Impossible Differentials Automation: Model Generation and New Techniques

Emanuele Bellini[1], Alessandro De Piccoli[2], David Gerault[1], Paul Huynh[1], Simone Pelizzola[2], and Andrea Visconti[2]

[1] Technology Innovation Institute, Abu Dhabi, UAE. name.lastname@tii.ae
[2] Università degli Studi di Milano, Milan, Italy, name.lastname@unimi.it

Abstract. In this paper, we propose new techniques for impossible differential cryptanalysis. The first one is a *hybrid* model for finding distinguishers on block ciphers that have both bit-oriented and word-oriented components; we apply this model to LBlock, and build an improbable differential for 18 rounds, improving over the previous 17-round results. Our second model builds impossible differential attacks for ARX ciphers automatically, including, for the first time, hash table based optimizations into the complexity evaluation of the attack. We apply this model to the HIGHT block cipher, and improve complexity of the state-of-the-art 27-round attack. Finally, we include these techniques in the cryptanalysis tool CLAASP, building the needed decryption functions automatically from the block cipher's graph representation; this inversion technique is of independent interest to other similar libraries, such as TAGADA.

Keywords: Impossible differential; LBlock; HIGHT; CLAASP; Automated cryptanalysis

1 Introduction

Block ciphers are a cornerstone of many secure applications, and serve as building blocks for various symmetric primitives; as such, their analysis is one of the most fundamental research areas in symmetric cryptography. A key step in the design of a new block cipher is evaluating its resistance to known cryptanalysis techniques, such as differential attacks, introduced in the late 1980s by Biham and Shamir [7], which evaluates the probability of a *differential* $\delta \to \gamma$ through the cipher, that is, the probability that a pair of plaintexts with XOR difference δ is encrypted to a pair of ciphertexts with XOR difference γ.

In this paper, we focus on impossible differential cryptanalysis [6] [21], the study of *impossible differentials*, which no input satisfies. Such distinguishers allow invalid key guesses to be discarded, reducing the space for exhaustive search. This remains the best known attack on the cipher Camellia [9].

The search for differentials, possible or not, is a challenging problem, due to the inherent combinatorial explosion resulting from the probabilistic aspect of differential transitions through the building blocks of a cipher. In recent years,

the community has put more focus towards generic tools, where the search process is automated through a dedicated solver, significantly simplifying the process. Libraries like TAGADA and CLAASP are now commonly used to analyze ciphers, given their representation in a specific language or format. For example, CLAASP was used to automatically evaluate the NSA's ARADI cipher shortly after its release, despite the lack of an official security analysis [3].

In this paper, we propose two new impossible differential models, as well as their automation in the CLAASP framework[3]. Our contributions are as follows.

- **Automation -** We extend CLAASP with a technique to automatically build the decryption function of a block cipher represented as graph; this technique can be adapted for libraries that use a similar representation, such as TAGADA. Using this automated inversion, we include the state of the art impossible differential search techniques from [16] to CLAASP, along with our new models in CLAASP.
- **Hybrid model -** We propose *hybrid* model that combines the cell-wise properties of [29] and the bit-wise granularity of [10] and [16]. Our hybrid model supports key-dependent probabilistic transitions and detects more incompatibilities than standard truncated models, by leveraging the impossible differential clustering effect.
- **Improved LBlock Cryptanalysis -** Using this model, we show that the 17-round improbable differential on LBlock presented in [10] is invalid and we exhibit the first 18-round *improbable* differential, valid for about $2^{-0.83}$ of the key space.
- **Hash Table Modeling -** We extend the automatic technique of [16] to integrate the use of hash-tables into the overall attack cost.
- **Improved HIGHT Cryptanalysis -** Using this model, we build a 27-round key recovery attack with time complexity $2^{118.9}$ and data complexity 2^{55}, improving the state-of-the-art complexity from $2^{120.58}$ time and $2^{59.3}$ data. Notably, our technique fully automates the highly sophisticated use of multiple hash tables to improve the attack complexity.

2 Automatic Impossible Differential Search Tools

Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ be a block cipher. An *impossible differential* for E is a pair of differences δ, γ, such that $E_K(X) \oplus E_K(X \oplus \delta) = \gamma$ never holds.

In differential cryptanalysis, the differential propagation rules associated with each operation of the cipher are used iteratively to build a sequence of transitions forming a *differential trail* from δ to γ. Impossible differentials are identified by proving that no such trail exists.

Since the introduction of Mixed Integer Linear Program (MILP) models for differential cryptanalysis [24], declarative techniques from the AI and operational research communities have become the tool of choice for similar problems. In these frameworks, the differential propagation rules are encoded into a model,

[3] https://github.com/Crypto-TII/claasp/blob/main/claasp/cipher_modules/models

defined by *variables* and *constraints*: Boolean CNF formulas for SAT, linear inequalities for MILP, and generic constraints for CP. The model is then given to a specialized solver.

Two independent works by Sasaki and Todo [28] and Cui et al. [13] propose to find impossible differentials fixing the input and output difference of an MILP differential model, and checking the existence of a trail; a negative answer indicates an impossible differential. This method has the advantage of making no assumption on the type of contradiction, but is inherently limited by the size of the subspace of input and output differences that can be enumerated.

Alternatively, the contradiction can be modeled explicitly by encoding deterministic propagation rules in the forward and backwards direction and applying a miss-in-the-middle approach, requiring a contradiction between the reached intermediate states. This technique shifts the problem from proving unsatisfiability to proving satisfiability, removing the need for explicit enumeration. In the following section, we briefly review miss-in-the-middle approaches, focusing on the operations that are used in the ciphers we study.

Sun et al. [29]. In [29], Sun et al. propose a CP-based model for the deterministic propagation of truncated differences in cell-oriented ciphers. Their tool enumerates forward truncated trails $\Delta_{in_0} \rightarrow \Delta_{out_0}$ and backward truncated trails $\Delta_{out_1} \leftarrow \Delta_{in_1}$. Incompatibility between Δ_{out_0} and Δ_{out_1} is then verified outside the CP model.

Variables. Consider $\Delta X = (\Delta X_0, \Delta X_1, \dots \Delta X_{m-1})$, the difference of the internal state X of size $m \cdot s$ bits, with $X_i \in \mathbb{F}_2^s$. For each cell difference ΔX_1, two variables are introduced: $\delta X_i \in \{0, 1, 2, 3\}$ represents the differential pattern of ΔX_i and $\zeta X_i \in \{-2, -1, 0, \dots, 2^{s-1}\}$ represents the actual value of ΔX_i:

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_i = 0 \ (Z) \\ 1 & \text{if } \Delta X_i \text{ is nonzero and fixed } (N) \\ 2 & \text{if } \Delta X_i \text{ is nonzero } (N^*) \\ 3 & \text{if } \Delta X_i \text{ is unknown } (U) \end{cases} \qquad \zeta X_i \in \begin{cases} \{0\} & \text{if } \delta X_i = 0 \\ \{1, \dots, 2^{s-1}\} & \text{if } \delta X_i = 1 \\ \{-1\} & \text{if } \delta X_i = 2 \\ \{-2\} & \text{if } \delta X_i = 3 \end{cases}$$

Constraints.

- For the **XOR** operation $Y = X_0 \oplus X_1$, the differential patterns satisfy:

  if $\delta X_0 + \delta X_1 > 2$ then $\delta Y = 3$ and $\zeta Y = -2$
  elseif $\delta X_0 + \delta X_1 = 1$ then $\delta Y = 1$ and $\zeta Y = \zeta X_0 + \zeta X_1$
  elseif $\delta X_0 = \delta X_1 = 0$ then $\delta Y = 0$ and $\zeta Y = 0$
  elseif $\zeta X_0 + \zeta X_1 < 0$ then $\delta Y = 2$ and $\zeta Y = -1$
  elseif $\zeta X_0 = \zeta X_1$ then $\delta Y = 0$ and $\zeta Y = 0$
  else $\delta Y = 1$ and $\zeta Y = \zeta X_0 \oplus \zeta X_1$ endif

- For the bijective **S-box** application $Y = S(X)$, the patterns satisfy:

  $$\delta Y \neq 1 \text{ and } \delta X + \delta Y \in \{0, 3, 4, 6\} \text{ and } \delta Y \geq \delta X \text{ and } \delta Y - \delta X \leq 1$$

Cao et al. [10]. Cao et al. extended Sun et al.'s technique to bit-oriented ciphers. Their MILP model relies on *undisturbed differential bits*, a concept previously defined by Teczan[30]. For an input difference Δ_{in}, an output bit Δ_{out_i} is said to be *undisturbed* if its value is deterministically fixed by Δ_{in}; such bits play a valuable role in identifying bit-level incompatibilities.

Variables. Consider $\Delta X = (\Delta X_0, \Delta X_1, \dots \Delta X_{n-1})$, the difference of the internal state X of size n bits. Each bit of the state is associated with a variable $\delta X_i \in \{0, 1, 2\}$ that represents the value of ΔX_i:

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_i = 0 \\ 1 & \text{if } \Delta X_i = 1 \\ 2 & \text{if } \Delta X_i \text{ is unknown} \end{cases}$$

Constraints.

- For the **XOR** operation $Y = X_0 \oplus X_1$, the differential patterns satisfy:

$$\begin{aligned} &\text{if } \delta X_0 = 2 \text{ or } \delta X_1 = 2 \text{ then } \delta Y = 2 \\ &\text{else } \delta Y = \delta X_0 \oplus \delta X_1 \text{ endif} \end{aligned}$$

- For the **Modular Addition** operation $Z = X \boxplus Y$, where $X = (x_{n-1}, \dots, x_0)$, $Y = (y_{n-1}, \dots, y_0)$ and $Z = (z_{n-1}, \dots, z_0)$, the differential patterns satisfy:

$$\begin{aligned} &\delta z_0 = \delta x_0 \oplus \delta y_0,\ c_0 = f_1(\delta x_0, \delta y_0) \\ &\delta z_1 = \delta x_1 \oplus \delta y_1 \oplus c_0,\ c_1 = f_2(\delta x_1, \delta y_1, c_0) \\ &\vdots \\ &\delta z_{n-2} = \delta x_{n-2} \oplus \delta y_{n-2} \oplus c_{n-3},\ c_{n-2} = f_2(\delta x_{n-2}, \delta y_{n-2}, c_{n-3}) \\ &\delta z_{n-1} = \delta x_{n-1} \oplus \delta y_{n-1} \oplus c_{n-2} \end{aligned}$$

$$\text{where } f_1(x, y) = \begin{cases} 0 & \text{if } x + y = 0 \\ 2 & \text{otherwise} \end{cases}, \quad f_2(x, y, c) = \begin{cases} 0 & \text{if } x + y + c = 0 \\ 2 & \text{otherwise.} \end{cases}$$

- For the **S-box** application $Y = S(X)$, where $X = (x_{n-1}, \dots, x_0)$, and $Y = (y_{m-1}, \dots, y_0)$, the DDT restricted to undisturbed bits must be encoded. For each input difference ΔX, let us consider the set $\mathcal{P}_{\Delta X}$ of undisturbed bits positions of S under ΔX. For p in $\mathcal{P}_{\Delta X}$, we denote by b_p the undisturbed bit value at position p of the output, that is, $\Delta y_p = b_p$. If $\mathcal{P}_{\Delta X} \neq \emptyset$ then $\delta X = (\delta X_{n-1}, \dots, \delta X_0)$ propagates to the output difference pattern $\delta Y = (\delta Y_{m-1}, \dots, \delta Y_0)$, where

$$\delta Y_i = \begin{cases} b_i & \text{if } i \in \mathcal{P}_{\Delta X} \\ 2 & \text{otherwise.} \end{cases}$$

  In all the other cases, the output is all unknown: $(\delta Y_{m-1}, \dots, \delta Y_0) = (2)_{i=0}^{m-1}$.

Limitations. In this model, if a nonzero input does not produce any undisturbed bits, an unknown value is assigned to the output. For bijective operations, this results in the loss of crucial information, as the output is guaranteed to be nonzero. Furthermore, the method requires fixing the input patterns, which represents a regression compared to the approach of [29]. Finally, some results in the paper appear to be incorrect, indicating potential issues in the model or its implementation; these are discussed in subsection 5.2.

Hadipour et al. [17]. `Zero` is a tool developed by Hadipour et al. for identifying impossible differential, zero-correlation and integral attacks on block ciphers. Like the approach in [29], it formulates the differential propagations as a constraint optimization problem. However, in this case, the forward and backward trail searches are integrated into a single unified model. The encoding is the one employed in [29] and thus we refer to section 2 for the detailed definitions of variables and constraints. A key strength of this approach is that the tool additionally gives the time complexity of the corresponding key recovery, allowing to find full impossible differential attacks; however, it is limited to word-oriented constructions and the number of rounds forward and backward must be specified in advance.

Hadipour et al. [16]. Earlier this year, Hadipour et al. released `Zeroplus`, an improvement of `Zero` [17]. This updated version extends the tool to weakly aligned primitives by including bit-wise propagations for the branching, XOR and S-box operations, with the inclusion of undisturbed bits, following the approach of [10]. While `Zeroplus` still does not apply to ARX, it addresses the second limitation of the previous version of the tool, by automating the identification of the middle round, removing the need for manual specification. The case of ARX/AndRX constructions has partly been adressed in a recent work by Hadipour et al. [11].

Going Further. All the previously described approaches were implemented manually for specific ciphers. This process is often tedious and error-prone, and work towards more automation is important within the community. We propose a method to automatically generate the decryption function of a cipher within an automated tool, as a stepping stone towards the fully automatic generation of impossible differential models.

3 Automating Cipher Inversion

As noted in the previous section, automation can help limit the risk of human oversight. With this in mind, we aimed to push this principle further by eliminating the need to manually construct the corresponding impossible differential model for each cryptographic primitive. This approach aligns with the philosophy behind automated cryptanalysis tools such as CLAASP [4], which served as

a starting point for our efforts. Indeed, CLAASP already supports the generation of SAT, SMT, CP, and MILP models for various attack scenarios. In particular, the tool can automate the search of differential and linear distinguishers. The first challenge was the systematic inversion of a cipher from its representation in CLAASP to model the backward propagations. We also notice that inversion is a fundamental step in many key recovery techniques. In CLAASP, a primitive is described as a list of connected components, forming a directed acyclic graph. Each component is a dictionary containing an identifier, a type, and a description specifying the operation. It also includes the identifiers of its input components, their bit positions, and the input and output sizes.

3.1 Method description

To ensure generality, the inversion process operates sequentially on individual components, starting from the output and working backward through the cipher to the input, rather than processing groups of components.

First, each component type must have defined handling rules. For components representing a *bijective operation*, the inversion simply involves swapping the input and output and applying the inverse permutation. The condition for this component to be *inverted* successfully is that all output bits must be available by the time it is reached. *Non-bijective operations*, like bit shifts, require more careful consideration. If such a component is encountered during the inversion process, it needs to be *evaluated*; in other words, its input bits must be available for the inversion to succeed. For example, in a 2-branch Feistel structure where the Feistel function is a shift operation, inversion is possible because the input to the Feistel function is also available via the round output.

The XOR operation is another case to examine. Given $a \oplus b = c$, the operation can be evaluated if a and b are known. Alternatively, it can also be inverted as either $a = b \oplus c$ or $b = a \oplus c$, depending on which input is available first.

Given a target cipher C, the inversion process begins by establishing a list L of available components. Specifically, at any stage of the process, a component is considered available if it can either be evaluated or inverted, given the cipher output and the components previously made available up to that point. Initially, L only contains the output of the primitive to be inverted as it becomes an input to the inverse: $L = [\texttt{cipher\_output}]$. All the other components remain in a separate list T, representing components yet to be inverted. As the components are processed, they are moved from T to L. The inversion is complete once T is empty. The procedure is detailed in algorithm 1. The `can_be_evaluated()` and `can_be_inverted()` functions are routines that verify whether enough input and/or output bits are available to process the component c, by establishing its connection to the elements of L. The functions `evaluate_component()` and `invert_component()` create a new component c' from c, based on the rule established for the operation it represents–as discussed above–and its connection to the elements of L. We illustrate this method on a toy example in section A.

Algorithm 1: Cipher inversion algorithm

Input : Target cipher C as a list of connected components
Output: Inverse cipher C_{inv}
$L = [\texttt{cipher\_output}]$
$T = [c \mid c \in C]$,
while $|T| > 0$ **do**
  for c *in* T **do**
    if *can_be_evaluated*(c, L) **then**
      $c' =$ `evaluate_component`(c, L)
      $L = L + [c']$
      remove c from T
    else if *can_be_inverted*(c, L) **then**
      $c' =$`invert_component`(c, L)
      $L = L + [c']$
      remove c from T
Build C_{inv} from the components list L

3.2 Limitations of this approach

This inversion method relies exclusively on local information and processes each component independently, rather than considering larger groups of operations. Consequently, certain components are challenging to invert. For instance, in the linear layer of Ascon [14], each row x is updated through a series of shifts and XORs. Specifically, x is shifted by two values, r_0 and r_1, and these shifted versions are then combined with the original x through an XOR: $y = x \oplus (x \ggg r_0) \oplus (x \ggg r_1)$. From this expression, the original value x can be recovered from y by solving the equation. However, consider an implementation where this transformation is broken into simpler components, as follows:

$$u = x \ggg r_0;\ v = x \ggg r_1;\ y = x \oplus u \oplus v.$$

In such a case, the component producing y is simply viewed as a 3-input XOR operation with no discernible connection to the earlier shifts. This lack of global context prevents the inversion process from reconstructing x. On the other hand, the entire linear layer of Ascon can be expressed as a binary matrix multiplication, which is inherently easy to invert. As such, the effectiveness of this inversion method heavily depends on how the cipher is described in terms of components.

4 Hybrid model for the distinguisher search

The bit-wise model combined with undisturbed bits properties has been shown to be effective against weakly aligned ciphers like Present and Ascon [16]. In fact, even in the case of word-oriented cipher, it may provide extra information that can lead to the detection of more impossible trails such as the undisturbed bits of an S-box or the result an XOR operation between one fully known input and one

partially known input: given $a = ???1$ and $b = 0010$, the bit-wise representation is $a \oplus b = ???1$ while the cell-wise model abstracts this operation as $N \oplus N^* = U$.

However, as already mentioned in section 2, this model presents one limitation: it is unable to keep track of groups of bits that are undetermined yet nonzero. This typically happens when a bijective operation maps a nonzero input difference to a truncated output difference with no undisturbed bit. Let us consider the first S-box of LBlock as an example:

$$S_0 = (14, 9, 15, 0, 13, 4, 10, 11, 1, 2, 8, 3, 7, 6, 12, 5).$$

There are 6 differential transitions with undisturbed bits: $(0000 \xrightarrow{S_0} 0000), (0001 \xrightarrow{S_0} ???1), (0010 \xrightarrow{S_0} ???1), (0011 \xrightarrow{S_0} ??10), (1000 \xrightarrow{S_0} ??1?), (1011 \xrightarrow{S_0} ??0?)$.

In the bit-based model, all other cases are mapped to a fully undetermined output $\delta_{out} = ????$, losing crucial information like the guarantee that a nonzero input implies a nonzero output. These observations motivated us to investigate a *hybrid model*, which integrates bit-based and cell-based representations into a unified framework. This approach eliminates the need to choose between the two models while retaining their respective advantages.

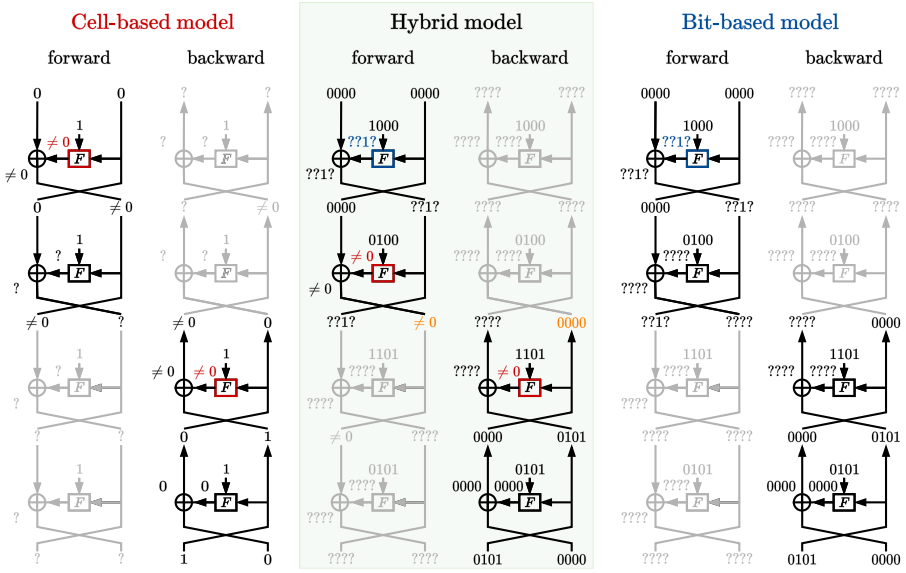
4.1 A toy example

Consider a simple Feistel cipher acting on an 8-bit state $(x_l||x_r)$, where the Feistel function F is defined by a round key addition followed by an S-box: $F(x_l, k_i) = S_0(x_l \oplus k_i)$, with S_0 being the first S-box used in LBlock, previously defined. For simplicity, the key schedule is linear, producing the 4-bit round key differences 1000, 0100, 1101, and 0101 for rounds 0 to 3. Fixing the input and output differences to (0000, 0000) and (0101, 0000), respectively, Figure 1 shows that the regular cell- and bit-based models detect no incompatibility, while the hybrid model identifies the related-key differential as impossible. Indeed, one extra information that the hybrid model gains over the other models is that the right branch at the end of round 1 is undetermined but nonzero. More precisely, these models differ in their treatment of $S(S(0000 \oplus 1000) \oplus 0110)$:

- for the *cell-based* model, $S(0000 \oplus 1000)$ produces an undetermined nonzero value α, making $S(S(0000 \oplus 1000) \oplus 0110) = S(\alpha \oplus 0110)$ unknown[4].
- for the *bit-based* model, $S(0000 \oplus 1000) = ??1?$ using the undisturbed bits of S, and thus $S(S(0000 \oplus 1000) \oplus 0101) = S(??1? \oplus 0101) = S(??1?)$ remains unknown since 1??? has no undisturbed bits;
- for the *hybrid-based* model, similar to the bit-based model, $S(S(0000 \oplus 1000) \oplus 0110) = S(??1? \oplus 0110) = S(??1?)$ but the result is evaluated as a bijective S-box on a nonzero input, so it is undetermined but nonzero.

[4] The XOR of an undetermined nonzero input with a fixed input is unknown.

Fig. 1: Comparison of the cell-based, bit-based, and hybrid models.



4.2 Model description

The hybrid model is an extension of the bit-wise model described in section 2. The core idea is to capture both notions of undisturbed bits and of active groups of bits in S-box outputs and, more generally, in any nonlinear operation that is bijective. As such, it can be seen as an extension of the bit-wise model with undisturbed bits, to which cell-wise properties are added.

Data representation. We recall that in bit-based approaches, for each round r, each bit difference is associated to an integer variable $\delta X_{i,r} \in \{0, 1, 2\}$, where 2 stands for the unknown bit. In the hybrid model, this domain is extended by introducing a unique integer $\texttt{id}_{s,r}$ for each n-bit S-box s of round r that serves as an identifier. The value should not intersect with the base domain of the bit-wise representation, meaning that for all s and r, $\texttt{id}_{s,r} > 2$.

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_{i,r} = 0 \\ 1 & \text{if } \Delta X_{i,r} = 1 \\ 2 & \text{if } \Delta X_{i,r} \text{ is unknown} \\ \texttt{id}_{s,r'} & \text{if } \Delta X_{i,r} \text{ is produced by S-box } s \text{ of round } r', \\ & \text{evaluated on a nonzero input difference} \end{cases}$$

Whenever a nonzero input difference has no undisturbed bits, the n bits of output are all set to $\texttt{id}_{i,r}$ to indicate that even though the output is unknown, at

least one of the bits is nonzero. Denoting n_{sb} the number of S-box per round and n_r the number of rounds, the new domain of each variable $\delta X_{r,i}$ thus becomes $\mathcal{D}(\delta X_{ri}) = \{0,1,2\} \bigcup_{s\in\mathcal{N}_{sb},k\in\mathcal{N}_r} \{\mathtt{id}_{s,k}\}$, where $\mathcal{N}_{sb} = \{0,\dots,n_{sb}-1\}$ and $\mathcal{N}_r = \{0,\dots,n_r-1\}$. The modeling of the S-box and XOR operations needs to be changed to reflect this domain extension.

Extended modeling of a bijective S-box. For the S-box application $Y = S(X)$, where S is the s-th S-box at round r, $X = (x_{n-1},\dots,x_0)$, and $Y = (y_{n-1},\dots,y_0)$, the encoding still uses information from the DDT restricted to undisturbed bits. For this, we introduce new variables $\beta X_i \in \{0,1,2\}$ for each bit difference ΔX_i. These variables satisfy the bit-wise constraints of the S-box. Consider the sets $\mathcal{P}_{\Delta X}$ and $\{b_p \mid p \in \mathcal{P}\}$ as defined in section 2. If $\mathcal{P}_{\Delta X}$ is not empty then $\beta X = (\beta X_{n-1},\dots,\beta X_0)$ propagates to the output difference $\beta Y = (\beta Y_{n-1},\dots,\beta Y_0)$, where

$$\beta Y_i = \begin{cases} b_i & \text{if } i \in \mathcal{P}_{\Delta X} \\ 2 & \text{otherwise.} \end{cases}$$

In all the other cases, $(\beta Y_{n-1},\dots,\beta Y_0) = (2)_{i=0}^{n-1}$. In turn, extended output patterns $\delta Y = (\delta Y_{n-1},\dots,\delta Y_0)$ satisfy

$$\delta Y = \begin{cases} \beta Y \text{ or } \mathtt{id}_{s,r} & \text{if } \exists i \in \{0,\dots,n-1\},\ \delta X_i = 1 \\ \mathtt{id}_{s,r} & \text{if } \forall i \in \{0,\dots,n-1\},\ \delta X_i = \mathtt{id}_{s',r'} \\ 2 & \text{otherwise.} \end{cases}$$

Since the output bits can either be set to the truncated output with undisturbed bits or to the S-box identifier when the input has undisturbed bits and one of the output bits equals 1, this model can find multiple trails given fixed input/output differences, but one is enough.

Extended modeling of the XOR. For the XOR operation $Y = X_0 \oplus X_1$, the propagation of the differential pattern follows:

$$\begin{aligned} &\text{if } \delta X_0 < 2 \wedge \delta X_1 < 2 \text{ then } \delta Y = \delta X_0 + \delta X_1 \\ &\text{elseif } \delta X_0 > 2 \wedge \delta X_1 = 0 \text{ then } \delta Y = \delta X_0 \\ &\text{elseif } \delta X_0 = 0 \wedge \delta X_1 > 2 \text{ then } \delta Y = \delta X_1 \\ &\text{else } \delta Y = 2 \end{aligned}$$

Objective function. With this extended approach, two types of incompatibilities can be detected. Denoting $\delta X u_{r,i}$ (respectively $\delta X l_{r,i}$) the difference in bit i of the internal state at round r in the forward (respectively backward) direction:

1. a **bit-based contradiction** occurs if there exist a round r and a bit position i for which $\delta X u_{r,i} + \delta X l_{r,i} = 1$;
2. a **word-based contradiction** occurs if there exist a round r, n bit positions $i_0,\dots i_{n-1}$ and an S-box index s that verify:

$$\forall j \in \{i_0,\dots,i_{n-1}\}, (\delta X u_{r,i_j} = \mathtt{id}_{s,r} \wedge \delta X l_{r,i_j} = 0).$$

5 Application to LBlock

LBlock is a lightweight block cipher proposed by Wu and Zhang at ACNS 2011 [32], described in section B. Due to its 4-bit cell-aligned round function, and rotation of 29 bits to the left in the key schedule, the cipher seemed like a good target for our hybrid model. Since our approach builds upon the bit-wise model, it achieves the same results for weakly aligned ciphers as those reported by Hadipour et al. [16].

5.1 Previous ID cryptanalysis results on LBlock

The authors of LBlock demonstrated a 20-round single-key impossible differential attack using a 14-round distinguisher found with Kim et al.'s $\mathcal{U}$-method [20]. Karakoç et al. [19] extended this to 21 rounds using the same characteristic, while a 22-round attack was devised using a different 14-round impossible differential. Boura et al. broke 23 rounds with a similar characteristic [8]. Minier and Naya-Plasencia [23] performed the first related-key analysis, covering 22 rounds with a 15-round impossible differential path, exploiting the low diffusion of the key schedule. Wen et al. [31] improved this with a dedicated algorithm, reaching 16 rounds by splitting all key values into partitions of differential trails and checking their impossibility for the entire key space. Cui et al. [13] identified several 16-round differentials, the highest number of rounds with impossible differentials for the full key space, using MILP modeling with constraints on key, input, and output differences. In 2022, Cao et al. [10] introduced an automatic tool incorporating Teczan's undisturbed differential bits [30], discovering a 17-round impossible differential in the weak related-key model. A summary of LBlock's main results is presented in Table 1.

Table 1: Previous results of impossible differential distinguishers for LBlock

Setting	Rounds	Key Space Covered	Search Method	Ref.
Single-Key	14	Full	$\mathcal{U}$-method	[32]
Related-Key	15	Full	Dedicated	[23]
Related-Key	16	Full	Dedicated	[31]
Related-Key	16	Full	MILP (Infeasibility-based)	[13]
Related-Key	17	2^{-2}	MILP (Deterministic Truncated + Undisturbed Bits)	[10]

5.2 Disproving the previous result on LBlock

In the work by Cao et al. [10], the authors present a 17-round related-key *improbable* differential trail for LBlock. Rather than modeling the key schedule with deterministic propagation patterns, they employ probabilistic differential

propagations, shown in Table 2. This approach leads to distinguishers that are longer but are not valid across the entire key space. However, we found out that the described key schedule trail follows invalid transitions.

Table 2: Subkeys used in the 17-round improbable trail by [10]. $\alpha \in \{0,1,2,3\}, \beta \in \{0,4,8,12\}, \gamma \in \{0,1\}, \delta \in \{0,2,4,6,8,10,12,14\}$.

Round	**Subkey**	**Round**	**Subkey**	**Round**	**Subkey**	**Round**	**Subkey**
0	00000000	5	00000000	10	00000000	15	00?00000
1	00030000	6	00000006	11	00000000	16	00000000
2	00000000	7	?0000000	12	000$\gamma\delta$000		
3	00000000	8	00000000	13	00000000		
4	01800000	9	00000$\alpha\beta$0	14	00000000		

More precisely, producing a subkey difference equal to `0x01800000` at round 5 is not possible if the subkey difference at round 2 equals `0x00030000`. Indeed, it is quite easy to observe that given any subkey $sk_i = (sk^i_{31} sk^i_{30} \dots sk^i_1, sk^i_0)$ of a round $i < 9$, the following equalities hold for the subkey of round $i + 3$[5]:

$$sk^{i+3}_{27\sim 24} = S_8(sk^i_{20\sim 17}). \tag{1}$$

For $\Delta sk_2 =$ `0x00030000` and $\Delta sk_5 =$ `0x01800000`, Equation 1 would imply that `0x1` $\rightarrow$ `0x1` is a valid transition for S_8, which is false. The authors of [10] kindly shared their source code with us; their model mistakenly uses a state SBox in the key schedule (S_0 and S_1 are used in place of S_8 and S_9, respectively), which explains the discrepancy. We confirmed that using the same incorrect key schedule, our models retrieve the same results. It is clear how crafting models manually can be an error-prone process, not as with fully automatic modeling.

5.3 New 18-round improbable distinguisher for LBlock

The longest impossible differentials covering the full key space of LBlock were identified by Cui et al. [13], who presented several 16-round impossible differentials in the related-key setting. These results cannot be reproduced using the bit-based model due to the limitations discussed earlier in this section. In section D, Table 7 and Table 8 detail the propagation in the subkeys and in the state under this model, for the differential $0 \xrightarrow{16r} 0$, with a single active bit located at k_{11} in the master key.

Using the hybrid model, we recover this differential. The resulting state is shown in Table 9 of section D. We also provide a script that can find other 16-round impossible differentials that were not listed in Cui et al.'s paper (Listing 1.1) and reported them in Table 3. This highlights the hybrid model's advantage in bridging gaps left by the bit-based approach. However, it is still unable

[5] This was also observed in [31]

Table 3: New 16-round impossible differentials found with our model

Δin	Δout	ΔK
(0x00000000,0x00000000)	(0x00000000,0x00000000)	0x0b
(0x00000000,0x00000000)	(0x00000000,0x00000000)	0x580
(0x00000600,0x00000000)	(0x00000000,0x00000000)	0x580

to capture all the other differentials reported by Cui et al. due to differences in our approach compared to theirs.

The strategy described in [13] involves testing the feasibility of a differential model given specific constraints on the input and output; if no solution exists, the differential is deemed impossible. One limitation of this model is that the input and output differences must be fixed in advance but it offers one advantage over ours: if the set of all *differential characteristics* composing a differential can be partitioned into subsets $P = \cup_{p_i}$ such that, for each pair (p_i, p_j), the cell or bit positions of the incompatibility differ, Cui et al.'s model will successfully detect this differential as impossible. In contrast, our truncated deterministic approach will fail to do so. One example among the differentials by Cui et al. is

$$(\texttt{0x00000000}, \texttt{0x00000000}) \xrightarrow[\Delta K = \texttt{0x40}]{16r} (\texttt{0x00000000}, \texttt{0x00000000}).$$

An analysis of the key schedule propagations (shown in Table 10) reveals that the contradiction at round 9 depends on the value of the most significant bit of the first byte of the subkey from round 8: if 1, the differential is impossible due to an inconsistency at byte 15; otherwise,the inconsistency is always at byte 11.

5.4 A new result using probabilistic trails

As observed in subsection 5.3, for LBlock, given fixed plaintext, ciphertext, and key differences, it is possible for the incompatibility to occur at different locations, depending on the S-box outputs of the key schedule. Subkey partitioning has previously been used to manually find impossible differentials [23,31]. Thus, allowing probabilistic differential transitions for the key schedule in our model can lead to the identification of more impossible differentials.

The modeling of the two key schedule S-boxes S_8 and S_9 were changed to encode their standard DDT and the corresponding probability, as done for automated differential cryptanalysis [15,1]. Then, we used this probabilistic hybrid model to enumerate all trails of weight less than 20. Indeed, each transition of S_8 and S_9 has a probability of at least 2^{-3}, and with a small number of active bits in the key, we can expect to have very few active S-boxes even for a high number of rounds. The trails found are then merged to obtain differentials.

For *16 rounds*, we are able to retrieve more of the 16-round impossible differentials found by the authors of [13]. The results are summarized in Table 4. While some missing instances are due to a limitation of our approach, it should be noted we were unable to reproduce the paper's results when either bit k_0

or k_1 of the master key is active, even with the strategy described by the authors. In fact, the SAT differential model generated by CLAASP shows that the differentials are satisfiable.

Table 4: 16-round impossible trails found by [13] tested against our model.

Δin	Δout	ΔK	SAT differential model	our model
0	0 or $1 \ll j$, $j \in \{40, 41, 42, 43\}$	$1 \ll 11$	✓	only $\Delta_{out} = 0$ or $1 \ll j$, $j \in \{43, 42\}$
0	0 or $1 \ll j$, $j \in \{52, 53, 54, 55\}$	$1 \ll 10$	✓	✗
0	0	$1 \ll 6$	✓	✓
0	0 or $1 \ll j$, $j \in \{44, 45, 46, 47\}$	4	✓	✗
0	0	2	✗	✗ ($p = 2^{-0.093}$)
0	0	1	✗	✗ ($p = 2^{-0.093}$)

For *18 rounds*, among the 121 trails of weight less than 20, 6 are part of the following differential

$$(\texttt{0x00000000}, \texttt{0x00000008}) \xrightarrow[\Delta K = \texttt{0x40000000}]{18r} (\texttt{0x00000000}, \texttt{0x00000000}).$$

This impossible differential holds on average for 2^{45} keys and requires a subkey at round 10 of the form $\texttt{0x}\alpha\texttt{0000000}$, where $\alpha = \texttt{0x0}$ or $\texttt{0x8}$, which occurs with probability $2^{-0.83}$. This new trail is reported in Table 11 of section E. Additionally, verification scripts based on CLAASP are provided in Listing 1.2 and Listing 1.3 of section E.

6 Automated key recovery: application to HIGHT

HIGHT is a lightweight block cipher proposed by Hong et al. at CHES 2006 [18]. Its description is provided in section C. The security analysis performed by the authors showed impossible differential results up to 18 rounds. Later works brought that number up to 27 rounds for single-key impossible differential cryptanalysis and 31 rounds for the related-key setting. In [22], Lu presented new impossible differentials reaching 25 rounds (rounds 6-30) in the single-key scenario and 28 rounds (rounds 3-30) in related-key, while removing the initial transformation of the cipher. Ozen et al. extended the attack to 26 (rounds 1-26) and 31 rounds (rounds 1-31) respectively, again without the initial transformation. In [12] Chen et al. finally included the initial transformation, and attacked 27 rounds (rounds 4-30) in the single-key scenario, using hash tables to optimize the attack. This approach was refined by Azimi et al. in [2]. In [27], Rostami et al. show a full-round attack for a class of weak keys in the related-key setting. A summary of all the results can be found in Table 5.

6.1 The ID attack

The general steps for a key recovery attack, given an impossible differential, are as follows.

Table 5: Previous results of impossible differential cryptanalysis of HIGHT

Scenario	Rounds	Transformations	Time	Data	Memory	Ref.
Single-key	18 (1-18)	Both	$2^{109.2}$ enc.	$2^{46.8}$ plaintexts	/	[18]
Single-key	25 (6-30)	Only final	$2^{126.75}$ enc.	2^{60} plaintexts	/	[22]
Single-key	26 (1-26)	Only final	$2^{119.53}$ enc.	2^{61} plaintexts	2^{109} B	[26]
Single-key	27 (4-30)	Both	$2^{126.6}$ enc.	2^{58} plaintexts	2^{120} B	[12]
Single-key	**27 (4-30)**	**Both**	$\mathbf{2^{124.5}}$ **enc.**	$\mathbf{2^{60}}$ **plaintexts**	$\mathbf{2^{116}}$ **B**	**this work**
Single-key	27 (4-30)	Both	$2^{120.58}$ enc. [6]	$2^{59.3}$ plaintexts	$2^{107.4}$ B	[2]
Related-key	28 (3-30)	Only final	$2^{125.99}$ enc.	2^{59} plaintexts	/	[22]
Related-key	31 (1-31)	Only final	$2^{127.28}$ enc.	2^{64} plaintexts	2^{117} B	[26]
Related-key	32	Both	$2^{127.276}$ enc.	2^{64} plaintexts	2^{104} B	[27]

1. **Distinguisher extension**: The attacker propagates the impossible differential trail to the plaintext and ciphertext. The propagation is performed in the backward direction from the start of the differential to the plaintext and in the forward direction from the output difference to the ciphertext.
2. **Pairs generation**: In this phase, pairs satisfying the extended differential are generated. This is usually done by generating structures of 2 sets of plaintexts (resp. ciphertexts), where fixed difference bits are constant, and free difference bits vary so that all plaintexts (resp. ciphertexts) pairs in the structure satisfy the expected difference. The structures are then encrypted (resp. decrypted), and the ones satisfying the output (resp. input) extended difference are kept.
3. **Pairs elimination**: The pairs satisfying each round's difference are propagated deterministically forward from the plaintext and backward from the ciphertext, exhaustively guessing the involved key bits. Since the starting states will contain more unknown difference bits than in the successive states, this process leads to the elimination of some of the initial pairs, namely the ones not satisfying at least one round difference.
4. **Subkeys elimination**: Once the last round of the distinguisher extension is reached, for each remaining pair, all guessed subkeys that satisfy the differential are eliminated.
5. **Exhaustive search**: The remaining keys are exhaustively checked until the correct one is found.

Chen et al. introduced a variation of the attack in [12] that uses hash tables to reduce the time complexity of the pairs elimination step, which is the heaviest one. In particular, portions of the distinguisher's extension are precomputed, in order to efficiently deduce the subkeys leading to the impossible differential when the guessed portion is reached during the pairs elimination phase. Therefore, instead of multiplying the complexities of each step, it is enough to add the sub-extension complexities of the evaluations. The hash table complexity gains are usually computed on top of an existing impossible differential, even though different impossible differentials may be better suited; in the following section, we account for this precomputation step in an impossible differential search model.

[6] Counts one hash table access as a quarter round encryption

6.2 Modeling the search for the distinguisher and its extensions

Our model extends the approach of the bit-based version of [16]. In fact we obtain a finer-grained complexity analysis by including the hash tables precomputation [12] in the objective function.

HIGHT operates on words of 8 bits using the XOR, modular addition, and left rotation. To model these operations, we use the bit-based encoding of [10] (section 2). We recall that each bit difference is represented by a variable with values in the set $\{0, 1, 2\}$, where 0 and 1 correspond to known bit differences of 0 and 1, and 2 represents an unknown bit difference.

We use the high-level language MiniZinc [25] to create this model. In section F, we provide the MiniZinc implementations of the different functions (Listing 1.4). In our standard model (without the use of hash tables), we need the following variables, tables and functions:

- n_I, the end round of the initial extension and beginning round of the impossible differential.
- n_M, the meeting round of the impossible differential.
- n_F, the end round of the impossible differential and beginning round of the final extension.
- n_r the total number of rounds.
- $S_r^b,\ r \in \{0, \ldots, n_r\},\ b \in \{0, \ldots, 7\}$, a table representing the state at round r and byte position b.
- $IS_r^b,\ r \in \{0, \ldots, n_I\} \bigcup \{n_F, \ldots, n_r\},\ b \in \{0, \ldots, 7\}$, a table representing the result of the deterministic propagation in the extension rounds from the plaintext and ciphertext.
- $F_r^b,\ r \in \{0, \ldots, n_I\} \bigcup \{n_F, \ldots, n_r\},\ b \in \{0, \ldots, 7\}$, A binary table listing filtering bytes, identified by the number of bit conditions they enforce.
- $FUse_r^b,\ r \in \{0, \ldots, n_I\} \bigcup \{n_F, \ldots, n_r\},\ b \in \{0, \ldots, 7\}$, a binary table containing the state bytes necessary to the evaluation of the filtering bytes, identified with a 1.
- $KP_i,\ i \in \{0, \ldots, 15\}$, an array of variables containing a permutation of the key bytes determined by the model to optimize the guess and filtering phase.
- $K_r^b,\ r \in \{0, \ldots, n_I\} \bigcup \{n_F, \ldots, n_r\},\ b \in \{0, \ldots, 7\}$, a table with values in $\{-1, \ldots, 15\}$ with, for each state byte active in $FUse$, the maximum of the indexes of the key bytes involved in its evaluation, with respect to the permutation KP (i.e. if the permutation is $(0, 15, 3, 2, 4, \ldots)$ and the byte (r, b) needs the key bytes $0, 2, 3$ it will be $K_r^b = 3$ since $KP_3 = 2$). The value will be a -1 if no key byte is needed or the state byte is not active in $FUse$.
- $H_R : \mathbb{F}_3^{64} \to \mathbb{F}_3^{64}$ a propagation function representing one round of deterministic progagation of HIGHT.
- $Dep : \{0, \ldots, n_r\} \times \{0, \ldots, 7\} \times \{0, \ldots, n_r\} \times \{0, \ldots, 7\} \to \{0, 1\}$ a dependency function such that $Dep(r_1, b_1, r_2, b_2) = 1$ if and only if the byte in position (r_1, b_1) is necessary for the evaluation of (r_2, b_2).
- $KB : \{0, \ldots, n_r\} \times \{0, \ldots, 7\} \to \{0, \ldots, 15\}$ a function giving the key byte necessary for the evaluation of the input byte.

Given the start and end of the impossible differential, the deterministic propagation is modeled using the function H_R and its inverse, to obtain the table S. The same is done starting from the plaintext and ciphertext, using the states of S, to compute IS with the same propagation functions.
From S and IS is easy to compute the table F, just by setting $F_r^b = 1$ iff $S_r^b \neq IS_r^b$, and $FUse$ is then computed thanks to the function Dep.
Finally the table K is evaluated with the use of the functions KB and Dep, the idea is that $K_r^b = max(KB(r,b), max\{KB(i,j), s.t.\ Dep(i,j,r,b) = 1\})$.
The complexity of each step of the guess and filter phase is evaluated by looking at each key byte in the order in which they appear in KP and determining which bytes active in $FUse$ have a value in the table K equal to the index considered. In particular at step i the number of checks to make for each remaining pair is $C_i = C_{i-1} \cdot 2^{8-b_i}$, where $b_i = \sum_{K_r^b = i-1} F_r^b$.

6.3 Complexity evaluation and Objective Function

The data complexity evaluation is similar to [17]. In practice, we implement the complexity of the pairs elimination phase with the set C_i. The heaviest complexity step is the maximum over i of $C_i + \log_2(|\{j\ s.t.\ C_j = C_i\}|)$, where for the logarithm we use the approximation $\log_2(x) = 2.2x - 2$ as in [5].
For the final exhaustive search, the complexity is as shown in [26],

$$\log_2\left(2^{128} \cdot \left(1 - \frac{2^f}{2^{8\cdot(\max(MK)-1)}}\right)^{2^{IP-EP}}\right)$$

where f is the number of bits of the last filtering byte where a fixed known difference has to be forced and 2^{IP} is the number of initial pairs and $EP = \sum_{r,b} F_r^b$ is the logarithm of the number of eliminated pairs. We approximate the function as

$$\log_2\left(2^{128}\left(1 - \frac{1}{2^k}\right)^{2^n}\right) \approx \log_2\left(2^{128} \cdot e^{-2^{n-k}}\right) = 128 - 2^{n-k}\log_2(e)$$

Modeling the Hash Tables. The main improvement of our automatic model compared to others in the literature is the automation of the *hash tables approach* and its integration with the model of the distinguisher search.

Let $IT_1, IT_2, \ldots, IT_T$ denote T tables with binary elements, representing the modeling of T hash tables. A table entry equal to 1 means that the byte in that position will be guessed in the corresponding hash table. Another set of T tables, $OT_1, OT_2, \ldots, OT_T$, represents the bytes that are computed during the creation of the hash table. Among the bytes leading to pairs eliminations, we select T of them as representatives of the hash tables. The constraints used for these tables (apart from the relations among the bytes) represent two properties. First, every

selected filtering byte must be active in at least one of the second set of tables. Moreover, as conditions for the first set of tables, given the filtering bytes related to a hash table, we set as 1 the nearest bytes to the filtering tables, which are involved in its computation and have already been evaluated as outputs in at least one previous table or in the pairs elimination part. An important addition to the model is the modification of the objective function. Next, the pairs filtering must be modeled only on the filtering bytes not involved in the hash tables, thus drastically reducing this part of the total complexity. On the other hand we have to take into account the generation of the tables and the memory accesses. Regarding the $\log_2$ of the first part of the evaluation, we have that

$$\max_{i=1}^{T}(8 \cdot (\sum_{k=0}^{n_r}\sum_{j=0}^{7}(IT_i[k,j]) + TK_i))$$

where TK_i is the number of key bytes to be guessed in the computation of table i. For the second part, in [2] each hash table access is counted as a quarter round encryption, while in [12] as a full round encryption. Given the size of the hash tables it is definitely better to count one access as a full round encryption and we would like to point out that in the second case, our structure would improve the attack of[2]. Each hash table is represented by some filtering bytes, which involve a total of CT_i bit conditions. Moreover, each table also determines as output some key bytes, denoted KT_i. Each table is then accessed a total number of times equal to the number of remaining pairs 2^{RP}, after filtering multiplied by 2 to the power of the sum of the numbers of the free bits after accessing the previous tables:

$$A_i = 2^{RP+\sum_{j=1}^{i-1}(8 \cdot KT_i - CT_i)}$$

The $\log_2$ of the tables' access complexity is approximately

$$\max_{i=1}^{T}(RP + \sum_{j=1}^{i-1}(8 \cdot KT_i - CT_i)) .$$

7 Single key attack improvement

Using our model, we improve the attack complexity on 27-rounds HIGHT in the single-key setting with initial and final transformations. The extension of the trail can be found in Table 6 . The hash tables filtering bytes are highlighted in red, and one of the pairs elimination processes is in blue. The bytes to be guessed in the pairs elimination process are highlighted in green. A detailed review of the attack procedure can be found in section G, where the input bytes of each table have already been guessed or evaluated in previous steps, either the pairs elimination phase or in previous tables. We first detail the filtering process and give a description of the hash tables computation. The permutation of the key bytes guessed in the pairs elimination phase is $(13, 12, 0, 3, 2, 7, 1, 6, 5)$, while the hash tables involve the following filtering bytes (in order of computation): $(S_6^3), (S_8^3), (S_7^3), (S_{25}^0, S_{24}^2), (S_{26}^6)$

Table 6: Our ID trail with extensions for a 27 rounds attack on HIGHT

PT	???????0	10000000	00000000	00000000	????????	????????	????????	????????	
R3	???????0	10000000	00000000	00000000	????????	????????	????????	????????	↑
R4	10000000	00000000	00000000	00000000	????????	????????	????????	???????1	↑
R5	00000000	00000000	00000000	00000000	????????	????????	???????1	10000000	↑
R6	00000000	00000000	00000000	00000000	????????	???????1	10000000	00000000	↑
R7	00000000	00000000	00000000	00000000	???????1	10000000	00000000	00000000	↑
R8	00000000	00000000	00000000	00000000	10000000	00000000	00000000	00000000	↑
				Trail	**start**				
R9	00000000	00000000	00000000	10000000	00000000	00000000	00000000	00000000	↓
R10	00000000	?????100	10000000	00000000	00000000	00000000	00000000	00000000	↓
R11	?????100	10000000	00000000	00000000	00000000	00000000	00000000	????????	↓
R12	10000000	00000000	00000000	00000000	00000000	????????	????????	???????1	↓
R13	00000000	00000000	00000000	????????	????????	????????	???????1	10000000	↓
R14	00000000	????????	????????	????????	????????	???????1	10000000	00000000	↓
R15	????????	????????	????????	????????	???????1	10000000	00000000	????????	↓
R16	????????	????????	????????	???????0	10000000	????????	????????	????????	↓
				Middle	**point**				
R16	????????	????????	????????	???????1	????????	????????	????????	????????	↑
R17	????????	????????	???????1	00000000	????????	????????	????????	????????	↑
R18	????????	???????1	00000000	00000000	????????	????????	????????	????????	↑
R19	???????1	00000000	00000000	00000000	????????	????????	????????	????????	↑
R20	00000000	00000000	00000000	00000000	????????	????????	????????	???????1	↑
R21	00000000	00000000	00000000	00000000	????????	????????	???????1	00000000	↑
R22	00000000	00000000	00000000	00000000	????????	???????1	00000000	00000000	↑
R23	00000000	00000000	00000000	00000000	???????1	00000000	00000000	00000000	↑
				Trail	**end**				
R24	00000000	00000000	00000000	???????1	00000000	00000000	00000000	00000000	↓
R25	00000000	????????	???????1	00000000	00000000	00000000	00000000	00000000	↓
R26	????????	???????1	00000000	00000000	00000000	00000000	00000000	????????	↓
R27	???????1	00000000	00000000	00000000	00000000	????????	????????	????????	↓
R28	00000000	00000000	00000000	????????	????????	????????	????????	???????1	↓
R29	00000000	????????	????????	????????	????????	????????	???????1	00000000	↓
R30	????????	????????	????????	????????	????????	???????1	00000000	????????	↓
CT	????????	????????	????????	????????	????????	????????	???????1	00000000	

Complexity Evaluation. We have to generate 2^{IP} initial pairs satisfying the plaintext and ciphertext truncated differences. From a fixed 64-bit string we can construct 2^{39} plaintexts having the same bit in the positions where the plaintext difference is fixed and another 2^{39} such that the difference with the first ones satisfies the plaintext difference. These sets can generate a total of 2^{78} pairs. Each one of these pairs satisfies the ciphertext condition with probability 2^{-9}, so for each structure, a total of 2^{69} good pairs will remain. Therefore we will need 2^{IP-69} structures of $2^{39} + 2^{39} = 2^{40}$ plaintexts each, resulting in a total amount of 2^{IP-29} initial plaintexts for our attack. A summary of the complexity evaluation of the attack follows (we count C_B as a quarter round encryption, $C_B = \frac{1}{108} C_E$):

- Initial pairs generation: the total complexity of the step is $C_0 = 2^{IP-29}C_E$
- Pairs elimination: the total complexity is the sum of the complexity of each guessing step, in our case $C_1 \leq 2^{IP+29}C_B$
- Hash tables computation: the total complexity is the sum of the complexity of each step, whose leading term in our case is $C_2 \leq (2^8)^{16}C_B$
- Hash tables access: the total complexity is the sum of each the complexity of table access, in our case $C_3 \leq 2^{IP-40} \cdot 2^{72} \cdot 9C_E$
- Exhaustive search: $C_4 = 2^{128} \cdot (1 - \frac{2}{2^{48}})^{2^{IP-40}}$

By setting $IP = 89$, the data complexity is 2^{60} plaintexts and the total time complexity is $C_T = C_0+C_1+C_2+C_3+C_4 \leq 2^{60}C_E+(2^{118}+2^{128})C_B+2^{124.2}C_E \leq 2^{124.5}C_E$. Moreover the memory required for storing the hash tables will be less than $2^{116}B$.

8 Conclusions and future work

We presented an extension of previous techniques to find impossible differential trails by introducing a hybrid model. The model is available in multiple formalisms and combines the advantages of the cell-wise propagation analysis with the precision of the bit-wise approach, while also supporting probabilistic transitions in the related-key setting. We also introduced an effective technique to automatically invert symmetric ciphers. We demonstrated the generality of the techniques above by implementing them in an automated tool, namely CLAASP. We effectively applied the tool to the LBlock cipher to not only recover state-of-the-art results for 16 rounds but also expand them with new 16-round impossible differentials and a new 18-round improbable differential.

Additionally, we developed a new CP model to evaluate the complexity of ID attacks and automatically identify the optimal impossible differential trail for key recovery in the HIGHT block cipher. Our model improves the key-recovery attacks on 27-round HIGHT by identifying a new trail and leveraging the hash tables approach presented in [12].

Future work will aim to generalize our ID attack model for HIGHT to enhance automated tools. This will involve expanding support for complexity evaluation of ID attacks for SAT, SMT, MILP, and CP, with the option to incorporate the hash tables approach. Another promising direction is integrating the differential clustering effect into the model for optimal distinguisher search, which is currently performed as a post-processing step in the hybrid probabilistic model.

References

1. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. (2017)
2. Azimi, S.A., Ahmadi, S., Ahmadian, Z., Mohajeri, J., Aref, M.R.: Improved impossible differential and biclique cryptanalysis of HIGHT. Int. J. Commun. Syst. (2018)

3. Bellini, E., Formenti, M., Gérault, D., Grados, J., Hambitzer, A., Huang, Y.J., Huynh, P., Rachidi, M., Rohit, R., Tiwari, S.K.: Claasping ARADI: automated analysis of the ARADI block cipher. IACR Cryptol. ePrint Arch. (2024)
4. Bellini, E., Gérault, D., Grados, J., Huang, Y.J., Makarim, R.H., Rachidi, M., Tiwari, S.K.: CLAASP: A cryptographic library for the automated analysis of symmetric primitives. In: Selected Areas in Cryptography - SAC (2023)
5. Bellini, E., Gérault, D., Grados, J., Makarim, R.H., Peyrin, T.: Fully automated differential-linear attacks against ARX ciphers. In: CT-RSA (2023)
6. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: EUROCRYPT (1999)
7. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: CRYPTO (1990)
8. Boura, C., Minier, M., Naya-Plasencia, M., Suder, V.: Improved impossible differential attacks against round-reduced lblock. IACR Cryptol. ePrint Arch. (2014)
9. Boura, C., Naya-Plasencia, M., Suder, V.: Scrutinizing and improving impossible differential attacks: Applications to clefia, camellia, lblock and simon. In: ASIACRYPT (2014)
10. Cao, W., Zhang, W., Zhou, C.: New automatic search tool for searching for impossible differentials using undisturbed bits. In: Inscrypt (2022)
11. Chakraborty, D., Hadipour, H., Nguyen, P.H., Eichlseder, M.: Finding complete impossible differential attacks on andrx ciphers and efficient distinguishers for ARX designs. IACR Trans. Symmetric Cryptol. (2024)
12. Chen, J., Wang, M., Preneel, B.: Impossible differential cryptanalysis of the lightweight block ciphers tea, XTEA and HIGHT. In: AFRICACRYPT (2012)
13. Cui, T., Chen, S., Fu, K., Wang, M., Jia, K.: New automatic tool for finding impossible differentials and zero-correlation linear approximations. Sci. China Inf. Sci. (2021)
14. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. J. Cryptol. (2021)
15. Gérault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: CP (2016)
16. Hadipour, H., Gerhalter, S., Sadeghi, S., Eichlseder, M.: Improved search for integral, impossible differential and zero-correlation attacks application to ascon, forkskinny, skinny, mantis, PRESENT and qarmav2. IACR Trans. Symmetric Cryptol. (2024)
17. Hadipour, H., Sadeghi, S., Eichlseder, M.: Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks. In: EUROCRYPT (2023)
18. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A new block cipher suitable for low-resource device. In: CHES (2006)
19. Karakoç, F., Demirci, H., Harmanci, A.E.: Impossible differential cryptanalysis of reduced-round lblock. In: Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems - IFIP (2012)
20. Kim, J., Hong, S., Sung, J., Lee, C., Lee, S.: Impossible differential cryptanalysis for block cipher structures. In: INDOCRYPT (2003)
21. Knudsen, L.: Deal-a 128-bit block cipher. Complexity (1998)
22. Lu, J.: Cryptanalysis of reduced versions of the HIGHT block cipher from CHES 2006. In: ICISC (2007)

23. Minier, M., Naya-Plasencia, M.: A related key impossible differential attack against 22 rounds of the lightweight block cipher lblock. Inf. Process. Lett. (2012)
24. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Inscrypt (2011)
25. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: CP 2007 (2007)
26. Özen, O., Varici, K., Tezcan, C., Kocair, Ç.: Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT. In: ACISP (2009)
27. Rostami, S., Chafjiri, S.B., Tabatabaei, S.A.H.: Related-key impossible differential cryptanalysis of full-round HIGHT. In: SECRYPT (2013)
28. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In: EUROCRYPT (2017)
29. Sun, L., Gérault, D., Wang, W., Wang, M.: On the usage of deterministic (related-key) truncated differentials and multidimensional linear approximations for SPN ciphers. IACR Trans. Symmetric Cryptol. (2020)
30. Tezcan, C.: Improbable differential attacks on present using undisturbed bits. J. Comput. Appl. Math. (2014)
31. Wen, L., Wang, M., Zhao, J.: Related-key impossible differential attack on reduced-round lblock. J. Comput. Sci. Technol. (2014)
32. Wu, W., Zhang, L.: Lblock: A lightweight block cipher. In: ACNS (2011)

A CLAASP automatic cipher inversion: a toy example

The example depicted in Figure 2 can be inverted as follows:

Step 1. $L = [\texttt{cipher\_output\_0\_2}]$. The bits `plaintext[0:2]` can be obtained as they are equal to `cipher_output_0_2[3:5]`.
Step 2. $L = [\texttt{cipher\_output\_0\_2}, \texttt{plaintext[0:2]}]$. The S-box `sbox_0_0` can be **evaluated** as its input bits `plaintext[0:2]` are available.
Step 3. $L = [\texttt{cipher\_output\_0\_2}, \texttt{plaintext[0:2]}, \texttt{sbox\_0\_0}]$. The XOR operation `xor_0_1` can be **inverted** as its output `cipher_output_0_2[0:2]` and its second input `sbox_0_0` are available.
Step 4. $L = [\texttt{cipher\_output\_0\_2}, \texttt{plaintext[0:2]}, \texttt{sbox\_0\_0}, \texttt{inv\_xor\_0\_1}]$. Finally, the last bits `plaintext[3:5]` can be obtained as they are equal to the output of the inverted XOR.

B Description of LBlock

LBlock is a lightweight block cipher proposed by Wu and Zhang at ACNS 2011 [32]. The round function acts on a 64-bit state. It follows a two-branched Feistel structure, where the receiving branch of the Feistel function is first rotated by 8 bits. The inner round function is made of one round key addition, a parallel application of 8 different 4-bit S-boxes and a nibble-wise permutation.

The 80-bit master key K is stored in a key register and denoted as $K = k_{79}k_{78}\cdots k_1k_0$. At each round i, the 32-bit round key sk_i consists of the 64 leftmost bits of the current key state: $k_i = k_{79\sim 48}$. The key update is described in algorithm 2.

```
cipher = {
...
'cipher_rounds' : [
[{# round = 0 - round component = 0
  'id': 'sbox_0_0',
  'type': 'sbox',
  'in_bit_size': 3,
  'in_id_link': ['plaintext'],
  'in_bit_positions': [[0, 1, 2]],
  'out_bit_size': 3,
  'description': [0, 5, 3, 2, 6, 1, 4, 7]},
 {# round = 0 - round component = 1
  'id': 'xor_0_1',
  'type': 'word_operation',
  'in_bit_size': 6,
  'in_id_link': ['sbox_0_0', 'plaintext'],
  'in_bit_positions': [[0, 1, 2], [3, 4, 5]],
   'out_bit_size': 3,
   'description': ['XOR', 2]},
 {# round = 0 - round component = 2
  'id': 'cipher_output_0_2',
  'type': 'cipher_output',
  'in_bit_size': 6,
  'in_id_link': ['xor_0_1', 'plaintext'],
  'in_bit_positions': [[0, 1, 2], [0, 1, 2]],
  'out_bit_size': 6,
  'description': ['cipher_output'],
 }]]
}
```

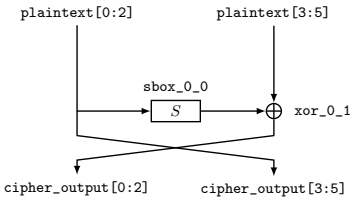


Fig. 2: A toy example and its CLAASP representation.

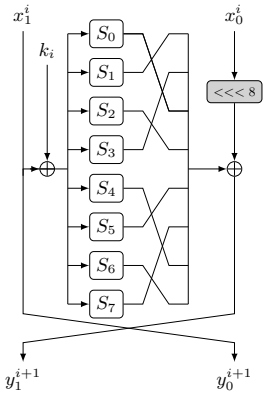


Fig. 3: One round of the LBlock cipher.

C Description of HIGHT

HIGHT is a lightweight block cipher proposed by Hong et al. at CHES 2006 [18]. The round function acts on a 64-bit state. Its round function can be seen as a concatenation of 4 Feistel structures acting on pairs of bytes which are then rotated. The key is inducted alternately on the bytes by XOR and modular

Algorithm 2: Key schedule of LBlock

$sk_0 = K_{79\sim 48}$;
for $1 \leq r \leq 31$ **do**
  $k_{79\sim 0} \leftarrow k_{79\sim 0} \lll 29$;
  $k_{79\sim 76} \leftarrow S_9(k_{79\sim 76})$;
  $k_{75\sim 72} \leftarrow S_8(k_{75\sim 72})$;
  $k_{50\sim 47} \leftarrow k_{50\sim 47} \oplus [i]_2$;
  $sk_{5r} \leftarrow k_{79\sim 48}$;

addition, while the state operations are rotations and either XOR, if the key is used with the modular addition, or modular addition. The 128-bit master key K is used for an initial transformation and then at each round the 32-bit subkey involved in the state is evaluated via modular addition with fixed constants of previous subkeys.

The 128-bit initial key MK is used for the computation of the 32-bit round keys SK_i and the initial and final whitening keys WK (algorithm 3 and algorithm 4). The initial and final whitening are performed at the beginning and end of the encryption by adding or XORing the whitening key to the plaintext and to the last round output respectively, as shown in algorithm 5. From now on we will indicate the i-th byte of the r-th round output as S_r^i.

Algorithm 3: Whitening key schedule of HIGHT

for $0 \leq i \leq 3$ **do**
  $WK_i \leftarrow MK_{i+12}$;
for $4 \leq i \leq 7$ **do**
  $WK_i \leftarrow MK_{i-4}$;

Algorithm 4: Key schedule of HIGHT

$\delta_0 \leftarrow 1011010$;
for $1 \leq i \leq 127$ **do**
  $\delta_i \leftarrow ((\delta_{i-1} \oplus (\delta_{i-1} \gg 3)) \ll 6) \oplus (\delta_{i-1} \gg 1)$;
for $0 \leq i \leq 7$ **do**
  for $0 \leq j \leq 7$ **do**
    $SK_{16i+j} \leftarrow MK_{j-1 \mod 8} \boxplus \delta_{16i+j}$;
  for $0 \leq j \leq 7$ **do**
    $SK_{16i+j+8} \leftarrow MK_{(j-1 \mod 8)+8} \boxplus \delta_{16i+j+8}$;

Algorithm 5: Initial and final whitening of HIGHT

$S_0^0 \leftarrow P_0$; $S_0^1 \leftarrow P_1 \oplus WK_3$; $S_0^2 \leftarrow P_2$; $S_0^3 \leftarrow P_3 \boxplus WK_2$; $S_0^4 \leftarrow P_4$;
$S_0^5 \leftarrow P_5 \oplus WK_1$; $S_0^6 \leftarrow P_6$; $S_0^7 \leftarrow P_7 \boxplus WK_0$;
ENC(S);
$C_0 \leftarrow S_{32}^7$; $C_1 \leftarrow S_{32}^0 \oplus WK_7$; $C_2 \leftarrow S_{32}^1$; $C_3 \leftarrow S_{32}^2 \boxplus WK_6$; $C_4 \leftarrow S_{32}^3$;
$C_5 \leftarrow S_{32}^4 \oplus WK_5$; $C_6 \leftarrow S_{32}^5$; $C_7 \leftarrow S_{32}^6 \boxplus WK_4$;

The ENC(S) encryption function consists of 32 applications of the round function (described in algorithm 6). This function needs the definition of two auxiliary functions, F_0 and F_1, which are defined as follows:

$$F_0(x) = (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 7)$$

$$F_1(x) = (x \lll 3) \oplus (x \lll 4) \oplus (x \lll 6)$$

Algorithm 6: Round function of HIGHT

for $0 \leq r \leq 31$ **do**
  for $0 \leq i \leq 3$ **do**
    $S_{r+1}^{2i} \leftarrow S_r^{2i+1}$
  for $0 \leq i \leq 1$ **do**
    $S_{r+1}^{4i-1 \mod 8} \leftarrow S_r^{4i} \oplus (F_1(S_{r-1}^{4i+1}) \boxplus SK_{4r+3-2i})$
  for $0 \leq i \leq 1$ **do**
    $S_{r+1}^{4i+1} \leftarrow S_r^{4i+2} \boxplus (F_1(S_{r-1}^{4i+3}) \oplus SK_{4r+2-2i})$

A graphic representation of the round function can be found in Figure 4.

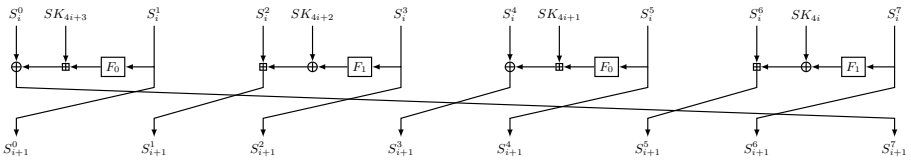


Fig. 4: One round of the HIGHT cipher.

D 16-round impossible differential for LBlock by [13]

Table 7: Deterministic subkeys used in the 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ of [13].

0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0010	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
?1??	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	00?1	??00	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	000?	1??0	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	?1??	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	00?1

Table 8: State of the the 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ of [13] using a bit-wise model.

0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	???1	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	????	0000	0000	0000	0000	0000	0000	0000	0000	0000	???1	0000	0000	0000	0000
????	???1	0000	0000	0000	0000	0000	0000	0000	????	0000	0000	0000	0000	0000	0000
????	0000	????	0000	0000	0000	0000	????	????	???1	0000	0000	0000	0000	0000	0000
0000	0000	????	????	0000	????	????	???1	????	0000	????	0000	0000	0000	0000	????
????	????	0000	????	????	????	????	????	0000	0000	????	????	0000	????	????	???1
????	????	????	????	????	????	????	????	????	????	????	????	????	????	????	????
????	????	????	????	????	????	????	????	????	????	????	????	????	????	????	????
0000	????	????	????	0000	????	0000	????	????	????	????	????	????	????	????	????
0000	????	????	0000	0000	0000	????	????	0000	????	????	????	0000	????	0000	????
0000	????	0000	0000	0000	????	0000	????	0000	????	????	0000	0000	0000	????	????
0000	0000	0000	0000	0000	0000	0000	????	0000	????	0000	0000	0000	????	0000	????
0000	0000	0000	0000	0000	0000	0000	????	0000	0000	0000	0000	0000	0000	0000	????
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	????
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Listing 1.1: CLAASP script to find 16-round impossible trails with the hybrid model, in the related-key setting

```
from claasp.cipher_modules.models.cp.mzn_models.
    mzn_hybrid_impossible_xor_differential_model import
    MznHybridImpossibleXorDifferentialModel
from claasp.ciphers.block_ciphers.lblock_block_cipher import
    LBlockBlockCipher
```

Table 9: State of the 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ of [13] using the hybrid model.

0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	2222	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	5555	0000	0000	0000	0000	0000	0000	0000	0000	0000	2222	0000	0000	0000	0000
4444	2222	0000	0000	0000	0000	0000	0000	0000	5555	0000	0000	0000	0000	0000	0000
6666	0000	????	0000	0000	0000	0000	5555	4444	2222	0000	0000	0000	0000	0000	0000
0000	0000	8888	????	0000	2222	4444	2222	6666	0000	????	0000	0000	0000	0000	5555
????	????	0000	5555	????	????	6666	????	0000	0000	8888	????	0000	2222	4444	2222
4444	????	8888	????	????	????	5555	????	????	????	????	????	????	????	????	????
????	8888	????	????	6666	4444	8888	????	4444	????	8888	????	????	????	5555	????
0000	????	4444	6666	0000	6666	0000	????	????	8888	????	????	6666	4444	8888	????
0000	4444	6666	0000	0000	0000	????	8888	0000	????	4444	6666	0000	6666	0000	????
0000	6666	0000	0000	0000	????	0000	2222	0000	4444	6666	0000	0000	0000	????	8888
0000	0000	0000	0000	0000	0000	0000	4444	0000	6666	0000	0000	0000	????	0000	2222
0000	0000	0000	0000	0000	0000	0000	6666	0000	0000	0000	0000	0000	0000	0000	4444
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	6666
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

```
from claasp.cipher_modules.models.utils import set_fixed_variables
lblock = LBlockBlockCipher(number_of_rounds=16)
mzn = MznHybridImpossibleXorDifferentialModel(lblock)
rk_key = [set_fixed_variables('key', 'not_equal', range(80), [0]*80)]
mzn.find_all_impossible_xor_differential_trails(16, rk_key, 'Chuffed', 1, 8,
    16, intermediate_components=False)
```

E 18-round improbable differential for LBlock

Listing 1.2: CLAASP script to verify the improbable differential

```
from claasp.ciphers.block_ciphers.lblock_block_cipher import
    LBlockBlockCipher
lblock = LBlockBlockCipher(number_of_rounds=18)
from claasp.cipher_modules.models.utils import set_fixed_variables
from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model
    import SatXorDifferentialModel
sat = SatXorDifferentialModel(lblock)
key = set_fixed_variables(component_id='key', constraint_type='equal',
    bit_positions=range(80), bit_values=[0]*49+[1]+[0]*30)
pt = set_fixed_variables(component_id='plaintext', constraint_type='equal',
    bit_positions= range(64), bit_values= [0]*60+[1,0,0,0])
ct = set_fixed_variables(component_id='intermediate_output_17_12',
    constraint_type='equal', bit_positions= range(64), bit_values= [0]*64)
key_proba = [set_fixed_variables(component_id='intermediate_output_10_0',
    constraint_type='equal', bit_positions = range(1,4), bit_values= [0]*3)]
sat.find_one_xor_differential_trail(fixed_values=key_proba + [key, pt, ct])
```

Listing 1.3: CLAASP script to verify the probability of the 18-round differential

Round	Subkey
0	0000 0000 0000 0000 0000 0000 0000 0000
1	0000 0000 0000 0000 0000 0000 0000 0000
2	0000 0000 0000 0001 0000 0000 0000 0000
3	0000 0000 0000 0000 0000 0000 0000 0000
4	0000 0000 0000 0000 0000 0000 0000 0000
5	0000 0000 1000 0000 0000 0000 0000 0000
6	0000 0000 0000 0000 0000 0000 0000 0000
7	0000 0000 0000 0000 0000 0000 0000 0010
8	$\alpha\beta\gamma\delta$ 0000 0000 0000 0000 0000 0000 0000
9	0000 0000 0000 0000 0000 0000 0000 0000
10	0000 0000 0000 0000 0000 00$\alpha\beta$ $\gamma\delta$00 0000
11	0000 0000 0000 0000 0000 0000 0000 0000
12	0000 0000 0000 0000 0000 0000 0000 0000
13	0000 0000 0000 000α $\beta\gamma\delta$0 0000 0000 0000
14	0000 0000 0000 0000 0000 0000 0000 0000
15	0000 0000 0000 0000 0000 0000 0000 0000

Table 10: The LBlock key schedule over 16 rounds, when $\Delta K = 0x40$, in the bit-wise deterministic truncated model. The differential $0 \xrightarrow{\Delta K=0x40} 0$ is impossible due to a contradiction at byte 11 of round 9 if $\alpha = 0$ and at byte 15 of round 9 otherwise.

```
from claasp.cipher_modules.models.cp.mzn_models.
    mzn_hybrid_impossible_xor_differential_model import
    MznHybridImpossibleXorDifferentialModel
from claasp.ciphers.block_ciphers.lblock_block_cipher import
    LBlockBlockCipher
from claasp.cipher_modules.models.utils import set_fixed_variables
lblock = LBlockBlockCipher(number_of_rounds=18)
mzn = MznHybridImpossibleXorDifferentialModel(lblock)
fixed_variables = [set_fixed_variables(component_id='key', constraint_type='
    equal', bit_positions=range(80), [0]*49+[1]+[0]*30)]
fixed_variables.append(set_fixed_variables(component_id='plaintext',
    constraint_type='equal', range(64), [0]*60 +[1,0,0,0]))
fixed_variables.append(set_fixed_variables('inverse_cipher_output_17_19', '
    equal', range(64), [0]*64))
mzn.find_all_impossible_xor_differential_trails(18, fixed_variables, 'Chuffed
    ', 1, 9, 18, intermediate_components=False, probabilistic=True)
```

F MiniZinc predicates for the HIGHT model

```
% Xor of two bits
function array[int] of var 0..2: Xor2(array[int] of var 0..2: a, array[int]
    of var 0..2: b)=
array1d(0..(length(a)-1), [if (a[j] == 2 \/ b[j] == 2) then 2 else ((a[j]+b[j
    ]) mod 2) endif | j in 0..(length(a)-1)]);

% Modular Addition of two words
predicate Modadd_2(array[int] of var 0..2: a, array[int] of var 0..2: b,array
    [int] of var 0..2: c) = (
  let {
    array [0..length(a)-1] of var 0..2: as = LShift(a,1),
    array [0..length(a)-1] of var 0..2: bs = LShift(b,1),
```

```
    array [0..length(a)-1] of var 0..2: cs = LShift(c,1),
    var 0..length(a)-1: pivot;
  } in
  forall (i in 0..length(a)-1) (
    if i<pivot then c[i]=2 else (as[i]=0) /\ (bs[i]=0) /\ (cs[i]=0) endif
  ) /\
  xor_bit_p1_2(a[pivot],b[pivot],c[pivot])
  /\ if pivot>0 then a[pivot]+b[pivot]>0 else true endif
);

% Left Rotation
function array[int] of var 0..2: LRot(array[int] of var 0..2: X, var int: val
    )=
array1d(0..(length(X)-1), [X[(j+val) mod length(X)] | j in 0..(length(X)-1)])
    ;
```

Listing 1.4: The MiniZinc predicates representing the operations of HIGHT

G Full procedure of the attack on 27-round HIGHT

Table 11: Our 18-round improbable trail (cell-wise notation): 0 for null difference, * for unknown nonzero, and ? for fully undetermined. Conditional subkey transitions are highlighted in blue, with $\alpha = 0x0$ or 0x8.

State (truncated)	Subkey
(00000000, 00000008)	00000000
(00000800, 00000000)	00000800
(00000000, 00000800)	00000000
(00080000, 00000000)	00000000
(0*000000, 00080000)	00040000
(*?000000, 0*000000)	00000000
(?0*0000*, *?000000)	00000000
(00?*0**?, ?0*0000*)	0*000000
(?*0?*??*, 00?*0**?)	00000000
(???*??*?, ?*0?*??*)	000000?*
(?*?*?**?, ??*?????)	
(0?0*?***, ?*?*?**?)	α0000000
(?00*0*?0, 0?0*?***)	00000000
(000*0*00, ?00*0*?0)	0000?*00
(000*0000, 000*0*00)	00000000
(000*0000, 000*0000)	00000000
(00000000, 000*0000)	000*0000
(00000000, 00000000)	00000000
(00000000, 00000000)	00000000
(00000000, 00000000)	

Table 12: Impossible differential attack procedure on 27 rounds of HIGHT

Step	Description	Time complexity
	Pairs elimination	
1	Guess MK_{13} and compute S_3^5, S_4^3 by partial decryption for each initial pair, then eliminate those whose difference does not propagate to the expected one (on average 1 pair every 2^8 will remain)	$2 \cdot 2^{IP} \cdot (2 \cdot 2^8) C_B$
2	Guess MK_{12}, MK_0 and compute S_3^7, S_4^5, S_5^3 by partial encryption and S_{30}^6 by partial decryption for each remaining pair, and filter	$2 \cdot 2^{IP-8} \cdot (3 \cdot 2^{16} + 2^{24}) C_B$
3	Guess MK_3 and compute S_{30}^0, S_{29}^0 by partial decryption for each remaining pair, and filter	$2 \cdot 2^{IP-16} \cdot (2 \cdot 2^{32}) C_B$
4	Guess MK_2, MK_7 and compute $S_{30}^2, S_{29}^2, S_{28}^2$ by partial decryption for each remaining pair, and filter	$2 \cdot 2^{IP-24} \cdot (2^{40} + 2 \cdot 2^{48}) C_B$
5	Guess MK_1, MK_6 and then MK_{15} bit by bit and compute $S_{30}^4, S_{29}^4, S_{28}^4, S_{27}^4$ by partial decryption for each remaining pair, and filter	$2 \cdot 2^{IP-32} \cdot (2^{56} + 2 \cdot 2^{64} + 2^{65}) C_B$
	Memory tables generation ($S_i^{j,\Delta} = \{S_i^j, \Delta S_i^j\}$)	
6	Build table T_0 by partial encryption, guessing bytes $P_0, \Delta P_0, P_1, P_7^{\Delta}, S_4^{5,\Delta}$, $C_6, S_{29}^0, MK_0, MK_4, MK_7, MK_{15}$. The condition $\Delta S_6^3 = 0$ is a 8-bit condition, therefore if we index the table by $P_0^{\Delta}, P_1, P_7^{\Delta}, S_4^{5,\Delta}, C_6, S_{29}^0, MK_0, MK_7, MK_{15}$ and take as outputs MK_4 and the implied $S_4^{7,\Delta}, S_5^{5,\Delta}, S_6^3, S_{28}^0$, for each set of inputs we get on average 1 output.	$2 \cdot (2^8)^{12} \cdot 2^7 C_B$
7	Build table T_1 by partial encryption, guessing bytes $P_1, S_3^2, P_3, S_4^3, S_4^{7,\Delta}$, $MK_1, MK_2, MK_3, MK_6, MK_{12}, MK_{14}, MK_{15}$. The condition $\Delta S_8^3 = 0$ is a 7-bit condition, therefore if we index the table by $P_1, S_3^2, P_3, S_4^3, S_4^{7,\Delta}, S_{28}^0, S_{27}^2, MK_1, MK_2, MK_3, MK_6, MK_{12}, MK_{15}$ and take as outputs MK_{14} and the implied $S_4^1, S_5^1, S_5^7, S_6^7, S_6^{5,\Delta}, S_7^5, S_8^3$, for each set of inputs we get on average 2 outputs.	$2 \cdot (2^8)^{12} \cdot 2^7 C_B$
8	Build table T_2 by partial encryption, guessing bytes $S_5^{5,\Delta}, S_6^{5,\Delta}, S_{29}^0$, $S_{28}^0, S_{28}^2, MK_8, MK_{12}$. The condition $\Delta S_7^3 = 0$ is a 8-bit condition, therefore if we index the table by $S_5^{5,\Delta}, S_6^{5,\Delta}, S_{29}^0, S_{28}^0, S_{28}^2, MK_{12}$ and take as outputs MK_8 and the implied S_{27}^2, S_{26}^2, for each set of inputs we get on average 2 outputs.	$2 \cdot (2^8)^8 \cdot 2^7 C_B$
9	Build table T_3 by partial decryption, guessing bytes $S_{26}^2, S_{28}^0, S_{29}^{4,\Delta}, S_{30}^{4,\Delta}$, $S_{30}^6, S_{30}^{5,\Delta}, MK_0, MK_1, MK_4, MK_5, MK_9, MK_{13}, MK_{14}$. The conditions $\Delta S_{25}^0 = 0, \Delta S_{24}^2 = 0$ are a 16-bit condition, therefore if we index the table by $S_{26}^2, S_{28}^0, S_{29}^{4,\Delta}, S_{30}^{4,\Delta}, S_{30}^6, S_{30}^{5,\Delta}, MK_0, MK_1, MK_4, MK_{13}, MK_{14}$ and take as outputs MK_5, MK_9 and the implied $S_{27}^{6,\Delta}$, for each set of inputs we get on average 2 outputs.	$2 \cdot (2^8)^{15} \cdot 2^7 C_B$
10	Build table T_4 by partial decryption, guessing bytes $S_{27}^{6,\Delta}, S_{28}^{4,\Delta}, MK_{10}$. The condition $\Delta S_{26}^6 = 0$ is a 8-bit condition, therefore if we index the table by $S_{27}^{6,\Delta}, S_{28}^{4,\Delta}$ and take as outputs MK_{10}, for each set of inputs we get on average 2 valid outputs, and finally one can filter wrong keys.	$2 \cdot (2^8)^5 C_B$
11	Each evaluation of the first 5 steps corresponds to some evaluated bytes which will be the inputs of the hash tables, along with the outputs of the previous tables, therefore for each remaining pair and subkeys guesses for steps 1-5, the tables can be accessed and they produce some wrong subkeys (2 per guess in particular).	
12	The remaining keys are exhaustively tested.	