IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 2, pp. 113–140. DOI:10.46586/tosc.v2022.i2.113-140

# Automatic Search of Rectangle Attacks on Feistel Ciphers: Application to WARP

Virginie Lallemand[1], Marine Minier[1] and Loïc Rouquette[2,3]

[1] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
firstname.name@loria.fr
[2] CITI, INRIA, INSA Lyon, Villeurbanne, France
loic.rouquette@insa-lyon.fr
[3] LIRIS, UMR5201 CNRS, Villeurbanne, France

**Abstract.** In this paper we present a boomerang analysis of `WARP`, a recently proposed Generalized Feistel Network with extremely compact hardware implementations. We start by looking for boomerang characteristics that directly take into account the boomerang switch effects by showing how to adapt Delaune *et al.* automated tool to the case of Feistel ciphers, and discuss several improvements to keep the execution time reasonable. This technique returns a 23-round distinguisher of probability $2^{-124}$, which becomes the best distinguisher presented on `WARP` so far. We then look for an attack by adding the key recovery phase to our model and we obtain a 26-round rectangle attack with time and data complexities of $2^{115.9}$ and $2^{120.6}$ respectively, again resulting in the best result presented so far. Incidentally, our analysis discloses how an attacker can take advantage of the position of the key addition (put after the S-box application to avoid complementation properties), which in our case offers an improvement of a factor of $2^{75}$ of the time complexity in comparison to a variant with the key addition positioned before. Note that our findings do not threaten the security of the cipher which iterates 41 rounds.

**Keywords:** Cryptanalysis · Feistel cipher · Boomerang attack · WARP

## 1 Introduction

Boomerang distinguishers [Wag99] were introduced at FSE'99 as a variant of differential distinguishers [BS91] taking advantage of the existence of short differentials of high probability. In its simplest version, the attacker sees the cipher $E$ as the composition of two subciphers ($E = E_1 \circ E_0$) and makes use of a differential for each part. If at first it was thought that these two differentials can be selected freely, following advances like [Mur11] showed that the interdependence should be carefully treated, as incompatibilities or better-than-expected probabilities might occur.

As a result, searching for the best boomerang distinguisher does not simply reduce to finding two differentials of high probability, and thus emerged a need for automated tools that would take into account the possible events in the middle, later formalized by the BCT [CHP+18] (for SPNs) and FBCT [BHL+20] (for Feistels) theories. Two techniques have been proposed recently to address this issue. In [HBS21], the authors proposed to give as input to a MILP model the size of the middle part (where dependencies happen) and to take into account one type of dependency (the so-called ladder switch [BK09]). A more precise approximation of the probability of the middle rounds was then obtained with the BCT framework or experimentally. A second technique has been proposed in [DDV20], that directly takes into account all the possible middle dependencies and does not require that the attacker specifies the size of the middle part.

Received: 2021-11-23 Revised: 2022-03-01 Accepted: 2022-05-01 Published: 2022-06-10

If many papers start by looking for the best distinguisher to next turn it into an attack, better results might be obtained by taking into account the incidence of the distinguisher on the key recovery phase. An example of this was given in [ZDJ19] when building a rectangle attack on Deoxys, and further discussions were provided in [QDW+21] with results on Skinny. The latter presents an automated tool that takes the model provided in [DDV20] and adds relations to include the dominating factors of the key-recovery phase so that the resulting model directly looks for an optimization of the attack as a whole.

**Our contribution.** In this work, we propose to study the security of the recently published block cipher `WARP` [BBI+20] against boomerang techniques. To do so, we start by showing how to adapt the model developed in [DDV20] to the case of Feistel ciphers, since the original tool was developed for SPN ciphers in general and for Skinny in particular. Since the execution time of the simple model is exponential in the number of rounds, covering more than 20 rounds of `WARP` is out of reach. We thus propose several techniques to speed up the model and to guide it to the solutions. By counting different solutions with the same input and output differences, we were able to find a 23-round distinguisher that covers 2 more rounds than the best result to date.

Second, we show how to extend this model to search for rectangle attacks, following the method developed in [QDW+21]. This extra step ensures that both the key recovery part and the distinguisher are optimized together to reach a (close to) optimal attack as a whole. Finally, we describe a 26-round attack on `WARP`, again reaching the best result to date. Our analysis shows that the designers' choice of positioning the key addition after the S-box in the Feistel round function (which is justified by the need to avoid complementation properties) turns in favour of the attacker.

Our results on `WARP` with a comparison with previous works are summarized in Table 1. The code of our tool is available at:

https://gitlab.inria.fr/lrouquet/boomerang-warp-fse-23

**Outline.** We start by recalling some preliminaries in Section 2 which include the specification of `WARP`, a reminder on boomerang attacks and a brief overview of the existing techniques to automatically find them. Section 3 is dedicated to the description of our model searching for boomerang distinguishers and to the discussion of our result on 23 rounds, that we can easily extend by 2 rounds. Our techniques to improve the execution time of the model are presented. Section 4 shows how to turn the previous model into one searching for rectangle attacks and in particular how the position of the key addition turns favourable to the attacker, leading us to a 26-round attack.

## 2 Preliminaries

### 2.1 Specification of `WARP`

`WARP` [BBI+20] is a lightweight block cipher that has been recently presented at SAC 2020 by Banik *et al.* The main objective of the designers was to propose a cipher that could be used as a direct replacement of AES-128 (thus with a 128-bit block and key) but that would be lighter in terms of hardware footprint. This challenge was met with flying colours as evidenced by the impressive reported number of around 800 Gate Equivalents (GEs) for a serialized circuit of `WARP`.

**Description.** The cipher follows a variant of a Type-2 Generalized Feistel Network (GFN) using 32 branches of 4 bits each. Special care was taken to the selection of the 32-branch

**Table 1:** Complexities of the existing results on WARP. Note that for all the distinguishers presented here we can add 2 rounds for free, see Section 3.4. ID = Impossible Differential, DC = Differential Characteristic.

| Technique | Rounds | Probability | Time | Data | Mem. | Ref. |
|---|---|---|---|---|---|---|
| DC distinguisher | 18 | $2^{-122}$ | - | - | - | [KY20] |
| DC distinguisher | 20 | $2^{-122.71}$ | - | - | - | [TB21] |
| ID distinguisher | 21 | 1 | - | - | - | [BBI+20] |
| Boomerang distinguisher | 21 | $2^{-121.11}$ | - | - | - | [TB21] |
| Boomerang distinguisher | 23 | $2^{-124}$ | - | - | - | Section 3.4 |
| Differential attack | 21 | - | $2^{113}$ | $2^{113}$ | $2^{72}$ | [KY20] |
| Differential attack | 23 | - | $2^{106.68}$ | $2^{106.62}$ | $2^{106.62}$ | [TB21] |
| Rectangle attack | 24 | - | $2^{125.18}$ | $2^{126.06}$ | $2^{127.06}$ | [TB21] |
| Rectangle attack | 26 | - | $2^{115.9}$ | $2^{120.6}$ | $2^{120.6}$ | Section 4.3 |

permutation $\pi$ in order to optimize both the diffusion and the number of active S-boxes in a differential or linear trail. The cipher iterates 41 rounds, where the final round misses $\pi$.

In detail, the 128-bit internal state is split over 32 branches of 4 bits. At the input of round $r$, the value of the 32 nibbles is denoted $X[r, 0]$ to $X[r, 31]$. They go through five elementary mappings in each (full) round, as depicted in Figure 1. Each nibble with an even index $X[r, 2i]$ is modified by the $F$ function, which consists in the application of a 4-bit S-box (denoted SBOX in the following, and given in Table 2) followed by a round key addition. The result is then xored to $X[r, 2i+1]$, a constant is added to $X[r, 1]$ and $X[r, 3]$ and finally the 32 branches are shuffled by the $\pi$ permutation given in Table 3. Since the values of the round constants have no impact on our analysis we refer the reader to the specification [BBI+20].

The key schedule is linear and relies on a 128-bit master key seen as the concatenation of two 64-bit keys: $K = K0||K1$. Each half is used alternatively as the round key, starting with the 16 nibbles of $K0$ that are used in the first round.

**Table 2:** 4-bit S-box SBOX.

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SBOX (x) | c | a | d | 3 | e | b | f | 7 | 8 | 9 | 1 | 5 | 0 | 2 | 4 | 6 |

**Table 3:** Shuffle $\pi$ mixing the 32 branches.

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi$(x) | 31 | 6 | 29 | 14 | 1 | 12 | 21 | 8 | 27 | 2 | 3 | 0 | 25 | 4 | 23 | 10 |
| x | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| $\pi$(x) | 15 | 22 | 13 | 30 | 17 | 28 | 5 | 24 | 11 | 18 | 19 | 16 | 9 | 20 | 7 | 26 |

**Security.** The designers of WARP claimed single-key security and did not claim any security in related-key and known/chosen-key settings. They provided a rather comprehensive security analysis of their design, with a discussion of differential, linear, impossible differential, integral, meet-in-the middle and invariant subspace attacks. The longer distinguisher they mentioned is a 21-round impossible differential distinguisher.

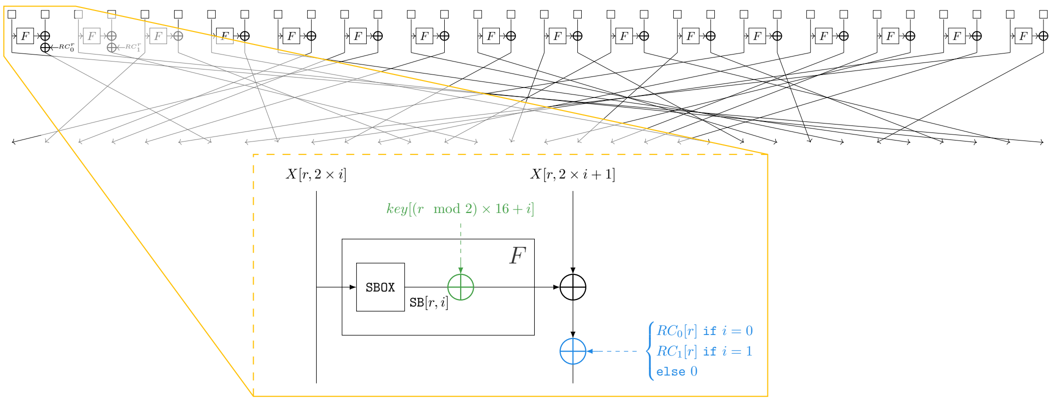


**Figure 1:** One round of `WARP`. The constant addition in blue (—) plays no role when searching for differential properties, and the round key addition in green (—) can be ignored when considering the single key scenario.

To the best of our knowledge, two external cryptanalyses have been reported to date, both studying differential-based attacks. In the article [KY20] by Kumar and Yadav, a 21-round differential attack is presented (based on a 16-round differential characteristic), with a time and data complexity of $2^{113}$. Concurrently with our work, a 23-round differential attack and a 24-round rectangle attack were reported by Teh and Biryukov in [TB21].

## 2.2 Boomerang Attacks

The boomerang attack is a variant of the differential attack that was introduced by David Wagner in 1999 [Wag99]. The boomerang distinguishers at their basis are defined by two differences $\alpha$ and $\delta$ chosen so that the probability of the following equality is higher for the (round-reduced) cipher $E$ than for a random permutation:

$$E^{-1}(E(X) \oplus \delta) \oplus E^{-1}(E(X \oplus \alpha) \oplus \delta) = \alpha \tag{1}$$

In their basic form, boomerang distinguishers are built by rewriting $E$ as the composition of two sub-ciphers; $E = E_1 \circ E_0$ and by finding a good differential for each part. If we denote by $p$ the probability of the differential over $E_0$, that is $p = \Pr(\alpha \rightarrow_{E_0} \beta)$ and by $q$ the probability of the second differential over $E_1$ ($q = \Pr(\delta \rightarrow_{E_1^{-1}} \gamma)$), a first approximation returns that the probability of Equation (1) is close to $p^2q^2$.

Kelsey *et al.* [KKS01] and Biham *et al.* [BDK01] independently introduced a chosen plaintext-only version of the boomerang distinguisher, that they respectively named the *amplified* and the *rectangle* technique. This variant relies on the same rewriting of the cipher and on the same differentials as previously and consists in observing twice the difference $\delta$ in the quartets at the output of the cipher. The distinguisher is expected to have a probability of $p^2q^22^{-n}$, while it would be $2^{-2n}$ for a n-bit random permutation.

As shown for instance in the analysis of Sean Murphy [Mur11], the naive approximation of the probability of a boomerang distinguisher might turn wrong due to an incompatibility between the upper and the lower differentials. To solve this problem, Dunkelman *et al.* introduced the sandwich attack [DKS10] which adds a middle part $E_m$ in the rewriting of $E$ (namely $E = E_1 \circ E_m \circ E_0$) to isolate and study separately the rounds where the two differentials are interdependent. This middle part is called boomerang switch [BK09]; if we denote by $r$ the probability that $E_m$ connects the upper and the lower trails, the final probability of the distinguisher becomes $p^2q^2r$.

**Computing the middle part probability.** A recent line of works showed how to approximate the value of $r$ with the use of various tables. The first to be introduced is the BCT [CHP+18], developed by Cid *et al.* to deal with 1-round boomerang switches in the case of SPN ciphers. In this paper we focus on the Feistel case and thus recall the FBCT, the FBDT and the FBET, introduced in [BHL+20]. The notation is recalled in Figure 2.

**Definition 1** (FBCT, FBDT and FBET [BHL+20])**.** Let $S$ be a function from $\mathbb{F}_2^n$ to itself, and $(\Delta_i, \delta, \nabla_o, \alpha)$ be elements of $(\mathbb{F}_2^n)^4$. The Feistel Boomerang Connectivity Table (FBCT), Feistel Boomerang Difference Table (FBDT) and Feistel Boomerang Extended Table (FBET) of $S$ are given by:

$$FBCT(\Delta_i, \nabla_o) = \# \{x \in \mathbb{F}_2^n \,|S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\}.$$

$$FBDT(\Delta_i, \delta, \nabla_o) = \# \left\{ x \in \mathbb{F}_2^n \middle| \begin{array}{l} S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0, \\ S(x) \oplus S(x \oplus \Delta_i) = \delta. \end{array} \right\}.$$

$$FBET(\Delta_i, \delta, \nabla_o, \alpha) = \# \left\{ x \in \mathbb{F}_2^n \middle| \begin{array}{l} S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0, \\ S(x) \oplus S(x \oplus \Delta_i) = \delta, \\ S(x \oplus \Delta_i) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = \alpha. \end{array} \right\}.$$

The table used to compute the 1-round probability depends on which input and output differences of the S-box are fixed: for instance, the FBCT is chosen when only the inputs $\Delta_i$ and $\nabla_o$ are fixed (see Figure 2).

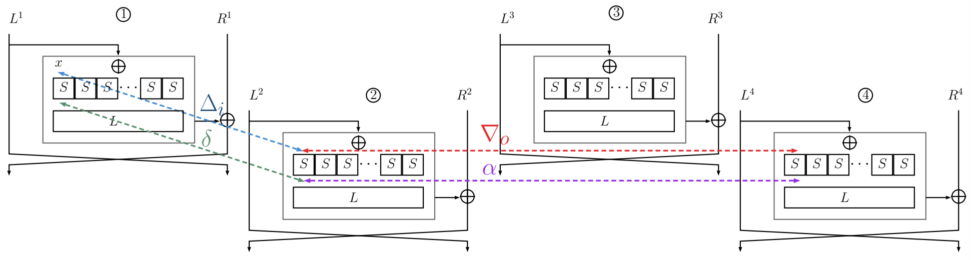


**Figure 2:** View of the parameters of the tables: $\Delta_i$ is the input difference and $\delta$ is the output difference of $S$ when looking at the difference between state ① and ②. $\nabla_o$ is the input difference of the same S-box $S$ when looking at the difference between state ① and ③ and $\alpha$ is its output difference. We focus on the case where the differences are the same on parallel sides.

**From the distinguisher to the attack.** Once an efficient boomerang distinguisher is found, there exist several techniques to extend it to a key recovery attack, as summarized for instance in [DQSW21]. In this article we focus on the technique devised by Zhao *et al.* in [ZDM+20].

The parameters on which the complexities of an attack depend are shown on the right in Figure 3. The key recovery works by adding few rounds before and after the $N_d$ distinguisher rounds $E_d$. We denote $E_b$ the $N_b$ rounds prepended and by $E_f$ the $N_f$ appended rounds. The attacker extends backward with probability one the input difference $\alpha$, obtaining a truncated difference $\alpha'$ with $r_b$ possibly active bits and $n - r_b$ inactive bits. In the same way, $\delta$ is extended forward with probability one over the $N_f$ rounds, giving a truncated difference equal to $\delta'$, with $n - r_f$ inactive bits.

The idea to recover some key material is to make a guess on key bits appearing in $E_b$ and $E_f$ in order to be able to count how many times the distinguishing property happens, the correct key material being amongst the ones with a large number of hits. In the

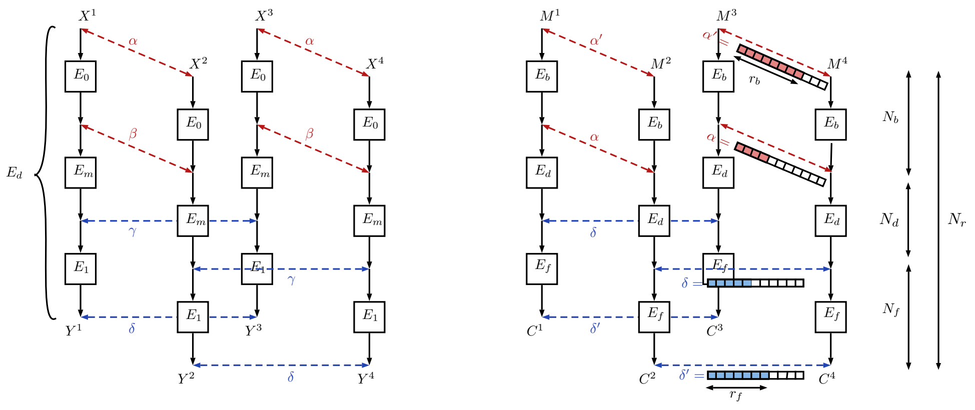


**Figure 3:** Sandwich distinguisher (left) and setting for an attack, including the key recovery (right).

following description, we denote by $m_b$ the number of key bits in $E_b$ and by $m_f$ the number of key bits in $E_f$.

The detail of the attack procedure devised by Zhao *et al.* [ZDM+20] is as follows, where $s$ is the expected number of right pairs:

1. Build $y = \sqrt{s} \cdot 2^{n/2-r_b}/\sqrt{p^2q^2r}$ structures of $2^{r_b}$ plaintexts each, and store them with their associated plaintexts.
2. For each possible value of the $m_b$ key bits:
   (a) Initialize $2^{m_f}$ key counters.
   (b) Partially encrypt each plaintext $M^1$ of each structure using the guessed $m_b$ key bits up to the beginning of $E_d$. Add $\alpha$ to the computed value and decrypt it up to the plaintext, to obtain $M^2$. Construct the set $S$ (of size $y \cdot 2^{r_b}$) given by:
   $$S = \{(M^1, C^1, M^2, C^2) | E_b(M^1) \oplus E_b(M^2) = \alpha\}.$$
   (c) Insert $S$ into a hash table $H$ indexed by the $n - r_f$ bits that are inactive in $\delta'$. Each collision defines a quartet $(C^1, C^2, C^3, C^4)$.
   (d) Use these quartets to determine the correct $m_f$ key bits. The time complexity of this stage is denoted $\epsilon$.

Depending on the parameters, two factors might be dominating the time complexity; either the cost of stage 2.(b) or the cost of the last stage. Their time complexity is respectively $2^{m_b+r_b} \cdot y \cdot \mu = 2^{m_b+n/2} \cdot \sqrt{s} \cdot \frac{1}{\sqrt{p^2q^2r}} \cdot \mu$ and $s \cdot 2^{m_b-n+2r_f}/(p^2q^2r) \cdot \epsilon$ encryptions. Since stage 2.(b) does partial encryptions over $E_b$, $\mu$ can be approximated by $\frac{N_b}{N_b+N_d+N_f}$ while $\epsilon$ corresponds to the cost of gradually decrypting rounds to check the validity of a key guess, so we decide to approximate it by $\frac{1}{s}$.

**Success probability.** We use the formula devised in [Sel08] for differential cryptanalysis (and later used in the context of rectangle attacks) to evaluate the probability of finding the correct key:

$$P_s = \Phi(\frac{\sqrt{sS_N} - \Phi^{-1}(1-2^{-h})}{\sqrt{S_N+1}}),$$

where $S_N$ is the signal-to-noise ratio, so is equal to $p^2q^2r/2^{-n}$ and $h$ is the advantage.

## 2.3 Delaune *et al.*'s Model

In [DDV20], Delaune *et al.* propose a model divided into two steps to search for optimal boomerang distinguishers on SPN ciphers. In Step 1, a MILP model searches for truncated boomerangs where each S-box is represented by 6 binary variables (described below). The solutions of Step 1 are the input of a Step 2 search that tries to instantiate those truncated boomerangs with concrete nibble differences so that the distinguisher has the highest possible probability. In this article, we use the same notation and similar steps.

A boomerang distinguisher uses two differential trails, one is called the upper trail and determines $\alpha$, the input difference of the distinguisher. The other one is called the lower trail and determines $\delta$, the output of the distinguisher (see Figure 3). In the model proposed in [DDV20] the division as a sandwich is not made but the upper and lower trails are searched on all the rounds. In what follows we denote by $\delta_X[r,i]$ the nibble difference at the input of an S-box and by $\delta_{\texttt{SB}}[r,i]$ the corresponding output difference of the S-box.

The boomerang model of [DDV20] uses six variables for each S-box in its Step 1: 3 variables relate to the upper trail (in the encryption direction) whereas the 3 others relate to the lower trail (in the decryption direction). These variables are used to select the proper boomerang transition tables and are defined as:

$\texttt{isActive}_{\texttt{Xup}}[r,i]$ (respectively $\texttt{isActive}_{\texttt{Xlo}}[r,i]$) is a Boolean variable that indicates if the nibble difference $\delta_X[r,i]$ is active in the upper (resp. lower) trail, considering that it represents the S-box input,

$\texttt{free}_{\texttt{Xup}}[r,i]$ (respectively $\texttt{free}_{\texttt{Xlo}}[r,i]$) is a Boolean variable that indicates if the nibble difference $\delta_X[r,i]$ is free of conditions, that is can take any value with a uniform probability in the upper (resp. lower) trail,

$\texttt{free}_{\texttt{SBup}}[r,i]$ is a Boolean variable that indicates if the nibble difference $\delta_{\texttt{SB}}[r,i]$ can take any value with a uniform probability in the upper trail as an output of the S-box. $\texttt{free}_{\texttt{SBlo}}[r,i]$ is a Boolean variable that indicates if the nibble difference $\delta_{\texttt{SB}}[r,i]$ can take any value with a uniform probability in the lower trail. Note that the $\texttt{free}_{\texttt{SBup}}[r,i]$ variable represents the state of the variable after the S-box in the encryption direction, so $\texttt{free}_{\texttt{SBlo}}[r,i]$ can be seen as the input state of the S-box in the decryption direction.

Several constraints describe the relations between these variables, starting with the one modelling the propagation of the free states through the S-boxes: if a variable is free before an S-box, it is also free after the S-box. Since the propagation is done in the opposite direction for the lower trail, the implication is in the other direction for the `lo` variables.

$$\begin{aligned}\texttt{free}_{\texttt{Xup}} &\implies \texttt{free}_{\texttt{SBup}}\\ \texttt{free}_{\texttt{SBlo}} &\implies \texttt{free}_{\texttt{Xlo}}\end{aligned}$$

The second rule ensures that if an S-box output is free then the S-box input must be non-zero. Again the lower trail is reversed since it represents the decryption direction.

$$\begin{aligned}\texttt{free}_{\texttt{SBup}} &\implies \texttt{isActive}_{\texttt{Xup}}\\ \texttt{free}_{\texttt{Xlo}} &\implies \texttt{isActive}_{\texttt{Xlo}}\end{aligned}$$

The third rule ensures that we can compute the probability of the S-box by setting a minimum number of parameters.

$$\begin{aligned}\texttt{free}_{\texttt{SBup}} + \texttt{free}_{\texttt{SBlo}} &\leq 1\\ \texttt{free}_{\texttt{Xup}} + \texttt{free}_{\texttt{Xlo}} &\leq 1\end{aligned}$$

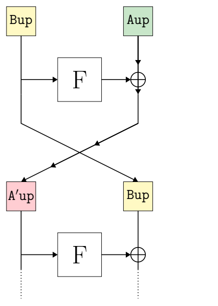
Finally, for any linear operation there is a constraint stating that if any input variable is free then all the output variables on which it depends are also free.

Given this set of constraints, the solver is going to choose the best truncated trail among the ones with a valid propagation of differences, where the quality of a trail is measured by the best probability it might reach. Namely, given the state of each S-box, one can determine which table (DDT, BCT, etc) should be used to compute its probability, and based on this the best probability of the Step 1 solution is obtained by assuming that the best transition in the table is met. Once the best solution for Step 1 is found it is given as input to the Step 2 model, which looks for a concrete instance of the upper and lower trails, again with the objective of reaching the best possible probability.

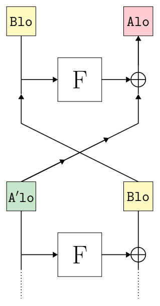
## 3 Automatic Search of Boomerang Distinguishers

### 3.1 Automatic Search of Truncated Boomerang Distinguishers for Feistel Ciphers

This section describes how to build an automated tool that searches for truncated boomerang distinguishers for Feistel ciphers. Our method follows the idea developed by Delaune *et al.* for Skinny in [DDV20] but makes the required adjustments to fit the Feistel structure.



**(a)** Feistel forward round.



**(b)** Feistel backward round.

**Figure 4:** Encryption (4a) and decryption (4b) procedure of a classical Feistel cipher. Note that the $F$ function is never inverted and that the only difference comes from the direction in which the $XOR$ is computed.

**Required changes.** The model presented in [DDV20] for SPN ciphers treats differently the S-boxes of the lower and upper trail to take into account the direction in which they are computed. Given the specific property of Feistel ciphers (that are their own inverse) and as illustrated in Figure 4, our model does not have to make this distinction, so we end up with the same constraints for the S-boxes in the upper and in the lower trail:

$$\begin{aligned} \texttt{free}_{\texttt{Xup}} &\implies \texttt{free}_{\texttt{SBup}} \\ \texttt{free}_{\texttt{Xlo}} &\implies \texttt{free}_{\texttt{SBlo}} \end{aligned} \tag{2}$$

For the same reason we change the second constraint as follows:

$$\begin{aligned} \texttt{free}_{\texttt{SBup}} &\implies \texttt{isActive}_{\texttt{Xup}} \\ \texttt{free}_{\texttt{SBlo}} &\implies \texttt{isActive}_{\texttt{Xlo}} \end{aligned} \tag{3}$$

Knowing which input and output differences are fixed for every S-box allows to select the correct table from Definition 1 to compute the associated boomerang probability. For instance, if the two input differences $\Delta_i$ and $\nabla_o$ are fixed while $\alpha$ and $\delta$ are free parameters, the required table is $FBCT(\Delta_i, \nabla_o)$.

For the model, it corresponds to the case where the input value of `Xup` and the input value of `Xlo` are fixed, so where $\texttt{free}_{\texttt{Xup}}$ and $\texttt{free}_{\texttt{Xlo}}$ are assigned to `false`. $\alpha$ and $\delta$ being free means that $\texttt{free}_{\texttt{SBup}}$ and $\texttt{free}_{\texttt{SBlo}}$ are equal to `true`, so we end up with constraint (4) that indicates when the $FBCT$ table is required to compute a 1-round probability.

$$\texttt{predicate isFBCT}(r,i) = \left( \begin{array}{c} \texttt{isActive}_{\texttt{Xup}}[r, 2\times i] \land \neg\texttt{free}_{\texttt{Xup}}[r, 2\times i] \land \texttt{free}_{\texttt{SBup}}[r,i] \\ \land \\ \texttt{isActive}_{\texttt{Xlo}}[r, 2\times i] \land \neg\texttt{free}_{\texttt{Xlo}}[r, 2\times i] \land \texttt{free}_{\texttt{SBlo}}[r,i] \end{array} \right) \tag{4}$$

Other tables are built in the same way and we obtain constraints (7) to (11) of Model 1 that select the correct table according to which variables are fixed or not.

To ensure that the probability of each S-box can be computed using one of the Feistel boomerang transitions, we add the following constraint:

$$\begin{array}{ll} \texttt{free}_{\texttt{Xup}} + \texttt{free}_{\texttt{SBlo}} & \leq 1 \\ \texttt{free}_{\texttt{Xlo}} + \texttt{free}_{\texttt{SBup}} & \leq 1 \end{array} \tag{5}$$

While S-boxes are treated in the same way in the upper and lower trail, special care has to be taken to correctly propagate knowledge through the $XOR$ operations. In the upper trail (Figure 4a) we have the following equation: $A' = F(B) \oplus A$ while for the lower trail (Figure 4b) we have: $A = F(B) \oplus A'$. This leads to the following distinction in the constraints:

$$\begin{array}{ll} \texttt{free}_{\texttt{A'up}} & = (\texttt{free}_{F(\texttt{Bup})} \lor \texttt{free}_{\texttt{Aup}}) \\ \texttt{free}_{\texttt{Alo}} & = (\texttt{free}_{F(\texttt{Blo})} \lor \texttt{free}_{\texttt{A'lo}}) \end{array} \tag{6}$$

Constraints (2) to (5) are the core mechanisms of the boomerang model for Feistel ciphers. They must be applied on every S-box transition. Constraint (6) must be used on the parts of the state that are $XOR$ed together.

**Resulting model.** The complete model is provided in Model 1. Its first half is dedicated to the selection of the correct boomerang table. The second part starts with constraint (12) which ensures that the trails are active (*i.e.* that there is at least one difference in $\alpha$ and $\delta$). Constraints (13) to (16) define the propagation from one round to the other, while the block of constraints (17) corresponds to the constraints (2), (3) and (5) explained at the beginning of this section and model the S-box transition. The model ends with the objective (18) given here in its naive form and that can be simplified as we now discuss.

## 3.2 Improvements

**Weighted sum simplification.** Given a model looking for differential characteristics, an upper bound of the probability is obtained by multiplying the number of active S-boxes found during Step 1 by the $log_2$ of the maximum probability of the transition of an active S-box, that is $UB = 2^{-\mathrm{p}_{DDT} \times \#SB}$.

Similarly, for a model looking for boomerang distinguishers, the upper bound has to take into account the various tables that are possible (the ones that can be selected in the first half of Model 1) and for each of them their maximum probability, denoted predicate isDDT$(r, i)$ =

$$\left(\begin{array}{c} \neg\mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \bigvee \left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \neg\mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \end{array}\right) \tag{7}$$

predicate isDDT$^2(r, i)$ =

$$\left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \bigvee \left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \tag{8}$$

predicate isFBCT$(r, i)$ =

$$\left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \tag{9}$$

predicate isFBDT$(r, i)$ =

$$\left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \bigvee \left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \tag{10}$$

predicate isFBET$(r, i)$ =

$$\left(\begin{array}{c} \mathtt{isActive}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xup}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBup}}[r, i] \\ \wedge \\ \mathtt{isActive}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{Xlo}}[r, 2\times i] \wedge \neg\mathtt{free}_{\mathtt{SBlo}}[r, i] \end{array}\right) \tag{11}$$

**Model 1:** Model searching for truncated boomerangs on WARP, part 1/2: table selection.

$\mathtt{P}_{DDT}, \mathtt{P}_{DDT^2}, ..., \mathtt{P}_{FBDT}$. The objective (and consequently the bound) thus corresponds to a weighted sum, as shown in (18), Model 1.

Even if the semantic remains the same, reordering this weighted sum may have a huge impact on the resolution time. The first simplification that can be done corresponds to cases where a table has a maximum probability of 1. In such a setting, the table can simply be ignored during Step 1. This happens for the *FBCT* of WARP. The second simplification occurs when different tables have the same maximum probability, in which case they can be grouped by their respective maximum probabilities. For WARP, such an equality happens for the *DDT*, the *FBDT* and the *FBET* which have the same maximum denoted $\mathtt{P}_{isTable}$. Also, the $DDT^2$ can be handled by counting them twice more than the *DDT* in the sum. Thus, the *obj* function can be simplified as follows:

$$\begin{aligned}&\sum_{i=0}^{31}\left(\texttt{isActive}_{\texttt{Xup}}[\textsf{first distinguisher round}, i]\right) \neq 0\\ &\sum_{i=0}^{31}\left(\texttt{isActive}_{\texttt{Xlo}}[\textsf{last distinguisher round}, i]\right) \neq 0\end{aligned} \tag{12}$$

$\forall r \in \textsf{all rounds}^{\star}, \forall i \in [0,\ \texttt{BR}/2[$

$$\begin{aligned}&\texttt{free}_{\texttt{Xup}}[r+1, \pi_{even}[i]] = \texttt{free}_{\texttt{Xup}}[r, 2\times i]\\ &\texttt{free}_{\texttt{Xup}}[r+1, \pi_{odd}[i]] = (\texttt{free}_{\texttt{Xup}}[r, 2\times i+1] \vee \texttt{free}_{\texttt{SBup}}[r, i])\end{aligned} \tag{13}$$

$\forall r \in \textsf{all rounds}, \forall i \in [0,\ \texttt{BR}/2[$

$$\begin{aligned}&\texttt{free}_{\texttt{Xlo}}[r, 2\times i] = \texttt{free}_{\texttt{Xlo}}[r+1, \pi_{even}[i]]\\ &\texttt{free}_{\texttt{Xlo}}[r, 2\times i+1] = (\texttt{free}_{\texttt{Xlo}}[r+1, \pi_{odd}[i]] \vee \texttt{free}_{\texttt{SBlo}}[r, i])\end{aligned} \tag{14}$$

$\forall r \in \textsf{all rounds}, \forall i \in [0,\ \texttt{BR}/2[$

$$\begin{aligned}&\texttt{isActive}_{\texttt{Xup}}[r+1, \pi_{even}[i]] = \texttt{isActive}_{\texttt{Xup}}[r, 2\times i]\\ &\texttt{isActive}_{\texttt{Xup}}[r+1, \pi_{odd}[i]] + \texttt{isActive}_{\texttt{Xup}}[r, 2\times i+1] + \texttt{isActive}_{\texttt{Xup}}[r, 2\times i] \neq 1\end{aligned} \tag{15}$$

$\forall r \in \textsf{all rounds}, \forall i \in [0,\ \texttt{BR}/2[$

$$\begin{aligned}&\texttt{isActive}_{\texttt{Xlo}}[r+1, \pi_{even}[i]] = \texttt{isActive}_{\texttt{Xlo}}[r, 2\times i]\\ &\texttt{isActive}_{\texttt{Xlo}}[r+1, \pi_{odd}[i]] + \texttt{isActive}_{\texttt{Xlo}}[r, 2\times i+1] + \texttt{isActive}_{\texttt{Xlo}}[r, 2\times i] \neq 1\end{aligned} \tag{16}$$

$\forall r \in \textsf{all rounds}, \forall i \in [0,\ \texttt{BR}/2[$

$$\begin{aligned}&\texttt{free}_{\texttt{SBup}}[r, i] \implies \texttt{isActive}_{\texttt{Xup}}[r, 2\times i]\\ &\texttt{free}_{\texttt{SBlo}}[r, i] \implies \texttt{isActive}_{\texttt{Xlo}}[r, 2\times i]\\ &\texttt{free}_{\texttt{Xup}}[r, 2\times i] \implies \texttt{free}_{\texttt{SBup}}[r, i]\\ &\texttt{free}_{\texttt{Xlo}}[r, 2\times i] \implies \texttt{free}_{\texttt{SBlo}}[r, i]\\ &\texttt{free}_{\texttt{Xup}}[r, 2\times i] + \texttt{free}_{\texttt{SBlo}}[r, i] \leq 1\\ &\texttt{free}_{\texttt{Xlo}}[r, 2\times i] + \texttt{free}_{\texttt{SBup}}[r, i] \leq 1\end{aligned} \tag{17}$$

$$obj = \sum_{r\in\substack{\textsf{distinguisher}\\ \textsf{rounds}}} \sum_{i=0}^{15} \begin{pmatrix} \texttt{P}_{DDT} \times \texttt{isDDT}[r,i] & + & \texttt{P}_{DDT^2} \times \texttt{isDDT}^2[r,i] & +\\ \texttt{P}_{FBCT} \times \texttt{isFBCT}[r,i] & + & \texttt{P}_{FBDT} \times \texttt{isFBDT}[r,i] & +\\ \texttt{P}_{FBET} \times \texttt{isFBET}[r,i] & & & \end{pmatrix} \tag{18}$$

$$\texttt{minimize } obj$$

$^{\star}$: all rounds include $E_b$, $E_d$ and $E_f$ rounds of Figure 3 while distinguisher rounds only include $E_d$ rounds.

**Model 1:** Model searching for truncated boomerangs on `WARP`, part 2/2. $\pi_{even}$ and $\pi_{odd}$ correspond to the subparts of the $\pi$ permutation for even or odd inputs only. $BR$ is the number of branches of the cipher, so is equal to 32 in the case of `WARP`.

$$obj = \sum_{r \in \substack{\text{distinguisher}\\\text{rounds}}} \sum_{i=0}^{15} \begin{pmatrix} \mathtt{P}_{isTable} \times (\mathtt{isDDT}[r,i] \vee \mathtt{isFBDT}[r,i] \vee \mathtt{isFBET}[r,i] \vee \mathtt{isDDT}^2[r,i]) & + \\ \mathtt{P}_{isTable} \times \mathtt{isDDT}^2[r,i] & \end{pmatrix} \tag{19}$$

In addition to this, since there is a single maximum probability for all the tables (except for the $FBCT$ removed previously), we can rewrite the weighted sum as:

$$obj = \mathtt{P}_{isTable} \times \sum_{r \in \substack{\text{distinguisher}\\\text{rounds}}} \sum_{i=0}^{15} \Big( (\mathtt{isDDT}[r,i] \vee \mathtt{isFBDT}[r,i] \vee \mathtt{isFBET}[r,i] \vee \mathtt{isDDT}^2[r,i]) + \mathtt{isDDT}^2[r,i] \Big) \tag{20}$$

Finally, once the objective function is simplified we can use the Quine-McCluskey algorithm to create a minimized Boolean predicate $\mathtt{isTable}[r,i] = (\mathtt{isDDT}[r,i] \vee \mathtt{isFBDT}[r,i] \vee \mathtt{isFBET}[r,i] \vee \mathtt{isDDT}^2[r,i])$ and use it in the weighted sum:

$$obj = \mathtt{P}_{isTable} \times \sum_{r \in \substack{\text{distinguisher}\\\text{rounds}}} \sum_{i=0}^{15} ((\mathtt{isTable}[r,i]) + \mathtt{isDDT}^2[r,i]) \tag{21}$$

**Incremental search.** In our model, the objective function non-strictly decreases as the number of rounds $r$ increases, since if we do not add an active S-box we will have the same optimal probability while if we do add one S-box the probability will always be equal (if the maximal possible probability of the table is 1) or lower (if the maximum possible probability is strictly less than 1). We can use this information to lower bound the objective function for $r+1$ rounds when we know the optimal probability for $r$ rounds:

$$\mathtt{P}_{isTable} \times \sum_{r'=N_b}^{N_b+r} \sum_{i=0}^{15} ((\mathtt{isTable}[r',i]) + \mathtt{isDDT}^2[r',i]) \geq obj_r \tag{22}$$

This observation allows to give the model additional information about the minimum bound, which makes it stop the search earlier and therefore save execution time.

**Forcing the pattern of the solution.** The previous model allows to find distinguishers for up to 20 rounds of WARP (in both Step 1 and Step 2), but more rounds are out of reach as the computation time grows exponentially in the number of rounds.

However, we found out that all the optimal solutions returned for $N_d = 15$ to $N_d = 20$ have a specific pattern of the form `1-1-0-1-1`, that is contain a sequence of 5 rounds with 1 active S-box in the first round, 1 active S-box in the second round, 0 in the third one, and so on.

While we cannot formally prove that this pattern is going to appear in the optimal solutions for 21 rounds and more, we believe that there are high chances that it does, so we decided to add a Step 1 constraint that forces the solutions to follow this specific pattern. This assumption seems reasonable as we did not observe a break in the probability chart.

Formally, it gives (note that we do not fix the position where the pattern appears):

$$\bigvee_{r=N_b}^{N_b+N_d-5} \left( \bigwedge_{i=r}^{r+4} (\sum_{j=0}^{15} \mathtt{isActive}_{\mathtt{Xup}}[i, 2\times j] = \mathtt{pattern}_i) \right) \text{ with } \mathtt{pattern} = [1,1,0,1,1] \tag{23}$$

## 3.3 Instantiating the Truncated Boomerang Distinguishers

As mentioned before, we decompose our analysis into two steps. The first one (described above) implements the search of truncated boomerang distinguishers and is written in Picat SAT [ZK16], the SAT compiler in the Picat system. Each S-box of each round is associated to 6 bits: 3 for the upper trail and 3 for the lower trail. They indicate if an S-box is active or not, if the S-box input is free or not and if the S-box output is free or not.

The second step looks for concrete instantiations of the previous truncated solutions. It is written in the open source Java constraint programming library Choco [PFL16]. This step is also inspired from the one of [DDV20].

### 3.3.1 CP Model

The Constraint Programming model of Step 2 takes as input the results of Step 1 to know the general shape of the distinguisher, in particular which nibbles are inactive. To transform a truncated solution into a concrete one we need to assign values to the nibbles. For each pair of nibble abstraction $(\texttt{isActive}_{\texttt{X}}, \texttt{free}_{\texttt{X}})$ we create a variable $\delta_X$ whose domain depends on the Step 1 solution:

$$\delta_X \in \begin{cases} [0,16[ & \texttt{if free}_{\texttt{X}} = \texttt{true} \\ \{0\} & \texttt{else if isActive}_{\texttt{X}} = \texttt{false} \\ [1;16[ & \texttt{else} \end{cases}$$

In the same way, $\delta_{\texttt{SB}}$ variables are created depending on the value of the pair $(\texttt{isActive}_{\texttt{X}}, \texttt{free}_{\texttt{SB}})$. As the `free` variables can take any value from the nibble domain and are not constrained by the model we ignore them in Step 2.

The exact probability of each round is computed by using table constraints, which are tables containing all the possible (or impossible) transitions. For instance, describing that $x \oplus y = z$ for binary variables could be done with the table constraint:

$$(x, y, z) \in Tab_{\oplus} \text{ where } Tab_{\oplus} = \{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}.$$

One table constraint is created for each of the tables appearing in the probability computation (DDT, DDT$^2$, FBCT, FBDT and FBET), and we also make one to handle the XOR over nibbles. In addition to indicating the valid transitions, it also contains a third variable corresponding to the absolute value of the base 2 logarithm of its probability. The truncated solutions outputted by Step 1 completely define which table is used. For example if we have: $(\texttt{isActive}_{\texttt{Xup}}[r, 2\times j] \wedge \neg\texttt{free}_{\texttt{Xup}}[r, 2\times j] \wedge \neg\texttt{free}_{\texttt{SBup}}[r, j] \wedge \neg\texttt{isActive}_{\texttt{Xlo}}[r, 2\times j])$, which corresponds to a DDT transition in the upper trail, we add the constraint: $(\delta_{\texttt{Xup}}[r, 2\times j], \delta_{\texttt{SBup}}[r, j], p[r, j]) \in Tab_{\texttt{DDT}}$. The objective function is then the following sum:

$$obj_2 = \sum_{r \in \substack{\text{distinguisher} \\ \text{rounds}}} \sum_{i=0}^{15} \left(p[r, i]\right) \tag{24}$$

**Combining the two Steps.** Step 1 is composed of two different strategies: $Step1 - Opt$ that searches for the truncated boomerang with the best objective $obj$ and $Step1 - Next$ that enumerates one by one the solutions that reach this minimum $obj$. The best $obj$ value is an upper bound ($UB$) that can not always be reached as Step 1 is an abstraction (some truncated solutions may not have concrete instances). The lower bound ($LB$) is fixed to 0 since it is the lowest possible value. For a given number of rounds, we first run $Step1 - Opt$ to find $UB$, and we next interleave $Step1 - Next$ with Step 2 to obtain a concrete boomerang with the best probability. Once done, we update $LB$ with this new value and repeat the process for all the Step 1 optimal solutions. If a Step 2 is returned it means that the solution has a better probability than the given $LB$, so we update $LB$ and we continue the search. If no Step 1 is found it means that we have already seen all the Step 1 solutions that can match $UB$, so we degrade $UB$ and we continue the search with $Step1 - Opt$. If $LB = UB$ we have found the best solution available. Note that the model generates many solutions in Step 1 and most of the time we stop the search when $LB = UB$ instead of enumerating all possible Step 1 solutions.

### 3.3.2 Clusters

Once the optimal solution has been found for Step 1 and Step 2 (this solution is hereafter denoted $< S_1ref, S_2ref >$), the goal is to obtain a better approximation of the actual probability of the boomerang distinguisher by considering clusters. Indeed, the solution returned by Step 2 has most of its S-box transitions fixed, while the only differences that matter when considering a boomerang distinguisher are the input and output differences ($\alpha$ and $\delta$ in Figure 3).

To get closer to this actual probability, we start by generating multiple Step 1 solutions that have their truncated differences in the first round and in the last round equal to the ones of $S_1ref$. The objective is to take into account many solutions that are all different one from the others. We need to be carefull about what being different means in our context as the situation is a bit more subtil than for a differential attack (for which the only two possible S-box status are "active" and "inactive").

In our model, we have the special case of the `free` S-box inputs that can take any value uniformly. If we focus on one particular S-box, the case of a fixed active input difference can be seen as contained in this one, so we must not count these cases as two independent ones. To be on the safe side, we choose to consider that two Step 1 solutions are different if at least one of their S-boxes is not free and inactive in one while it is not free and active in the other.

We thus search for the Step 2 solutions corresponding to these, with the additional condition that the nibble differences in the first and last rounds are the one of $S_2ref$. We sum over the different values of $obj_2$ in a variable called $OBJ$. To avoid counting solutions with too low probabilities we leave out the solutions of probability lower than $2^{-10} \times p(S_1ref)$ for both Step 1 and Step 2. Still, the large number of solutions forces us to set a limit on the number of solutions enumerated in Step 2 for a given Step 1 solution: we set this limit to $2^{20}$.

To check the validity of our approach, we wanted to compare the result of the simple model and of the model with the clusters with what can be experimentally observed. To do so, we decided to pick a Feistel cipher with a smaller block size than `WARP` with the hope that the clustering effect would be easier and faster to observe. We selected the 64-bit block cipher TWINE [SMMK13] reduced to 12 rounds. We first computed its experimental probability by fixing the differences $\alpha$ and $\delta$ and we counted how many times the boomerang comes back. For the considered example, we obtained a probability close to $2^{-25.8}$ when testing $2^{29}$ plaintexts with $2^4$ keys with a Rust experiment. The corresponding optimal Step 1 solution has a probability of $2^{-26}$ and is instantiated in Step 2 with a trail of probability $2^{-26}$. When aggregating solutions with the same input and output differences we obtain an approximation of the distinguisher probability of $2^{-25.15}$ which is close to the experimental result, albeit slightly exceeding it.

## 3.4 23-round Distinguisher on `WARP`

**Implementation details.** The Step 1 is written in MiniZinc and runs on Picat [ZK16] which uses the Lingeling solver [Bie11] under the hood, the Step 2 is written in Choco [PFL16] version 4.10.6 which is a dedicated framework for Constraint Programming running

**Table 4:** Best distinguishers found after 2 days when summing up boomerang characteristics in the same cluster for the best Step 1 solutions.

| Rounds | Cluster | $obj_{step_2}$ | gain |
|---|---|---|---|
| 18 | $2^{-54}$ | $2^{-58}$ | $2^4$ |
| 19 | $2^{-66}$ | $2^{-70}$ | $2^4$ |
| 20 | $2^{-76}$ | $2^{-84}$ | $2^8$ |
| 21 | $2^{-96}$ | $2^{-104}$ | $2^8$ |
| 22 | $2^{-108}$ | $2^{-120}$ | $2^{12}$ |
| 23 | $2^{-124}$ | $2^{-140}$ | $2^{16}$ |

on the Java Virtual Machine. We choose the Picat solver to solve the Step 1 as it is a SAT solver, so is especially suited to problems on Boolean formulae. Previous works like [LDLS21] have shown that Picat has good performances on multiple Step 1 models. Since the Step 2 contains a lot of table constraints, it appears that CP solvers are more adapted. The experiments are run on a virtual machine `Ubuntu 18.04.5 LTS x86_64` with an `Intel Xeon Gold 5118` processor and `32 Gio` of RAM. The requirements are : `Java 10.0.12 OpenJDK`, `Gradle 6.8`, `MiniZinc 2.5.5`, `Picat 3.1.2` and `Choco 4.10.6`. Each instance is run on a single thread.

We found 20 instances (from 3 rounds up to 22 rounds) with a probability better than $2^{-128}$. They all took less than 48 hours to solve.

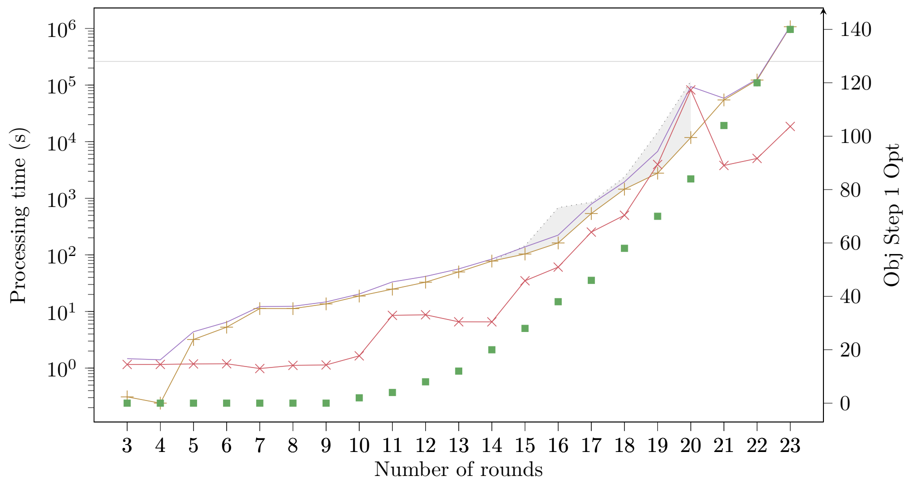


**Figure 5:** Execution time of Step 1 Opt enforcing the `1-1-0-1-1` pattern (in seconds) (—+—), execution time of Step 1 Enum + Step 2 Opt (—×—), Total time (——). Best probability found with Step 1 Opt (■). Time of Step 1 Opt without the `1-1-0-1-1` pattern (⋯⋯). The black line corresponds to the probability $2^{-128}$.

Without taking into account the clusters, the longest distinguisher that can be obtained is a 22-round boomerang of probability $2^{-120}$. By summing up several boomerang trails inside one 23-round solution we are able to build a distinguisher of 23 rounds with probability $2^{-124}$. By exploiting the position of the key addition, it can easily be extended to a 25-round distinguisher, thanks to the easy trick that we now present[1].

The distinguisher is depicted in Appendix A. Its 32 nibbles of input and output

[1]Note that a similar trick can be used to extend the 21-round impossible differential distinguisher proposed by the designers of `WARP` to a 23-round distinguisher.

differences are given by:

$$\begin{aligned}\alpha &= \texttt{57 00 00 07} \quad \texttt{00 00 57 57} \quad \texttt{07 57 00 07} \quad \texttt{00 00 57 00}\\ \delta &= \texttt{70 05 00 70} \quad \texttt{05 00 70 70} \quad \texttt{00 00 70 70} \quad \texttt{00 00 00 05}\end{aligned}$$

(note that for simplicity we kept the last round permutation in the figures and here). To exploit this distinguisher, an attacker would ask for the encryption of a large number of pairs $M^1, M^2$ verifying $M^1 \oplus M^2 = \alpha$, and build two new ciphertexts by computing $C^3 = E(M^1) \oplus \delta$ and $C^4 = E(M^2) \oplus \delta$. She would then ask for the corresponding plaintexts and check if $E^{-1}(C^3) \oplus E^{-1}(C^4) = \alpha$.

Since the round keys are added after the application of the S-boxes, an attacker can compute the difference entering the second round of the upper trail, and similarly the difference at the input of the S-boxes of the penultimate round of the lower trail. This easily leads to an extension of two rounds of any boomerang distinguisher. The attacker starts by picking a random message $M^1 = M^1[0], M^1[1], \cdots, M^1[31]$, and computes $M^2$ according to the difference she wants to observe one round later. For instance, it would give the begining of $M^2$ to be

$$M^2 = M^1[0], M^1[1], M^1[2] \oplus \texttt{0x7}, M^1[3] \oplus \texttt{0x5} \oplus S(M^1[2]) \oplus S(M^1[2] \oplus \texttt{0x7}), \cdots$$

A similar idea gives $C^3$ in function of $C^1$ and $C^4$ in function of $C^2$, and the boomerang returns if $M^3$ and $M^4$ verify

$$M^4 = M^3[0], M^3[1], M^3[2] \oplus \texttt{0x7}, M^3[3] \oplus \texttt{0x5} \oplus S(M^3[2]) \oplus S(M^3[2] \oplus \texttt{0x7}), \cdots.$$

## 4 Automatic Search of Rectangle Attacks

As already discussed in [ZDJ19] in the case of Deoxys-BC, the best rectangle distinguishers do not always lead to the best attacks, and choosing a sub-optimal (in terms of probability) distinguisher might allow to cover more rounds in the key recovery phase, and then to attack a bigger version of the cipher.

Following this idea, Lingyue Qin *et al.* proposed an automatic model that directly searches for an attack [QDW+21] by taking into account the dominating factors of the key-recovery step. The model simply minimizes the time complexity of the attack instead of maximizing the probability of the distinguisher, while making sure that the data complexity does not exceed the full codebook. The number of rounds on which is run the model is gradually increased until the returned time complexity exceeds the cost of an exhaustive search of the secret key. This technique turned effective as it leads to improved attacks on the SPN ciphers Skinny and ForkSkinny [QDW+21].

In this section we study how to apply a similar idea to find good attack parameters for `WARP` and show that when considering the attack technique introduced by Zhao *et al.* in [ZDM+20] there are at least two possible improvements in comparison to a variant of `WARP` with the key addition positioned before the S-box. The first one is the reduction of the value of $m_b$ (the number of key bits that have to be guessed in the upper rounds), and the second one is the potential growth of the number of filtering bits, that is of $n - r_f$.

These two improvements are crucial since the two predominating factors of the time complexity of the attack of [ZDM+20] are $2^{m_b+n/2} \cdot \sqrt{s} \cdot \frac{1}{\sqrt{p^2q^2r}} \cdot \frac{N_b}{N_b+N_d+N_f}$ and $2^{m_b-n+2r_f}/(p^2q^2r)$.

## 4.1 Taking Advantage of the Structure of WARP

**Reduction of $m_b$.** To understand the first point, we look at a simple example that considers $N_b = 3$ rounds of key recovery prepended to the distinguisher, where $\alpha$, the top difference of the boomerang distinguisher, has only two active nibbles, in position 1 and 4 (see the bottom of Figure 6).

To determine the value of $r_b$ (the number of active bits in the plaintext structures), an attacker starts by propagating backwards the difference $\alpha$ to know which nibbles might be active and which are inactive for sure. This process is rather straightforward, and in our example it returns $r_b = 32$ active nibbles (denoted in green in Figure 6).

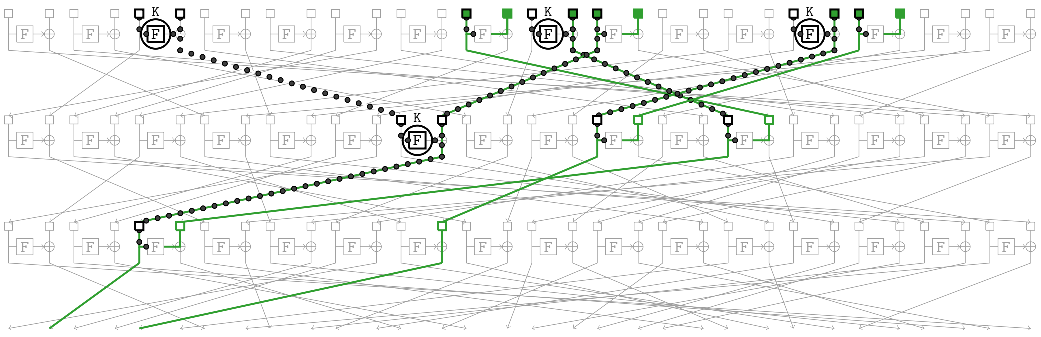


**Figure 6:** Determining the required key bits to apply [ZDM+20] over 3 rounds.

The next step is the determination of the key bits that are required to compute $M^2$ from $M^1$. In the description given in [ZDM+20], the attacker starts from $M^1$, computes partially the state at the input of the boomerang distinguisher so that she can add $\alpha$ to it, and decrypts the result to get $M^2$. Put differently, it can be seen as guessing the necessary key bits to uniquely determine the difference to add to $M^1$ to get $M^2$ knowing that after $N_b$ rounds the internal states differ of $\alpha$, which can also be seen as being able to uniquely propagate $\alpha$ backwards.

Consequently, the attacker starts from $\alpha$, and takes note of the information needed: in round 2, the exact difference at the output of the second S-box has to be uniquely determined, so the input *value* of this S-box is needed. This is denoted by the dashed lines in Figure 6. Knowing this value in round 2 implies that the two inputs of the xor number 6 of round 1 are known, which in particular forces to make a guess on the key nibble added after the S-box. The rest of the backward propagation is processed similarly, and we obtain that 4 nibbles of key are required in total. In the case where round keys are added before the S-boxes, this computation would have return a total of 10 nibbles of key (we consider here that the round keys are independent).

**Improved filtering process.** For some specific shapes of output difference $\delta'$, the attacker is able to increase the number of bits on which she looks for collision in step number 2(c) of the attack. An example of this is given in Figure 7 with $N_f = 2$: by counting the number of active nibbles at the output, we obtain $r_f = 21 \times 4 = 84$, which means that the hash table is used to look for collisions over $2 \times (128 - 84) = 88$ bits.

In our example, the $F$ functions number 4 to 9, 12 and 13 in the last round all have a similar difference pattern where the input of the S-box ($X[r, 2i]$) is active while the right part ($X[r, 2i + 1]$) is not.

The inactivity of $X[r, 2i + 1]$ can be translated into the equality

$$F(C^1[2i]) \oplus C^1[2i + 1] \oplus F(C^3[2i]) \oplus C^3[2i + 1] = 0.$$

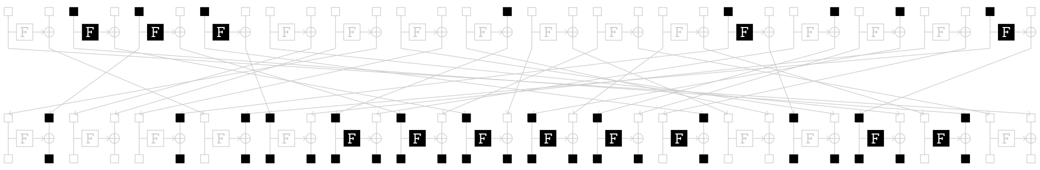


**Figure 7:** Example of difference propagation over $N_f = 2$ rounds.

Given that $F$ is the application of an S-box followed by a sub-key addition it can be simplified into: $S(C^1[2i]) \oplus C^1[2i+1] = S(C^3[2i]) \oplus C^3[2i+1]$ which does not depend on a secret value. Thus, the idea is to simply add as index the value of $S(C[2i]) \oplus C[2i+1]$ when building the $H$ table of step 2.(c), in addition to the value of the bits where the difference is expected to be 0 in the ciphertexts. In our example, it means that we are colliding on 32 additional bits, and thus that the filter for quartets is of size $2 \times (128 - 52) = 152$ bits.

## 4.2 Model for Searching a Rectangle Attack

In this subsection, we briefly go over the main characteristics of the model searching for the rectangle attack. The detailed model is given in Model 2 in Appendix C. It takes as input the number of rounds covered by the distinguisher and by the prepended and appended rounds of key recovery (respectively $N_d, N_b$ and $N_f$) and returns the complexities of the best attack that can be found.

The intermediate values that are needed to evaluate the time and data complexity of the attack are $m_b, r_f$, and the probability of the distinguisher (previously denoted $p^2q^2r$). This latter is determined with the same constraints as in the model searching for the best distinguisher, and we simply add constraints to model the additional $N_b$ and $N_f$ rounds.

- $r_b$ and $r_f$ are computed by propagating with probability one the difference at the input and at the output of the boomerang distinguisher, see constraints (30) and (34). To take into account the above described trick on the filtering process, we define $r_f$ as the number of active nibbles *entering* the last round.
- The data complexity is computed so that $s$ right quartets are found, see constraint (27). To make sure that it does not exceed was is available, one may add a constraint stating that $\frac{\sqrt{s} \cdot 2^{n/2}}{\sqrt{p^2q^2r}} < 2^n$.
- As it is impossible to compute the cluster at this stage, we introduce a variable $cluster_{gain}$ which is set by the attacker to represents the expected gain obtained with clusters. Its value is precisely computed afterwards, once a solution to the model is obtained.
- The time complexity is computed as the maximum between the two most expensive stages detailed in Section 2.2, see constraint (27). Again, one may add a constraint saying that the resulting time complexity has to be smaller than the cost of an exhautive search of the key.
- Constraint (29) makes the link between the `known` variables and the $\texttt{guess}_{\texttt{key}}$ variables.
- We take into account the simple key schedule of `WARP` to precisely compute the value of $m_b$. We start by determining the states that have to be known in value (denoted *known* in the model, constraints (31), (32) and (33)) and then link them to the keys and to $m_b$, taking into account the key schedule (constraints (28) and (29)).

All the values (except $t$ which is related to the distinguisher probability and $cluster_{gain}$ which is first only approximated) can be computed during Step 1. As a result all the constraints are computed in Step 1 and only the constraints that imply $2t$ or $t$ are modeled in Step 2.

## 4.3 Results: A 26-round Attack on WARP

We apply the previous model to search for rectangle attacks on WARP with various values for the parameters $N_b, N_d$ and $N_f$. As the execution time rapidly increases with the number of rounds, we added another constraint (proposed in [DDV20]) which consists in using the bounds obtained for differential distinguishers. The idea is that the upper trail (resp. lower trail) cannot be better than a differential trail, i.e. the number of active S-boxes in the upper trail (resp. lower trail) cannot be lower than the minimal number of active S-boxes of a differential trail ($optimal_{diff}$). To implement this idea for WARP we use the lower bound of the number of active S-boxes computed in [BBI+20].

The best attack we found covers 26 rounds of WARP based on a $N_d = 22$ rounds distinguisher, $N_b = 1$ round added before and $N_f = 3$ rounds added after. The model took 6 days to solve this instance, and returned the following values: $m_b = 0$ , $r_b = 72$, $r_f = 60$ and a distinguisher probability of $2^{-128}$. This search was made by assuming that $s$ is equal to 4 and that the value of the cluster gains would be the ones given in Table 4, so in the case of a 22-round distinguisher equal to a factor of $2^{12}$.

We next run the cluster search on the 22-round distinguisher. We obtained a probability approximation of $2^{-111.2}$ (so with a cluster gain a bit larger than what was expected), resulting in the associated attack having a data complexity of $2^{120.6}$ messages, and a time complexity of little less than $2^{116}$ encryptions when following the key recovery method introduced by Zhao *et al.* [ZDM+20].

The success probability of the attack is equal to 97,67 % (using the formula given in [Sel08]) and this is the best attack reported so far on WARP.

## 4.4 On the Impact of the Key Addition Position

We now briefly discuss a variant of WARP with a round key addition made before the S-box application, and consider an attack using the same 22-round distinguisher, and the same number of rounds $N_b$ and $N_f$ added before and after the distinguisher. The different round structure changes the value of $m_b$ and $r_f$, as the improvements discussed in Section 4.1 cannot be applied anymore. $m_b$ increases (from 0) to 32, while $r_f$ is now equal to 88 instead of 60. The data complexity of an attack with such parameters would still be $2^{120.6}$, but the dominating factor of the time complexity becomes $2^{191.2}$.

This example shows the importance of the key addition position and of the techniques discussed in Section 4.1 that save a factor of $2^{75.3}$ in the time complexity.

## 5 Conclusion

In this article, we propose the adaptation of two recent techniques to the case of Feistel ciphers to find boomerang distinguishers and rectangle attacks. Our analysis reveals a 23-round distinguisher and a 26-round attack of WARP, beating by 2 rounds the recent results of [TB21]. Our code is public and can be used as a basis to attack other Feistel ciphers, and we actually demonstrate its versatility by providing results for TWINE and LBlock-s (see Appendix D).

Secondarily, while studying WARP we show how to take advantage of the key addition position to reduce the complexity of the attack. In our specific case, this design decision allows to reduce by a factor of $2^{75}$ the time complexity of the attack in comparison to a variant of WARP that would have the key addition positioned before the S-box (and thus would have the complementation property).

## Acknowledgments

This work has been partly funded by the French Agence Nationale de la Recherche through the Decrypt project under Contract ANR-18-CE39-0007. Some of the experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). We also used LIMOS' servers.

## References

[BBI+20] Subhadeep Banik, Zhenzhen Bao, Takanori Isobe, Hiroyasu Kubo, Fukang Liu, Kazuhiko Minematsu, Kosei Sakamoto, Nao Shibata, and Maki Shigeri. WARP : Revisiting GFN for lightweight 128-bit block cipher. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 535–564. Springer, 2020.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the Serpent. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 340–357. Springer, Heidelberg, May 2001.

[BHL+20] Hamid Boukerrou, Paul Huynh, Virginie Lallemand, Bimal Mandal, and Marine Minier. On the Feistel counterpart of the boomerang connectivity table (long paper). *IACR Trans. Symm. Cryptol.*, 2020(1):331–362, 2020.

[Bie11] Armin Biere. Lingeling and Friends at the SAT Competition 2011. 2011. Publisher: Institut for Formal Models and Verification, Johannes Kepler University.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, Heidelberg, December 2009.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer, Heidelberg, August 1991.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, Heidelberg, April / May 2018.

[DDV20] Stéphanie Delaune, Patrick Derbez, and Mathieu Vavrille. Catching the fastest boomerangs application to SKINNY. *IACR Trans. Symm. Cryptol.*, 2020(4):104–129, 2020.

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 393–410. Springer, Heidelberg, August 2010.

[DQSW21] Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. Key guessing strategies for linear key-schedule algorithms in rectangle attacks. Cryptology ePrint Archive, Report 2021/856, 2021. https://eprint.iacr.org/2021/856.

[HBS21] Hosein Hadipour, Nasour Bagheri, and Ling Song. Improved rectangle attacks on SKINNY and CRAFT. *IACR Trans. Symm. Cryptol.*, 2021(2):140–198, 2021.

[KKS01] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 75–93. Springer, Heidelberg, April 2001.

[KY20] Manoj Kumar and Tarun Yadav. MILP based differential attack on round reduced WARP. Cryptology ePrint Archive, Report 2020/1598, 2020. https://eprint.iacr.org/2020/1598.

[LDLS21] Luc Libralesso, François Delobel, Pascal Lafourcade, and Christine Solnon. Automatic generation of declarative models for differential cryptanalysis. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Mur11] Sean Murphy. The return of the cryptographic boomerang. *IEEE Trans. Inf. Theory*, 57(4):2517–2521, 2011.

[PFL16] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[QDW+21] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. Automated search oriented to key recovery on ciphers with linear key schedule. *IACR Trans. Symm. Cryptol.*, 2021(2):249–291, 2021.

[Sel08] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, January 2008.

[SMMK13] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 339–354. Springer, Heidelberg, August 2013.

[SN14] Hadi Soleimany and Kaisa Nyberg. Zero-correlation linear cryptanalysis of reduced-round lblock. *Des. Codes Cryptogr.*, 73(2):683–698, 2014.

[TB21] Je Sen Teh and Alex Biryukov. Differential cryptanalysis of WARP. Cryptology ePrint Archive, Report 2021/1641, 2021. https://eprint.iacr.org/2021/1641.

[Wag99] David Wagner. The boomerang attack. In Lars R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 156–170. Springer, Heidelberg, March 1999.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. In Javier Lopez and Gene Tsudik, editors, *ACNS 11*, volume 6715 of *LNCS*, pages 327–344. Springer, Heidelberg, June 2011.

[ZDJ19] Boxin Zhao, Xiaoyang Dong, and Keting Jia. New related-tweakey boomerang and rectangle attacks on deoxys-bc including BDT effect. *IACR Trans. Symm. Cryptol.*, 2019(3):121–151, 2019.

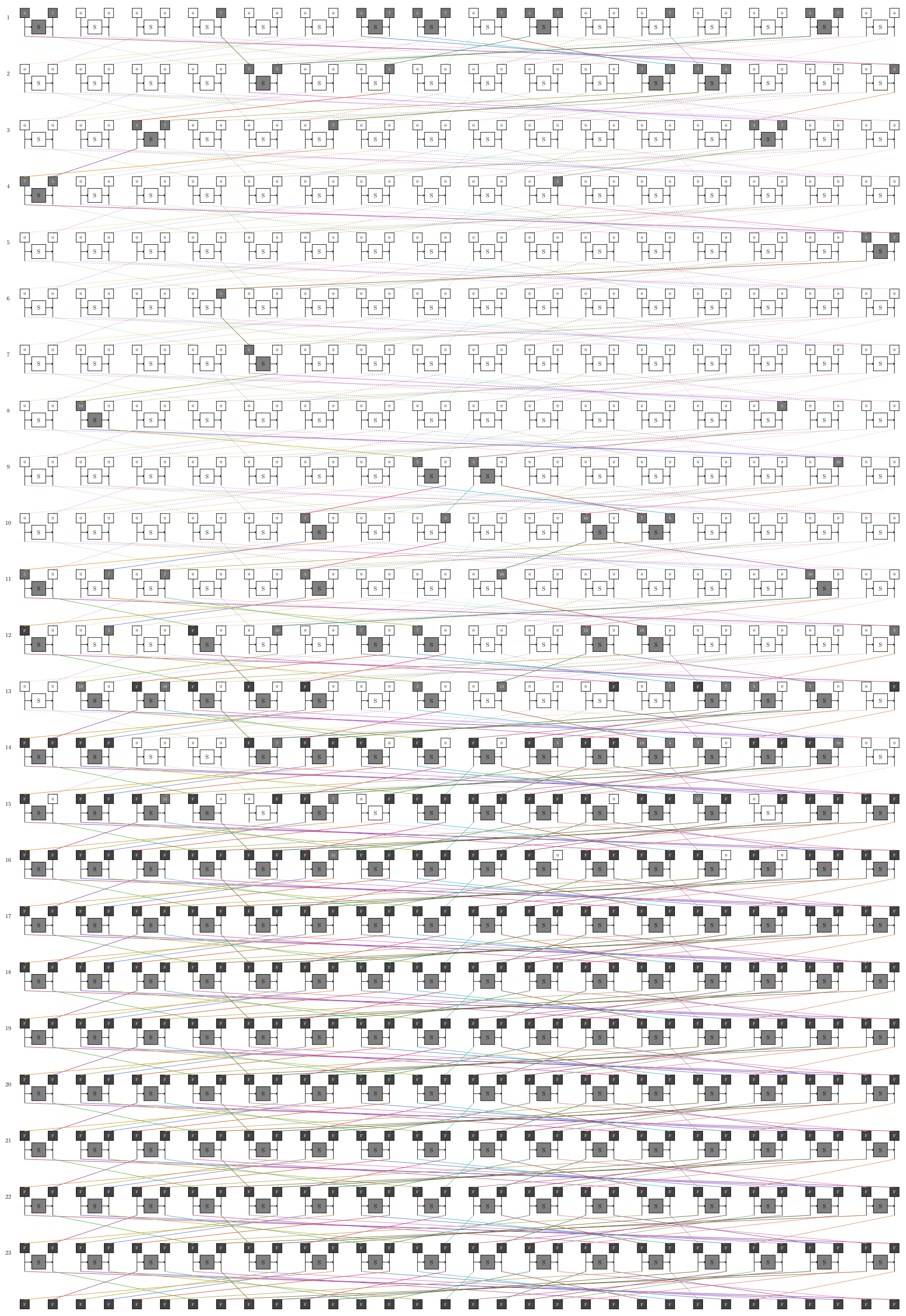
[ZDM$^+$20] Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. Generalized related-key rectangle attacks on block ciphers with linear key schedule: applications to SKINNY and GIFT. *Des. Codes Cryptogr.*, 88(6):1103–1126, 2020.

[ZK16] Neng-Fa Zhou and Håkan Kjellerstrand. The picat-sat compiler. In *Practical Aspects of Declarative Languages - PADL 2016*, volume 9585 of *LNCS*, pages 48–62. Springer, 2016.

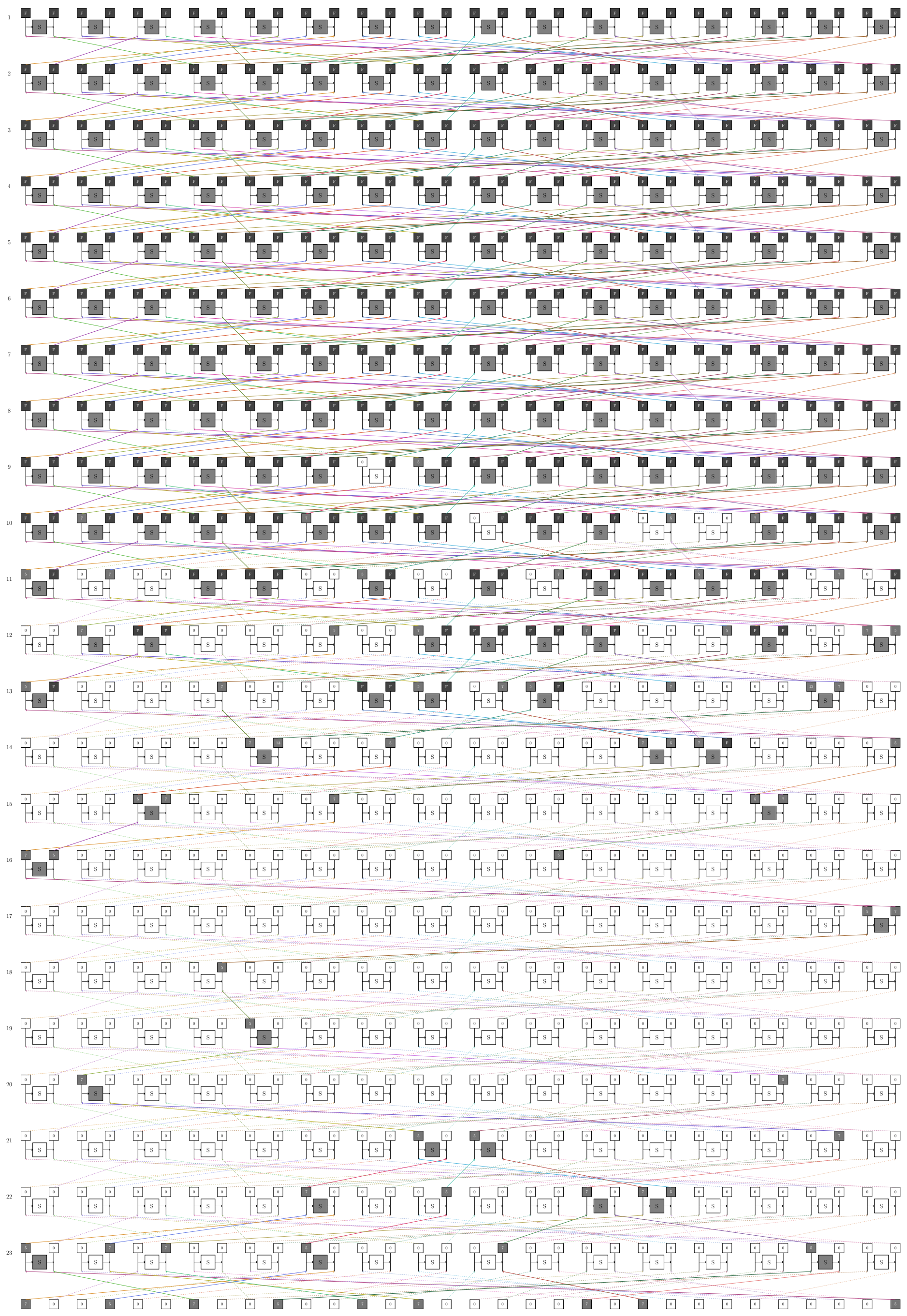
[ZWW$^+$14] Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. Lac: A lightweight authenticated encryption cipher. *Submitted to the CAESAR competition*, 2014.

## A 23-round Boomerang Distinguisher on WARP

### A.1 Upper Trail

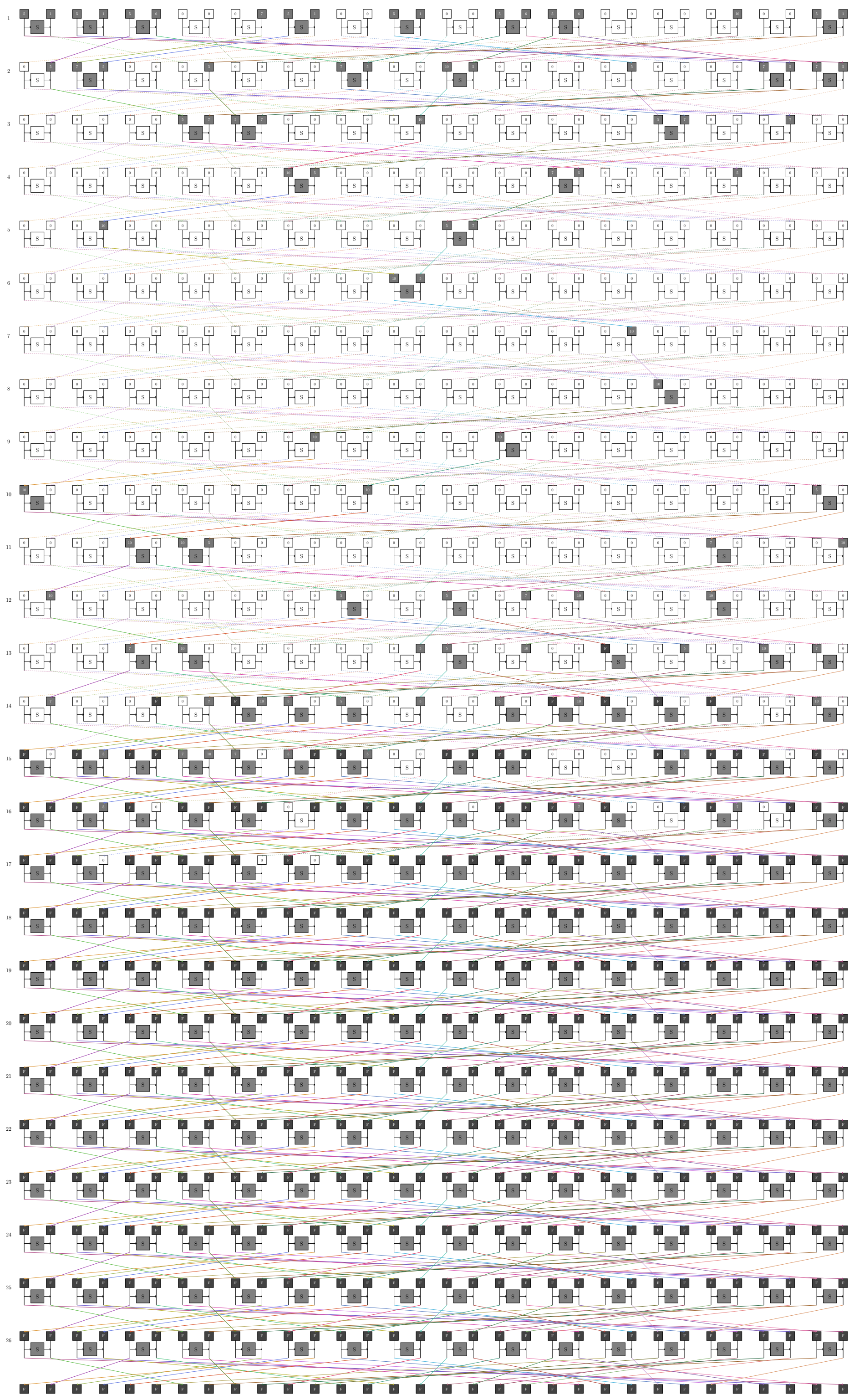


## A.2 Lower Trail

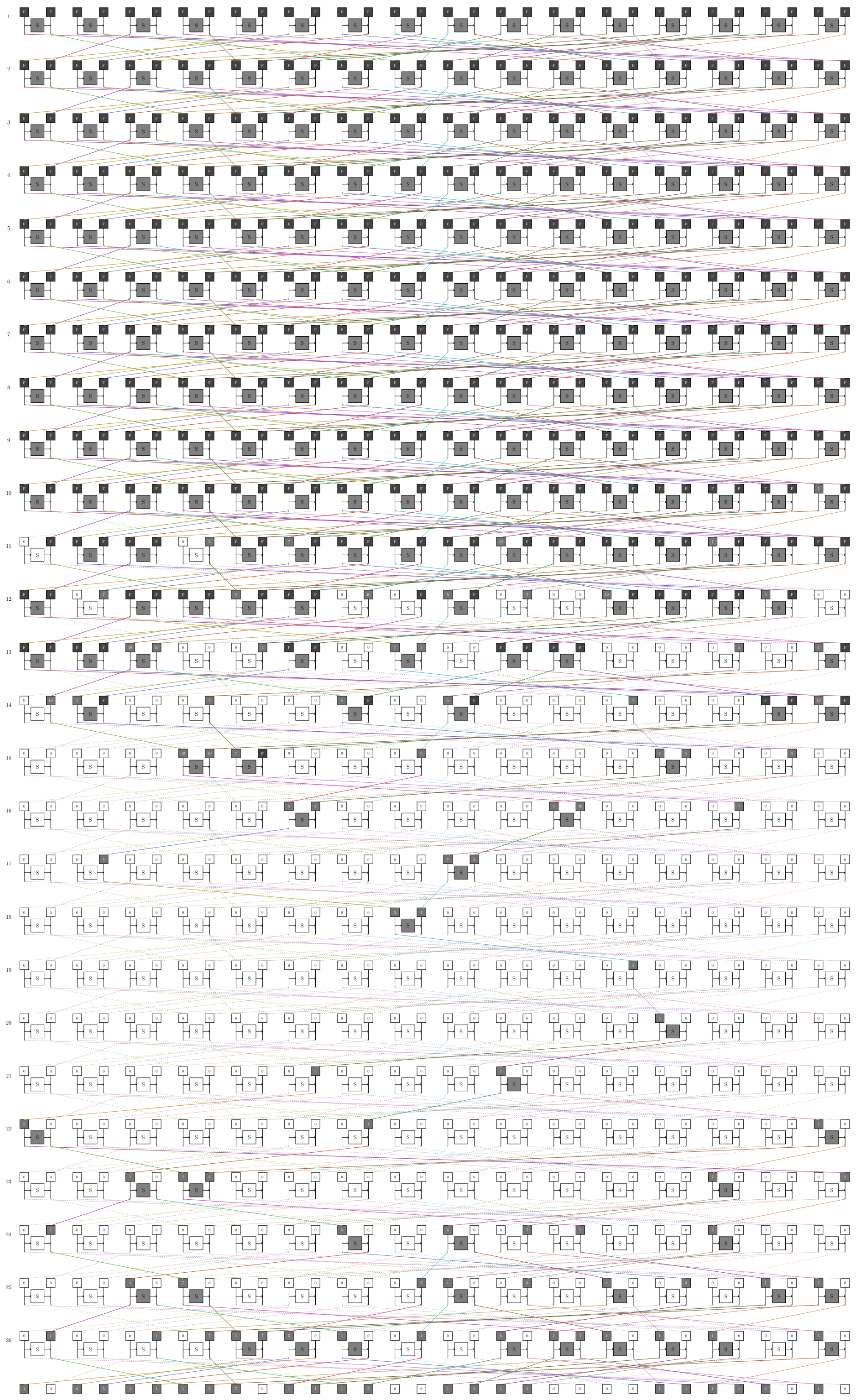


## B 26-round Boomerang Attack on WARP

### B.1 Upper Part



## B.2 Lower Part



## C Model Searching for Rectangle Attacks with the Technique of [ZDM+20]

New variables

$$\begin{aligned} 2t &= obj{-}cluster_{gain} \quad \triangleright \text{ we pose that: } p^2q^2r = 2^{-2t} \\ t &= \lceil 2t/2 \rceil \\ rb &= \sum_{i=0}^{31} \texttt{isActive}_{\texttt{Xup}}[0,i] \times 4 \\ rf &= \sum_{i=0}^{32} \texttt{isActive}_{\texttt{Xlo}}[N_r-1,i] \times 4 \end{aligned} \tag{25}$$

$$data = \sigma + 64 + t \tag{26}$$

$$\begin{aligned} t_1 &= \sigma + 64 + t + mb - 4 \quad \triangleright \text{ we pose that: } s = 2^{2\sigma} \\ t_2 &= 2\sigma + mb - 128 + 2rf + 2t \\ time &= \max(t_1, t_2) \end{aligned} \tag{27}$$

$$mb = \sum_{alt \in \{0,1\}} \sum_{i=0}^{15} \texttt{guess}_{\texttt{key}}[alt, i] \times 4 \tag{28}$$

New constraints

$\forall i \in [0,\ \texttt{BR}/2[$,

$$\begin{aligned} \texttt{guess}_{\texttt{key}}[0,i] &= \#\{\texttt{known}[r, 2\times i+1] \mid r \in [0, N_b[\ \wedge\ r \mod 2 = 0\} \geq 1 \wedge \\ \texttt{guess}_{\texttt{key}}[1,i] &= \#\{\texttt{known}[r, 2\times i+1] \mid r \in [0, N_b[\ \wedge\ r \mod 2 = 1\} \geq 1 \end{aligned} \tag{29}$$

$\forall r \in [0, N_b[, \quad \forall i \in [0,\ \texttt{BR}/2[$,

$$(\texttt{isActive}_{\texttt{Xup}}[r+1, \pi_{odd}[i]] \vee \texttt{isActive}_{\texttt{Xup}}[r, 2\times i]) \implies \texttt{isActive}_{\texttt{Xup}}[r, 2\times i+1] \tag{30}$$

$\forall r \in [0, N_b-1[, \quad \forall i \in [0,\ \texttt{BR}/2[$,

$$\texttt{known}[r+1, \pi_{odd}[i]] \implies (\texttt{known}[r, 2\times i] \wedge \texttt{known}[r, 2\times i+1]) \tag{31}$$

$\forall r \in [0, N_b[, \quad \forall i \in [0,\ \texttt{BR}/2[$,

$$\texttt{isActive}_{\texttt{Xup}}[r+1, \pi_{even}[i]] \implies \texttt{known}[r, 2\times i] \tag{32}$$

$\forall i \in [0,\ \texttt{BR}/2[$

$$\texttt{known}[N_b-1, 2\times i+1] = \texttt{false} \tag{33}$$

$\forall r \in [N_b+N_d, Nr[, \quad \forall i \in [0,\ \texttt{BR}/2[$,

$$\texttt{isActive}_{\texttt{Xlo}}[r, 2\times i+1] \vee \texttt{isActive}_{\texttt{Xlo}}[r, 2\times i] \implies \texttt{isActive}_{\texttt{Xlo}}[r+1, \pi_{odd}[i]] \tag{34}$$

**Model 2:** Attack extension for the attack technique of Zhao *et al.* [ZDM+20].

## D Application of our Technique to TWINE and LBlock-s

To illustrate the flexibility of our tool, this section reports the results obtained when applying it to two well-known Feistel ciphers, TWINE and LBlock-s. TWINE [SMMK13] is a 64-bit block cipher with a Type-II GFN structure and LBlock-s is used in the authenticated encryption LAC [ZWW+14] submitted to the CAESAR competition. LBlock-s is a simplified version of the original cipher LBlock [WZ11] which uses only one S-box instead of the 8 original ones and admits 16 rounds or 32 rounds according to where it is used in LAC. It is also a 64-bit cipher and it could also be represented as a Type-II GFN as shown in [SN14]. This is that representation that we used for our models. Then, for those two ciphers, we apply our method for computing the boomerang clusters and the results are summed up in Table 5.

**Table 5:** Summary of the results for computing the best boomerang clusters for TWINE and LBlock-s.

| Cipher | Distinguishers | Rounds | Probability | Ref. |
|---|---|---|---|---|
| TWINE | Boomerang distinguisher | 15 | $2^{-58.92}$ | [TB21] |
| TWINE | Boomerang distinguisher | 16 | $2^{-61.62}$ | [TB21] |
| TWINE | Boomerang Distinguisher + Clustering | 15 | $2^{-47.7}$ | This paper |
| TWINE | Boomerang Distinguisher + Clustering | 16 | $2^{-59.8}$ | This paper |
| LBlock-s | Boomerang distinguisher | 15 | $2^{-58.64}$ | [TB21] |
| LBlock-s | Boomerang Distinguisher + Clustering | 16 | $2^{-56.14}$ | [BHL+20] |
| LBlock-s | Boomerang Distinguisher + Clustering | 16 | $2^{-54.8}$ | This paper |

In [BHL+20], the authors used a C code to experimentally compute the probability of the 8 middle rounds of the boomerang distinguisher, while a single trail was used for the top and the bottom parts of the sandwich. The slightly improved value obtained with our new method shows that the 8-round boomerang switch does not capture everything, and that other trails contribute to the boomerang.