$\begin{aligned} x^L_{in_1} + x^L_{in_2} + \cdots + x^L_{in_\mathcal{M}} + x^L_{out_1} + x^L_{out_2} + \cdots + x^L_{out_\mathcal{M}} &\geq \mathcal{B}_D d^L \ , \\ d^L &\geq x^L_{in_1} \ , \\ d^L &\geq x^L_{in_2} \ , \\ &\cdots\cdots \\ d^L &\geq x^L_{in_\mathcal{M}} \ , \\ d^L &\geq x^L_{out_1} \ , \\ d^L &\geq x^L_{out_2} \ , \\ &\cdots\cdots \\ d^L &\geq x^L_{out_\mathcal{M}} \ . \end{aligned}$$

The Objective Function. The objective function that has to be minimized, is the number of active S-boxes. This function is equal to the sum of all variables that correspond to the S-box inputs.

Additional Constraints. An extra linear equation is added to ensure that at least one S-box is active: this avoids the trivial solution where the minimum active S-boxes is zero. If all d-variables and all x-variables are restricted to be binary, the resulting program is a pure ILP (Integer Linear Programming) problem. If all d-variables are restricted to be binary, but only the x-variables corresponding to the input (plaintext), the equations ensure that the optimal solution for all other x-variables will be binary as well. This is similar to Borghoff's suggestion in [7], and results in an MILP (Mixed-Integer Linear Programming) problem that may be solved faster.

2.2 Linear Cryptanalysis

For linear cryptanalysis, we define a linear mask vector as follows:

Definition 2 *Given a set of linear masks $\Gamma = (\Gamma_0, \Gamma_1, \ldots, \Gamma_{n-1})$, the linear mask vector $y = (y_0, y_1, \ldots, y_{n-1})$ corresponding to Γ is defined as*

$$y_i = \begin{cases} 0 & \text{if } \Gamma_i = 0 \ , \\ 1 & \text{otherwise} \ . \end{cases}$$

The duality between differential and linear cryptanalysis was already pointed out by Matsui [20]. The equations describing a linear function are the same as in the case for differential cryptanalysis, however the differential branch number $\mathcal{B}_D$ is replaced by the linear branch number $\mathcal{B}_L$. The linear branch number is the minimum number of non-zero linear masks for the input and output of a function, excluding the all-zero case. No extra equations are introduced for the XOR operations, because the input and output linear masks are the same.

For a three-forked branch, we proceed as follows. Let the input linear mask vector for the three-forked branch be $y_{in}^{\vdash}$, and the corresponding output linear mask vector be $(y_{out_1}^{\vdash}, y_{out_2}^{\vdash})$. We introduce a binary dummy variable $l^{\vdash}$ to generate the following linear equations for the three-forked branch:

$$\begin{aligned} y_{in}^{\vdash} + y_{out_1}^{\vdash} + y_{out_2}^{\vdash} &\geq 2l^{\vdash} \ , \\ l^{\vdash} &\geq y_{in}^{\vdash} \ , \\ l^{\vdash} &\geq y_{out_1}^{\vdash} \ , \\ l^{\vdash} &\geq y_{out_2}^{\vdash} \ . \end{aligned}$$

3 Description of Enocoro-128v2

The first Enocoro specification was given in [25]. Enocoro is a stream cipher, inspired by the Panama construction [10]. Two versions of Enocoro were specified: Enocoro-80v1 with a key size of 80 bits, and Enocoro-128v1 with a key size of 128 bits. Later, a new version for the 128-bit key size appeared in [15]. It is referred to as Enocoro-128v1.1. We now give a short description of Enocoro-128v2. For more details, we refer to the design document [26, 27].

Internal State. The internal state of Enocoro-128v2 is composed of a buffer b consisting of 32 bytes $(b_0, b_1, \ldots, b_{31})$ and a state a consisting of two bytes (a_0, a_1). The initial state is loaded with a 128-bit key K and a 64-bit IV I as follows:

$$\begin{aligned} b_i^{(-96)} &= K_i, \qquad 0 \leq i < 16 \ , \\ b_{i+16}^{(-96)} &= I_i, \qquad 0 \leq i < 8 \ . \end{aligned}$$

All other internal state bytes are loaded with predefined constants.

Update Function. The update function $Next$ uses functions ρ and λ to update the internal state as follows:

$$(a^{(t+1)}, b^{(t+1)}) = Next(S^{(t)}) = (\rho(a^{(t)}, b^{(t)}), \lambda(a^{(t)}, b^{(t)})) \ .$$

An schematic overview of this function is given in Fig. 1.

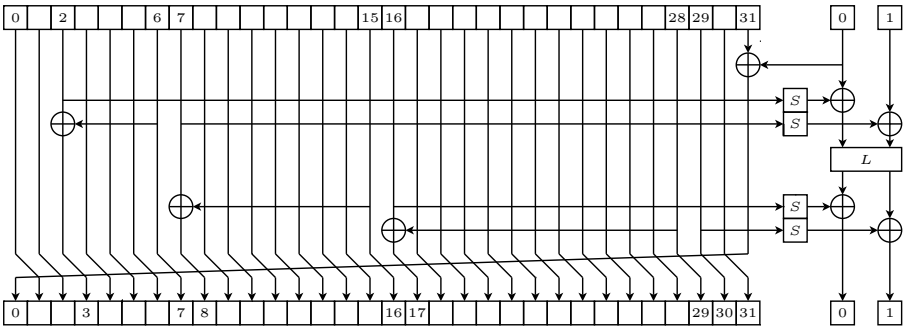

Fig. 1. State Update during the Initialization of Enocoro-128v2. Indices of buffer (on the left) refer to b-variables, indices of the state (on the right) refer to a-variables.

Function ρ. The function ρ updates the state a. It consists of an 8-bit S-box operation, a linear transformation L and XORs. The transformation L is defined as a linear transformation with a 2-by-2 matrix over $\mathrm{GF}(2^8)$:

$$\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = L(u_0, u_1) = \begin{pmatrix} 1 & 1 \\ 1 & d \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \ , \qquad d \in \mathrm{GF}(2^8) \ ,$$

where $d = \texttt{0x02}$, $u_0 = a_0^{(t)} \oplus S[b_2^{(t)}]$ and $u_1 = a_1^{(t)} \oplus S[b_7^{(t)}]$. The updated state $(a_0^{(t+1)}, a_1^{(t+1)})$ is then calculated as follows:

$$\begin{aligned} a_0^{(t+1)} &= v_0 \oplus S[b_{16}^{(t)}] \ , \\ a_1^{(t+1)} &= v_1 \oplus S[b_{29}^{(t)}] \ . \end{aligned}$$

Function λ. The λ function of Enocoro-128v2 consists of XOR operations and a byte-wise rotation of the buffer b. It is defined as follows:

$$b_i^{(t+1)} = \begin{cases} b_{31}^{(t)} \oplus a_0^{(t)}, & \text{if } i = 0 \ , \\ b_2^{(t)} \oplus b_6^{(t)}, & \text{if } i = 3 \ , \\ b_7^{(t)} \oplus b_{15}^{(t)}, & \text{if } i = 8 \ , \\ b_{16}^{(t)} \oplus b_{28}^{(t)}, & \text{if } i = 17 \ , \\ b_{i-1}^{(t)} & \text{otherwise} \ . \end{cases}$$

Output Function *Out*. After 96 initialization rounds, the Enocoro-128v2 output function outputs the lower byte of the state.

$$Out(S^{(t)}) = a_1^{(t)} \ .$$

Several results [14, 17, 18, 21, 27] on differential and linear cryptanalysis have already been published for different versions of Enocoro. In this paper, we consider the most recent version Enocoro-128v2 [26, 27] as an example to illustrate our technique. Watanabe *et al.* already showed that at least $2^{177.8}$ chosen IVs are required for a differential attack on Enocoro-128v2 [27]. For a linear attack, Konosu *et al.* [18] showed that 2^{216} known IVs are required for an attack on the 64-round variant Enocoro-128v1.1. Although these results are already sufficient to prove the security of Enocoro-128v2 against linear and differential cryptanalysis, we explain in this paper how to prove the security against these attacks in a much easier way.

4 Differential Cryptanalysis of Enocoro-128v2

Our technique is now used to find the minimum number of active S-boxes for the stream cipher Enocoro-128v2. We will consider an idealized variant of Enocoro-128v2, for which the minimum number of active S-boxes is a lower bound for the real Enocoro-128v2. In this idealized variant of Enocoro-128v2, the S-boxes can map any non-zero input difference to any non-zero output difference. The same holds for the L-function, with the restriction that the branch number is 3.

For this idealized variant of Enocoro-128v2, we have written a program to calculate the minimum number of active S-boxes. We present our problem as a mixed-integer linear programming (MILP) problem, and use CPLEX to solve it. The solution corresponds to the minimum number of differentially active S-boxes for Enocoro-128v2. It is used to prove the security of the cipher against differential cryptanalysis, using a similar proof as for the block cipher AES [12, 13]. Note that an actual characteristic with the given number of active S-boxes may or may not exist, depending on the specific S-box and L-function that is used. This is not a concern for us, as our goal is to prove a security bound against differential cryptanalysis.

4.1 Constructing the MILP Program

Enocoro-128v2 has eight XOR operations and one linear transformation L in each round. We represent the differential behavior of each of these operations by a set of linear equations, as described in Sect. 2. Let us take the first round of Enocoro-128v2 as an example. The initial difference vector in the buffer and states is represented by the binary variables $(x_0, x_1, \ldots, x_{31})$ and (x_{32}, x_{33}) respectively. Let us consider the XOR operation which has the rightmost byte of buffer b, i.e. b_{31}, and state byte a_0 as inputs. These correspond to binary variables x_{31} and x_{32} respectively, the input difference vector for this XOR operation. From the update function, we can obtain the corresponding value of the leftmost byte of buffer b, i.e. b_0, after the first round. Let the corresponding output difference vector be x_{34}, which is the first new binary variable that we

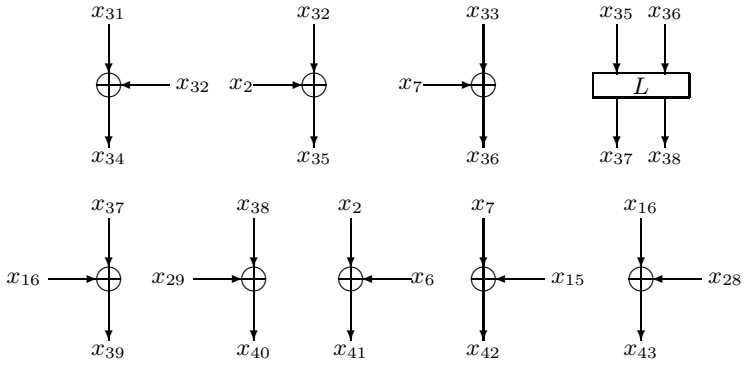

Fig. 2. Difference Vectors for Nine Operations in the First Round

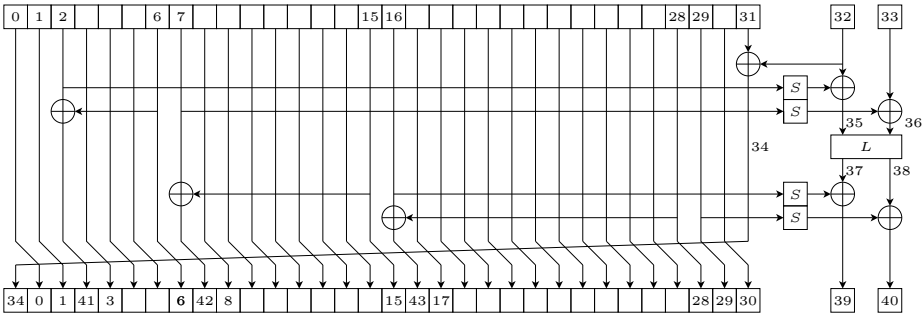

Fig. 3. Differential State Update during the Initialization of Enocoro-128v2. The indices refer to x-variables.

introduce. After introducing a binary dummy variable d_0, this XOR operation can be described by the equations:

$$\begin{aligned} x_{31} + x_{32} + x_{34} &\geq 2d_0 \ , \\ d_0 &\geq x_{31} \ , \\ d_0 &\geq x_{32} \ , \\ d_0 &\geq x_{34} \ . \end{aligned}$$

We now consider the second XOR operation, for which buffer b_2 (input to the first S-box) and the state a_0 are the inputs. Because the S-box is bijective, it is not only the case that the zero input difference results in a zero output difference, but also that a non-zero input difference results in a non-zero output difference. We find that (x_2, x_{32}) is the difference vector of the second XOR operation. The second new variable, x_{35}, will be the output difference vector for this second

XOR operation. Similarly, for the third XOR operation, the input difference vector is (x_7, x_{33}) (corresponding to (b_7, a_1)), and the output difference vector is x_{36}. Given two binary dummy variables d_1 and d_2 for the second and third XOR operation respectively, we again obtain four linear equations for every XOR operation.

From the structure of the linear transformation of Enocoro-128v2, we know that (x_{35}, x_{36}) is the input difference vector for the linear transformation L in the first round. By introducing a new binary variable d_3, the relations between the output difference vector (x_{37}, x_{38}) and the input difference vector (x_{35}, x_{36}) are easily described by the following equations:

$$\begin{aligned} x_{35} + x_{36} + x_{37} + x_{38} &\geq 3d_3 \ , \\ d_3 &\geq x_{35} \ , \\ d_3 &\geq x_{36} \ , \\ d_3 &\geq x_{37} \ , \\ d_3 &\geq x_{38} \ . \end{aligned}$$

The other five XORs in the first round are represented in a similar way. The new variables x_{39}, x_{40}, x_{41}, x_{42} and x_{43} are shown in Fig. 2. These equations result in the binary dummy variables d_4, d_5, d_6, d_7, d_8. For all the eight XORs and one linear transformation L, ten new binary variables $x_{34}, x_{35}, \ldots, x_{43}$ and nine binary dummy variables $d_0, d_1, \ldots, d_8$ are required. Therefore, a system of $4 \cdot 8 + 5 \cdot 1 = 37$ equations is obtained to describe all the nine operations in the first round (and also every subsequent round) of Enocoro-128v2. The detailed input and output vectors for all the nine operations are shown in Fig. 2.

After one round the difference vector for buffer and state will be

$$(x_{34}, x_0, x_1, x_{41}, x_3, \ldots, x_6, x_{42}, x_8, \ldots, x_{15}, x_{43}, x_{17}, \ldots, x_{30})$$

and (x_{39}, x_{40}) respectively. All binary x_i-variables obtained for the first round are illustrated in Fig. 3. Therefore, using this technique we can represent the differential update of Enocoro-128v2 for any round with a system of linear equations.

4.2 The Minimum Number of Active S-Boxes for Differential Cryptanalysis

We now focus on the variables that represent the S-box inputs in every round. Note that x_2, x_7, x_{16}, and x_{29} correspond to the input differences of the S-boxes, and therefore determine if the S-box is active or not. Let D_i include the four indices of variables that represent the four S-box inputs in the i-th round $(1 \leq i \leq 96)$. The 96 sets include the indices for variables that represent the four S-box inputs in each round. They can easily be obtained from Sect. 4.1, and are as follows:

$$
\begin{aligned}
D_1 &= \{2, 7, 16, 29\} \ , \\
D_2 &= \{1, 6, 15, 28\} \ , \\
D_3 &= \{0, 5, 14, 27\} \ , \\
D_4 &= \{34, 4, 13, 26\} \ , \\
D_5 &= \{44, 3, 12, 25\} \ , \\
&\vdots \\
D_{96} &= \{954, 941, 902, 863\} \ .
\end{aligned}
$$

Let k_N be the number of active S-boxes for N rounds of Enocoro-128v2. If

$$I_N = \bigcup_{1 \le i \le N} D_i \ ,$$

then

$$k_N = \sum_{i \in I_N} x_i$$

will be the number of active S-boxes in N rounds of Enocoro-128v2. To avoid the trivial case where no S-boxes are active, we add an extra linear constraint to specify that least one S-box is active. If we can minimize the linear function $k_N = \sum_{i \in I_N} x_i$, it will give us the minimum number of active S-boxes for N rounds of Enocoro-128v2. This will provide a security bound for Enocoro-128v2 against differential cryptanalysis. The objective function $k_N = \sum_{i \in I_N} x_i$ is a linear function, constrained by a system of $37N$ linear equations. If all variables must be binary variables, this corresponds to an ILP program.

It is easy to verify that the maximum differential probability for the 8-bit S-box of Enocoro-128v2 is $2^{-4.678}$. As the IV is limited to 64 bits, there are at most 2^{64} IV pairs for any given difference (if the key is fixed). Because there exists a generic attack with a data complexity of 2^{64} IVs (obtaining the entire codebook under one key), attacks requiring 2^{64} IVs or more should not be feasible. Therefore, we do not consider attacks using more than 2^{64} IVs, even in the related-key setting. If the number of active S-boxes in the initialization rounds is at least $14 > 64/4.678$, we consider the cipher to be resistant against differential cryptanalysis. Because we allow differences in both the key and the IV, our results hold both in the single-key and in the related-key setting. We note that typically, differential and linear cryptanalysis are used to attack a few more rounds than the number of rounds of the characteristic. The cipher must also be resistant against other types of attacks and add extra rounds to provide a security margin. For these reasons, more rounds should be used than suggested by our analysis.

In order to optimize the MILP program, we use CPLEX. The experiments are implemented on a 24-core Intel Xeon X5670 @ 2.93 GHz, with 16 GB of RAM. Because this computer is shared with other users, execution times may be longer

than necessary, which is why we do not give timing information for all problem instances. We found that it takes about 52.68 seconds to show that the minimum number of active S-boxes for 38 rounds of Enocoro-128v2 is 14. Therefore, 38 rounds of Enocoro-128v2 or more are secure against differential cryptanalysis. The minimum number of active S-boxes for each round of Enocoro-128v2 are listed in Table 1.

We would like to point out to the reader, that the seemingly complex bookkeeping of variable indices should not be a concern for the cryptanalyst who wishes to use this technique. The MILP linear equations can be generated by a small computer program. This program keeps track of the next unused x- and d-variables. It is then easy to replace every XOR and L function operation in the reference implementation of the cipher by a function to generate the corresponding equations, and every S-box application by a function that constructs the objective function. For a typical cipher, this should not require more than half an hour of work for a minimally experienced programmer.

If all d-variables are restricted to binary variables, as well as variables x_0 up to x_{33}, the equations ensure that the optimal solution for all other x_i-variables will be binary as well. Therefore, similar to Borghoff's suggestion in [7], we might solve an MILP program where only the d-variables and x_0 up to x_{33} are binary variables, instead of a pure ILP program. We find that Borghoff's observation can give dramatic speed-ups in some cases: for 72 rounds, it takes 5,808.15 seconds using an MILP, compared 342,747.78 seconds using a pure ILP. However, our MILP program for 38 rounds takes longer: 75.68 seconds instead of 52.68 seconds. Explaining this phenomenon seems to be a useful direction for future work.

5 Linear Cryptanalysis of Enocoro-128v2

We will use our technique to analyze an ideal variant of Enocoro-128v2 for linear cryptanalysis. Similarly as for differential cryptanalysis, the real Enocoro-128v2 will have at least as many linearly active S-boxes as the idealized one, and therefore can be used to prove a security bound.

5.1 Constructing the MILP Program

We now illustrate our technique by presenting the equations for the first round of the stream cipher Enocoro-128v2 for linear cryptanalysis. For the initial state, let the linear mask vector for the buffer be $(y_0, y_1, \ldots, y_{31})$, and for the state be (y_{32}, y_{33}). Consider the three-forked branch, which has the state byte a_0 as the input linear mask and buffer byte b_{31} as one output linear mask. We obtain the first new binary variable y_{34} as the other output vector. The input and output linear mask vector for this three-forked branch are then y_{32} and (y_{31}, y_{34}) respectively. By introducing the binary dummy variable l_0, the four equations describing the three-forked branch can be described as follows:

Table 1. Minimum Number of Differentially Active S-boxes $\min(k_N)$ for N rounds of Enocoro-128v2

N	$\min(k_N)$
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	2

N	$\min(k_N)$
21	2
22	3
23	3
24	3
25	4
26	5
27	7
28	8
29	8
30	8
31	8
32	9
33	9
34	10
35	11
36	12
37	13
38	14
39	15
40	15

N	$\min(k_N)$
41	16
42	17
43	18
44	18
45	18
46	19
47	20
48	20
49	21
50	22
51	22
52	22
53	22
54	22
55	22
56	22
57	23
58	23
59	24
60	24

N	$\min(k_N)$
61	25
62	26
63	27
64	27
65	28
66	29
67	30
68	30
69	30
70	31
71	32
72	34
73	35
74	35
75	36
76	37
77	37
78	38
79	38
80	38

N	$\min(k_N)$
81	39
82	39
83	40
84	40
85	40
86	41
87	42
88	43
89	43
90	44
91	44
92	45
93	45
94	46
95	47
96	47

$$\begin{aligned} y_{31} + y_{32} + y_{34} &\geq 2l_0 \ , \\ l_0 &\geq y_{31} \ , \\ l_0 &\geq y_{32} \ , \\ l_0 &\geq y_{34} \ . \end{aligned}$$

For the XOR operation, the two inputs and the output all have the same linear mask. The bijectiveness of the S-box implies the linear mask at the output will be non-zero if and only if the input mask is non-zero. Therefore, the linear transformation L has an input linear mask vector of (y_{34}, y_{33}), and an output linear mask vector of (y_{35}, y_{36}). Using a new binary dummy variable l_1, the equations describing the L transformation are:

$$\begin{aligned} y_{34} + y_{33} + y_{35} + y_{36} &\geq 3l_1 \ , \\ l_1 &\geq y_{34} \ , \\ l_1 &\geq y_{33} \ , \\ l_1 &\geq y_{35} \ , \\ l_1 &\geq y_{36} \ . \end{aligned}$$

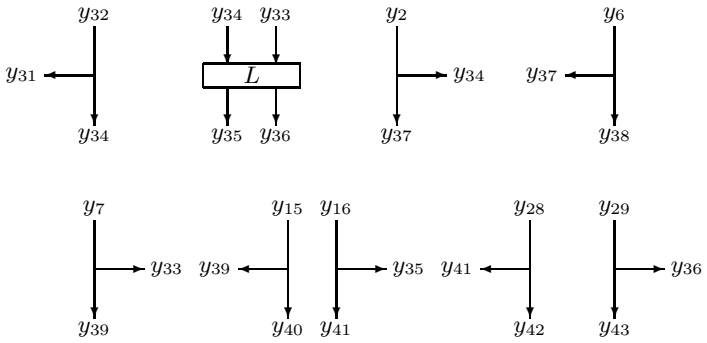

Fig. 4. Linear Mask Vectors for Nine Operations in the First Round

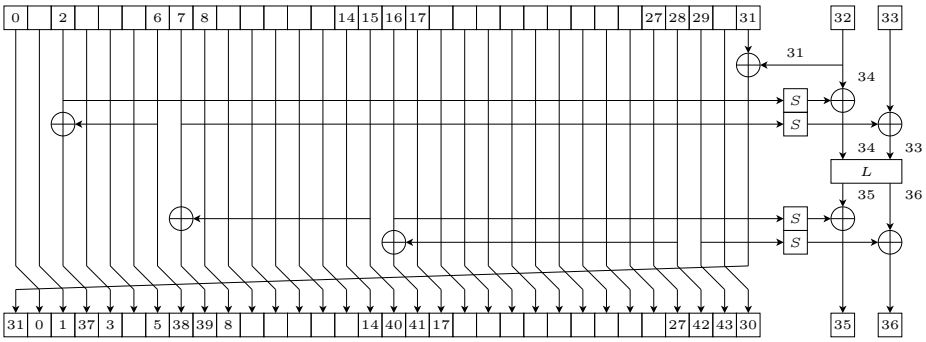

Fig. 5. Linear Mask Vectors Update during the Initialization of Enocoro-128v2. The indices refer to y-variables.

As an Enocoro-128v2 round contains eight three-forked branch operations and one linear transformation L, ten new binary variables $y_{34}, y_{35}, \ldots, y_{43}$, as well as nine binary dummy variables $l_0, l_1, \ldots, l_8$ are introduced. Therefore, $4 \cdot 8 + 5 \cdot 1 = 37$ equations are required to describe the propagation of linear masks for the first round (as well as any subsequent round) of Enocoro-128v2. The input and output linear mask vectors for all nine operations in the first round are shown in Fig. 4. The linear mask vector for the buffer and state after one round are

$$(y_{31}, y_0, y_1, y_{37}, y_3, \cdots, y_5, y_{38}, y_{39}, y_8, \cdots, y_{14}, y_{40}, y_{41}, y_{17}, \cdots, y_{27}, y_{42}, y_{43}, y_{30})$$

and (y_{35}, y_{36}) respectively. They are shown in Fig. 5.

5.2 The Minimum Number of Active S-Boxes for Linear Cryptanalysis

Using the technique in the previous section, we can represent any number of rounds of Enocoro-128v2. We now explain how to calculate the number of active S-boxes. Let L_i include all indices of the four variables representing the input linear mask vector of S-boxes in the i-th round ($1 \leq i \leq 96$). We then obtain the following 96 sets:

$$\begin{aligned} L_1 &= \{34, 33, 35, 36\} \ , \\ L_2 &= \{44, 36, 45, 46\} \ , \\ L_3 &= \{54, 46, 55, 56\} \ , \\ L_4 &= \{64, 56, 65, 66\} \ , \\ L_5 &= \{74, 66, 75, 76\} \ , \\ &\vdots \\ L_{96} &= \{984, 976, 985, 986\} \ . \end{aligned}$$

Let m_N be the number of active S-boxes for N rounds of Enocoro-128v2. If

$$J_N = \bigcup_{1 \leq j \leq N} L_j \ ,$$

then

$$m_N = \sum_{j \in J_N} y_j$$

will be the number of active S-boxes for N rounds of Enocoro-128v2. By minimizing the linear objective function m_N, we obtain the minimum number of linearly active S-boxes for N rounds of Enocoro-128v2.

The maximum correlation amplitude of the 8-bit S-box of Enocoro-128v2 is $C_{\max} = 2^{-2}$. For the same reasons as for differential cryptanalysis, we limit the number of IVs to 2^{64}. Let us denote the minimum number of active S-boxes by a. From the limit on the number of IVs, we then find that resistance against linear cryptanalysis requires [13, pp. 142–143]:

$$C_{\max}^a = (2^{-2})^a \leq 2^{-64/2} \ .$$

This inequality is satisfied for $a \geq 16$. Therefore, if the number of linearly active S-boxes is at least 16, Enocoro-128v2 can be considered to be resistant against linear cryptanalysis (in both the single-key and related-key setting).

If we solve the resulting MILP problem using CPLEX, we find that the minimum number of active S-boxes is 18 for 61 rounds of Enocoro-128v2. This result was obtained after 227.38 seconds. Therefore, we conclude that Enocoro-128v2 with 96 initialization rounds is secure against linear cryptanalysis (in both the single-key and related-key setting). The minimum number of active S-boxes for Enocoro-128v2 are listed in Table 2.

Table 2. Minimum Number of Linearly Active S-boxes $\min(m_N)$ for Enocoro-128v2

N	$\min(m_N)$
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0

N	$\min(m_N)$
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0
32	0
33	3
34	6
35	6
36	6
37	6
38	6
39	6
40	6

N	$\min(m_N)$
41	6
42	9
43	9
44	9
45	12
46	12
47	12
48	12
49	12
50	12
51	12
52	15
53	15
54	15
55	15
56	15
57	15
58	15
59	15
60	15

N	$\min(m_N)$
61	18
62	18
63	18
64	18
65	18
66	18
67	18
68	21
69	21
70	21
71	21
72	21
73	21
74	21
75	21
76	24
77	24
78	24
79	24
80	24

N	$\min(m_N)$
81	24
82	27
83	27
84	27
85	27
86	27
87	27
88	27
89	27
90	27
91	27
92	27
93	30
94	30
95	33
96	33

6 Future Work

It is interesting to investigate how the internal parameters of CPLEX can be fine-tuned to calculate bounds against linear and differential cryptanalysis in the fastest possible time. If there are symmetries in the round function, these may be used to speed up the search as well. Similarly, the attacker may improve a given (suboptimal) lower bound for a particular cipher by clocking the round functions forward or backward in order to obtain a lower number of S-boxes. To obtain a rough lower bound for a large number of rounds, the "split approach" (see for example [3]) may be used. For example, if r rounds of a cipher contain at least a active S-boxes, then kr rounds of a cipher must contain at least ka active S-boxes. It is useful to explore how these observations can be applied when CPLEX takes a very long time to execute. Otherwise, the shorter solving time does not compensate for the additional time to construct the program. For ILP programs with a very long execution time, it may be better to calculate the minimum number of active S-boxes using a different technique (e.g. [3]).

The technique in this paper is quite general, and may also be used for truncated differentials, higher-order differentials, impossible differentials, saturation attacks,... It can also be applied to other ciphers constructed using S-box operations, linear permutation layers, three-forked branches and/or XOR operations. We leave the exploration of these topics to future work as well.

7 Conclusion

In this paper, we introduced a simple technique to calculate the security of many ciphers against linear and differential cryptanalysis. The only requirement is that the cipher is composed of a combination of S-box operations, linear permutation layers and/or XOR operations. Our technique involves writing a simple program to generate a mixed-integer linear programming (MILP) problem. The objective function of the MILP program is the number of linearly or differentially active S-boxes, which we want to minimize. This MILP problem can then easily be solved using an off-the-shelf optimization package, for example CPLEX. The result can be used to prove the security of a cryptosystem against linear and differential cryptanalysis.

Our technique can be applied to a wide variety of cipher constructions. As an example, we apply the technique in this paper to the stream cipher Enocoro-128v2. We prove that for Enocoro-128v2 38 rounds are sufficient for security against differential cryptanalysis, and 61 rounds against linear cryptanalysis. These results are valid both in the single-key and related-key models. As Enocoro-128v2 consists of 96 initialization rounds, this proves the security of Enocoro-128v2 against linear and differential cryptanalysis.

We would like to point out that only little programming is required to obtain this result. A minimally experienced programmer can modify the reference implementation of a cipher, in order to generate the required MILP program in about half an hour. In the case of Enocoro-128v2, it takes CPLEX less than one minute on a 24-core Intel Xeon X5670 processor to prove security against differential cryptanalysis, and less than four minutes to prove security against linear cryptanalysis. We note that because very little programming is required, both the time spent on cryptanalysis and the possibility of making errors are greatly reduced.

Acknowledgments. The authors would like to thank their colleagues at COSIC, as well as the anonymous reviewers for their detailed comments and suggestions. Special thanks to Hirotaka Yoshida for reviewing an earlier draft of this paper.

References

1. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology 4(1), 3–72 (1991)
2. Biryukov, A., Gong, G., Stinson, D.R. (eds.): SAC 2010. LNCS, vol. 6544. Springer, Heidelberg (2011)
3. Biryukov, A., Nikolić, I.: Search for Related-Key Differential Characteristics in DES-Like Ciphers. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 18–34. Springer, Heidelberg (2011)
4. Bodganov, A.: Personal Communication (2011)
5. Bogdanov, A.: Analysis and Design of Block Cipher Constructions. Ph.D. thesis, Ruhr University Bochum (2009)
6. Bogdanov, A.: On unbalanced Feistel networks with contracting MDS diffusion. Des. Codes Cryptography 59(1-3), 35–58 (2011)

7. Borghoff, J., Knudsen, L.R., Stolpe, M.: Bivium as a Mixed-Integer Linear Programming Problem. In: Parker, M.G. (ed.) Cryptography and Coding 2009. LNCS, vol. 5921, pp. 133–152. Springer, Heidelberg (2009)
8. Bouillaguet, C., Fouque, P.A., Leurent, G.: Security Analysis of SIMD. In: Biryukov, et al. (eds.) [2], pp. 351–368
9. McDonald, C., Charnes, C., Pieprzyk, J.: An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem. Cryptology ePrint Archive, Report 2007/129 (2007), http://eprint.iacr.org/
10. Daemen, J., Clapp, C.S.K.: Fast Hashing and Stream Encryption with PANAMA. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 60–74. Springer, Heidelberg (1998)
11. Daemen, J., Govaerts, R., Vandewalle, J.: Resynchronization Weaknesses in Synchronous Stream Ciphers. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 159–167. Springer, Heidelberg (1994)
12. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
13. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
14. Hell, M., Johansson, T.: Security Evaluation of Stream Cipher Enocoro-128v2. CRYPTREC Technical Report (2010)
15. Muto, K., Watanabe, D., Kaneko, T.: Strength evaluation of Enocoro-128 against LDA and its Improvement. In: Symposium on Cryptography and Information Security, pp. 4A1–1 (2008) (in Japanese)
16. Kanda, M.: Practical Security Evaluation against Differential and Linear Cryptanalyses for Feistel Ciphers with SPN Round Function. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 324–338. Springer, Heidelberg (2001)
17. Okamoto, K., Muto, K., Kaneko, T.: Security evaluation of Pseudorandom Number Generator Enocoro-80 against Differential/Linear Cryptanalysis (II). In: Symposium on Cryptography and Information Security, pp. 20–23 (2009) (in Japanese)
18. Konosu, K., Muto, K., Furuichi, H., Watanabe, D., Kaneko, T.: Security evaluation of Enocoro-128 ver.1.1 against resynchronization attack. IEICE Technical Report, ISEC2007-147 (2008) (in Japanese)
19. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
20. Matsui, M.: On Correlation between the Order of S-Boxes and the Strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995)
21. Muto, K., Watanabe, D., Kaneko, T.: Security evaluation of Enocoro-80 against linear resynchronization attack. In: Symposium on Cryptography and Information Security (2008) (in Japanese)
22. Raddum, H.: Cryptanalytic Results on Trivium. eSTREAM report 2006/039 (2006), http://www.ecrypt.eu.org/stream/triviump3.html
23. Schrage, L.: Optimization Modeling with LINGO. Lindo Systems (1999), http://www.lindo.com
24. Shibutani, K.: On the Diffusion of Generalized Feistel Structures Regarding Differential and Linear Cryptanalysis. In: Biryukov, et al. (eds.) [2], pp. 211–228
25. Watanabe, D., Kaneko, T.: A construction of light weight Panama-like keystream generator. IEICE Technical Report, ISEC2007-78 (2007) (in Japanese)
26. Watanabe, D., Okamoto, K., Kaneko, T.: A Hardware-Oriented Light Weight Pseudo-Random Number Generator Enocoro-128v2. In: The Symposium on Cryptography and Information Security, pp. 3D1–3 (2010) (in Japanese)

27. Watanabe, D., Owada, T., Okamoto, K., Igarashi, Y., Kaneko, T.: Update on Enocoro Stream Cipher. In: ISITA, pp. 778–783. IEEE (2010)
28. Wu, S., Wang, M.: Security evaluation against differential cryptanalysis for block cipher structures. Cryptology ePrint Archive, Report 2011/551 (2011), http://eprint.iacr.org/

A Number of Active S-Boxes for AES

The four-round propagation theorem of AES [13] proves that the number of active S-boxes in a differential or linear characteristic of four AES rounds is at least 25. Combined with the properties of the AES S-box, this result was used in the AES design document to prove the resistance against linear and differential attacks. In this section, we illustrate our technique by applying it to the block cipher AES. We not only confirm the four-round propagation theorem, but also determine the minimum number of active S-boxes for up to 14 rounds in Table 4.

An AES round update consists of four operations: AddRoundKey (AR), SubBytes (SB), ShiftRows (SR) and MixColumns (MC). The update of the first AES round is shown in Table 3. Every variable corresponds to a byte of the AES state. The variable is 1 if the difference is non-zero, and 0 if the difference is zero. All variables corresponding to the inputs of the SubByte operations are summed in the objective function, this corresponds to the number of active S-boxes. The linear function used in the MixColumns operation has a differential as well as a linear branch number of 5.

A program was written in C to generate the equations for this optimization problem in the CPLEX LP format. To illustrate the simplicity of our technique, we provide this program (including source code comments) below in full. None of the optimization problems in Table 4 took longer than 0.40 seconds to solve, using only a single core of our 24-core Intel Xeon X5670 processor.

Table 3. The Variables in the First Round Update of AES

$$\begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix} \xrightarrow{\text{SB}} \begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix} \xrightarrow{\text{SR}} \begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_5 & x_9 & x_{13} & x_1 \\ x_{10} & x_{14} & x_2 & x_6 \\ x_{15} & x_3 & x_7 & x_{11} \end{bmatrix} \xrightarrow{\text{MC}} \begin{bmatrix} x_{16} & x_{20} & x_{24} & x_{28} \\ x_{17} & x_{21} & x_{25} & x_{29} \\ x_{18} & x_{22} & x_{26} & x_{30} \\ x_{19} & x_{23} & x_{27} & x_{31} \end{bmatrix}$$

Table 4. Minimum Number of Differentially or Linearly Active S-boxes $\min(k_N)$ for N rounds of AES

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\min(k_N)$	1	5	9	25	26	30	34	50	51	55	59	75	76	80

```
#include <stdio.h>
int i,j,r;
const int ROUNDS = 4; /* number of rounds */
int next = 0; /* next unused state variable index */
int dummy = 0; /* next unused dummy variable index */

void ShiftRows(int a[4][4]) {
  int tmp[4];
  for(i = 1; i < 4; i++) {
    for(j = 0; j < 4; j++) tmp[j] = a[i][(j + i) % 4];
    for(j = 0; j < 4; j++) a[i][j] = tmp[j];
  }
}
void MixColumn(int a[4][4]) {
  for(j = 0; j < 4; j++) {
    for (i = 0; i < 4; i++) printf("x%i + ",a[i][j]);
    for (i = 0; i < 3; i++) printf("x%i + ",next+i);
    printf("x%i - 5 d%i >= 0\n",next+3,dummy);

    for(i = 0; i < 4; i++)
      printf("d%i - x%i >= 0\n",dummy,a[i][j]);
    for(i = 0; i < 4; i++)
      printf("d%i - x%i >= 0\n",dummy,a[i][j]=next++);
    dummy++;
  }
}
int main() {
  int a[4][4]; /* the bytes of the AES state */
  for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
      a[i][j] = next++; /* initialize variable indices */

  printf("Minimize\n"); /* print objective function */
  for (i = 0; i < ROUNDS*16-1; i++) printf("x%i + ",i);
  printf("x%i\n\n",ROUNDS*16-1);

  printf("Subject To\n"); /* round function constraints */
  for (r = 0; r<ROUNDS; r++) { ShiftRows(a); MixColumn(a); }

  /* at least one S-box must be active */
  for (i = 0; i < ROUNDS*16-1; i++) printf("x%i + ",i);
  printf("x%i >= 1\n\n",ROUNDS*16-1);

  printf("Binary\n"); /* binary constraints */
  for (i = 0; i < 16; i++) printf("x%i\n",i);
  for (i = 0; i < dummy; i++) printf("d%i\n",i);
  printf ("End\n");
  return 0;
}
```