# tutorial_impossible_differential_trail_search

April 8, 2025

## 1 Impossible Differential search with CLAASP

In this notebook, we will have a look at the newly added Impossible Differential module of CLAASP. It offers 3 models: - a word-based model - a bit-based model - a hybrid model that extends the bit-based model with word-based properties

### 1.1 Impossible Differential Cryptanalysis

Given a block cipher $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$, differential cryptanalysis studies how a specific input difference $\delta$ propagates to an output difference $\gamma$ through the cipher. More precisely, an attacker is interested in pairs $(\delta, \gamma)$ such that

$$E_K(X) \oplus E_K(X \oplus \delta) = \gamma$$

hold with a high probability over all keys $K$ and plaintexts $X$.

Conversely, **impossible differential cryptanalysis** focuses on differentials with probability 0: an impossible differential for $E$ is a pair $(\delta, \gamma)$ such that:

$$E_K(X) \oplus E_K(X \oplus \delta) \neq \gamma$$

for all keys $K$ and plaintexts $X$.

First, we instantiate the cipher class representing the analyzed cipher. In this example, we will look at 6 rounds of Speck-32-64

[3]:
```
from claasp.ciphers.block_ciphers.speck_block_cipher import SpeckBlockCipher
speck6 = SpeckBlockCipher(number_of_rounds=6)
```

Next, we instantiate the SAT Differential model class with the cipher object.

[4]:
```
from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model␣
  ↪import SatXorDifferentialModel
sat = SatXorDifferentialModel(speck6)
```

Then, we define the differential whose validity we want to check. Here, we set both the plaintext and ciphertext differences to `0x82c08101` and fix the key difference to `0`. This is achieved using the `set_fixed_variables` method, which requires the values to be provided as binary lists. To simplify this, we can also use the `integer_to_bit_list` method, which converts an integer into its binary representation.

[5]:
```
from claasp.cipher_modules.models.utils import set_fixed_variables,␣
 ↪integer_to_bit_list
ciphertext_id =speck6.get_all_components()[-1].id
pt_value = 0x82c08101
ct_value = 0x82c08101
differential_0 = [
    set_fixed_variables(component_id='key', constraint_type='equal',␣
 ↪bit_positions=list(range(64)), bit_values=[0] * 64),
    set_fixed_variables(component_id='plaintext', constraint_type='equal',␣
 ↪bit_positions=list(range(32)),␣
 ↪bit_values=integer_to_bit_list(int_value=pt_value, list_length=32,␣
 ↪endianness='big')),
    set_fixed_variables(component_id=ciphertext_id, constraint_type='equal',␣
 ↪bit_positions=list(range(32)),␣
 ↪bit_values=integer_to_bit_list(int_value=ct_value, list_length=32,␣
 ↪endianness='big'))
    ]
```

Finally, we ask the model to find one trail satisfying the differential:

[6]:
```
from claasp.cipher_modules.report import Report

trail = sat.find_one_xor_differential_trail(fixed_values=differential_0,␣
 ↪solver_name='KISSAT_EXT')

def print_differential(trail):
    if trail['status'] != 'UNSATISFIABLE':
        # reformat the trail to make the hex values more explicit
        trail["components_values"] = {k: {**v, "value": "0x" + v["value"]} for␣
 ↪k, v in trail["components_values"].items()}
        report = Report(trail)
        report.show()
    else:
        print(f"The differential({hex(pt_value)}, {hex(ct_value)}) is␣
 ↪impossible.")

print_differential(trail)
```

```
1 _ _ _ _ _ 1 _ 1 1 _ _ _ _ _ _
1 _ _ _ _ _ _ 1 _ _ _ _ _ _ _ 1    plaintext

_ _ 1 1 1 1 1 _ _ _
1 1 1 _ 1 _ _ _ 1
1 1 _ 1 _ _ _ 1 1
1 1 _ _    intermediate_output_0_6

1 1 _ _ _ 1 1 1
1 _ 1 _ _ _ _ _ _ _ 1 _ 1
```

```
1 1 1 _ 1 _ 1 _ _ _ _
intermediate_output_1_12

1 1 1 _ 1 1 _ _
1 _ 1 1 1 _ _ 1 _
1 _ 1 _ _ _ 1 1 1
1 1 1 _ _ 1
intermediate_output_2_12

1 1 1 _ 1 1 _
1 1 1 _ 1 1 1
1 _ 1 _ 1 _ 1 _ 1 _ _ _
1 1 1 _ 1 1
intermediate_output_3_12

1 1 1 _ 1 _ _ _ _ _ _ 1
1 1 1 _ _ 1 _ _ _ _ _ _ 1
1 1 1 _ _ _ _    intermediate_output_4_12

1 _ _ _ _ _ 1 _ 1 1 _ _ _ _ _ _
1 _ _ _ _ _ _ 1 _ _ _ _ _ _ _ 1
cipher_output_5_12


total weight = 66.0
```

Here the model returned a trail. Let us try different values:

[7]:
```
pt_value = 0x1000
ct_value = 0x80008002
differential_1 = [
    set_fixed_variables(component_id='key', constraint_type='equal',␣
 ↪bit_positions=list(range(64)), bit_values=[0] * 64),
    set_fixed_variables(component_id='plaintext', constraint_type='equal',␣
 ↪bit_positions=list(range(32)),␣
 ↪bit_values=integer_to_bit_list(int_value=pt_value, list_length=32,␣
 ↪endianness='big')),
    set_fixed_variables(component_id=ciphertext_id, constraint_type='equal',␣
 ↪bit_positions=list(range(32)),␣
 ↪bit_values=integer_to_bit_list(int_value=ct_value, list_length=32,␣
 ↪endianness='big'))
    ]

trail = sat.find_one_xor_differential_trail(fixed_values=differential_1,␣
 ↪solver_name='KISSAT_EXT')
print_differential(trail)
```

```
The differential(0x1000, 0x80008002) is impossible.
```

While this approach works to identify impossible differentials, using a standard differential model is not well-suited to the task: - classical differential distinguisher requires showing the existence of one differential trail - on the other hand, an impossible differential requires proving that such a property does not exist

Given the size of the search space, it is significantly easier to show that a property exists for some choice of input and output difference, than to prove that no trail connecting them exists.

Finding impossible differentials with this model requires fixing the input and output iteratively to all possible values, and asking the solver to find a solution; if none is found, then the propagation is impossible. Such approach was proposed by two independent works by Sasaki and Todo (2017) and Cui et al. (2016). The following method does so for every value of Hamming weight 1:

[8]:
```
impossible_differentials = speck6.impossible_differential_search("smt",␣
  ↪"YICES_EXT", scenario="single-key")
print(f"Found {len(impossible_differentials)} impossible differentials.")
for d_in, d_out in impossible_differentials:
    print(f'({hex(d_in)}, {hex(d_out)})')
```

```
Found 3 impossible differentials.
(0x400000, 0x1)
(0x400000, 0x2)
(0x400000, 0x8000)
```

Similarly, it is possible to find zero-correlation linear approximations.

[11]:
```
zero_correlation_linear_approximations = speck6.
  ↪zero_correlation_linear_search("smt", "YICES_EXT")
print(f"Found {len(zero_correlation_linear_approximations)} zero-correlation␣
  ↪linear approximations.")
```

```
Found 743 zero-correlation linear approximations.
```

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. Next, we will look at a different approach.

---

### 1.2 Miss-in-the-Middle Approach

The cipher is analyzed by propagating differences both forward and backward through deterministic rules. A contradiction at an intermediate state indicates that no full differential trail exists. This shifts the problem from proving unsatisfiability to demonstrating a contradiction in the middle of the propagation, removing the need for explicit enumeration.

**Related Works:**

- **Sun et al. (2020):** Proposed a CP-based model for the deterministic propagation of truncated differences in cell-oriented ciphers.
- **Cao et al. (2021):** Adapts the previous framework for an MILP-based approach that incorporates bit-level propagations and uses the concept of "undisturbed bits" i.e. propagation

that occur with probability 1 for each operation.

- **Hadipour et al. (2023, 2024)** Tools such as Zero and Zeroplus expand on these ideas to propose a unified framework that removes the need to specify the contradiction round and takes into account the complexity of the key recovery part.

The encoding can be split into two types of models: the **bit-based** model and the **word-based** model. Both are available in CLAASP.

---

### 1.2.1 1. Bit-Oriented Model

**Data Representation:** The $n$-bit internal state is represented bit by bit:

$$\Delta X = (\Delta X_0, \Delta X_1, \dots, \Delta X_{n-1}),$$

where each bit difference is modeled with:

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

*Example (S-box application $Y = S(X)$)*

The S-box is modeled using its Difference Distribution Table (DDT) restricted to **undisturbed bits**. For a given input difference $\Delta X$, let $P_{\Delta X}$ be the set of positions of undisturbed bits. Then the output difference is given by:

$$\delta Y_i = \begin{cases} b_i & \text{if } i \in P_{\Delta X} \\ 2 & \text{otherwise,} \end{cases}$$

where $b_i$ is the undisturbed bit value. If $P_{\Delta X}$ is empty, then

$$\delta Y = (2, 2, \dots, 2).$$

For Present:

$0000 \to 0000, \quad 0001 \to ???1, \quad 1000 \to ???1, \quad 1001 \to ???0$

```
16  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  .  4  .  .  .  4  .  4  .  .  .  4  .  .
 .  .  .  2  .  4  2  .  .  .  2  .  2  2  2  .
 .  2  .  2  2  .  4  2  .  .  2  2  .  .  .  .
 .  .  .  .  .  4  2  2  .  2  2  .  2  .  2  .
 .  2  .  .  2  .  .  .  .  2  2  2  4  2  .  .
 .  .  2  .  .  .  2  .  2  .  .  4  2  .  .  4
 .  4  2  .  .  .  2  .  2  .  .  .  2  .  .  4
 .  .  .  2  .  .  .  2  .  2  .  4  .  2  .  4
 .  .  2  .  4  .  2  .  2  .  .  .  2  .  4  .
 .  .  2  2  .  4  .  .  2  .  2  .  .  2  2  .
 .  2  .  .  2  .  .  .  4  2  2  2  .  2  .  .
```

```
.  .  2  .  .  4  .  2  2  2  2  .  .  .  2  .
.  2  4  2  2  .  .  2  .  .  2  2  .  .  .  .
.  .  2  2  .  .  2  2  2  2  .  .  2  2  .  .
.  4  .  .  4  .  .  .  .  .  .  .  .  .  4  4
```

**Incompatibility detection**

At a round $r$, if the forward ($\delta X_i^F$) and backward ($\delta X^B{}_i$) bits satisfy:

$$\delta X_i^F + \delta X_i^B = 1$$

---

**In CLAASP** Once again, we will look at 6 rounds of Speck-32-64:

[9]:
```
from claasp.ciphers.block_ciphers.speck_block_cipher import SpeckBlockCipher
speck6 = SpeckBlockCipher(number_of_rounds=6)
```

Let us also define a simple printing method that will print the trail in nicer way:

[10]:
```
def trail_pretty_print(trail, wordbased=False, hybrid=False):
    if trail['status'] == 'UNSATISFIABLE':
        print("No impossible differential found.")
        return

    translation_table = str.maketrans("02", ".?")
    if hybrid:
        for key in trail['components_values']:
            if key == "plaintext":
                translated = "Plaintext"
            elif re.match(r"(inverse_)?intermediate_output_\d+_0", key):
                translated = f"Round key {key.split('_')[-2]}"
            elif re.match(r"(inverse_)?intermediate_output_\d+_4", key):
                translated = f"Round output {key.split('_')[-2]}"
            elif "cipher_output_" in key:
                translated = "Cipher output"
            else:
                continue
            print(f"{translated}:".ljust(20),␣
↪trail['components_values'][key]['value'])
        return

    if wordbased:
        translation_table = str.maketrans("032", ".?*")
        if trail['status'] != 'UNSATISFIABLE':
            max_key_length = max(len(str(k)) for k in␣
↪trail['components_values'])
            for k, v in trail['components_values'].items():
                index = k.split("_")[-2:][0]
                if 'backward' in k:
```

```
                    index = k.split("_")[-3:][0]
                    print("----------------------------------")
                if 'plaintext' in k:
                    print(f"{'Plaintext':<{max_key_length}}: {v['value'].
↪translate(translation_table)}")
                if 'intermediate_output' in k:
                    print(f"{'Round ' + str(index):<{max_key_length}}:␣
↪{v['value'].translate(translation_table)}")
                if 'cipher_output' in k:
                    print(f"{'Ciphertext':<{max_key_length}}: {v['value'].
↪translate(translation_table)}")

        return

    components = trail['components_values']

    def get_translated_value(key):
        return components[key]['value'].translate(translation_table) if key in␣
↪components else ""

    lines = [
        ("key", get_translated_value("key"), ""),
        ("plaintext", get_translated_value("plaintext"), "")
    ]

    intermediates = {}
    inverses = {}

    for name in components.keys():
        if "intermediate_output" in name and "inverse" not in name:
            intermediates[name] = get_translated_value(name)
        elif "inverse_intermediate_output" in name:
            inverses[name] = get_translated_value(name)

    all_indices = set(intermediates.keys()) | set(inverses.keys())

    extract_index = lambda k: tuple(map(int, k.split("_")[-2:]))
    sorted_keys = sorted(all_indices, key=extract_index)
    already_processed = []
    for key in sorted_keys:
        base_key = key.replace("inverse_", "")
        extract_index = key.split("_")[-2:][0]
        if base_key in already_processed:
            continue
        elif key in intermediates:
            inter_value = intermediates.get(key, "")
            inverse_value = inverses.get("inverse_" + key, "")
```

```
        elif key in inverses:
            inverse_value = inverses.get(key, "")
            inter_value = intermediates.get(base_key, "")
        lines.append((f"Round {extract_index}", inter_value, inverse_value))
        already_processed.append(base_key)

    cipher_key = [key for key in components if "cipher_output" in key][0]
    lines.append(("Output", "", get_translated_value(cipher_key)))

    col1_width = max(len(label) for label, _, _ in lines) + 2
    col2_width = max(len(val) for _, val, _ in lines) + 2
    col3_width = max(len(val) for _, _, val in lines) + 2

    for label, value1, value2 in lines:
        print(f"{label:<{col1_width}}: {value1:<{col2_width}} | {value2:
↪<{col3_width}}")
```

We can also add a method that will confirm that a differential is impossible using the unsatisfiability approach:

[12]:
```
from claasp.cipher_modules.report import Report
from claasp.cipher_modules.models.utils import set_fixed_variables,␣
↪integer_to_bit_list
from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model␣
↪import SatXorDifferentialModel

def get_fix_variables_from_differential(cipher, impossible_trail):
    ciphertext_id = cipher.get_all_components()[-1].id
    fix_differential = []
    for input, input_size in cipher.inputs_size_to_dict().items():
        value = impossible_trail['components_values'][input]['value']
        indices_0 = [i for i, char in enumerate(value) if char == '0']
        fix_differential += [set_fixed_variables(component_id=input,␣
↪constraint_type='equal', bit_positions=indices_0, bit_values=[0] *␣
↪len(indices_0))]

        indices_1 = [i for i, char in enumerate(value) if char == '1']
        fix_differential += [set_fixed_variables(component_id=input,␣
↪constraint_type='equal', bit_positions=indices_1, bit_values=[1] *␣
↪len(indices_1))]

    value = impossible_trail['components_values']['inverse_' +␣
↪ciphertext_id]['value']
    indices_0 = [i for i, char in enumerate(value) if char == '0']
    fix_differential += [set_fixed_variables(component_id=ciphertext_id,␣
↪constraint_type='equal', bit_positions=indices_0, bit_values=[0] *␣
↪len(indices_0))]
```

```
    indices_1 = [i for i, char in enumerate(value) if char == '1']
    fix_differential += [set_fixed_variables(component_id=ciphertext_id,␣
 ↪constraint_type='equal', bit_positions=indices_1, bit_values=[1] *␣
 ↪len(indices_1))]
    return fix_differential

def verify_differential(cipher, impossible_trail):
    sat = SatXorDifferentialModel(cipher)
    differential = get_fix_variables_from_differential(cipher, impossible_trail)
    verification_trail = sat.
 ↪find_one_xor_differential_trail(fixed_values=differential,␣
 ↪solver_name='KISSAT_EXT')
    if verification_trail['status'] != 'UNSATISFIABLE':
        report = Report(verification_trail)
        report.show()
    else:
        ciphertext_id = cipher.get_all_components()[-1].id
        pt_value = impossible_trail['components_values']['plaintext']['value']
        ct_value = impossible_trail['components_values']['inverse_' +␣
 ↪ciphertext_id]['value']
        print(f"The differential({pt_value}, {ct_value}) is impossible.")
```

As before, we instantiate the model with the cipher object

[13]:
```
from claasp.cipher_modules.models.cp.mzn_models.
 ↪mzn_impossible_xor_differential_model import␣
 ↪MznImpossibleXorDifferentialModel
cp = MznImpossibleXorDifferentialModel(speck6)
```

The model is now ready to be used to find one impossible differential. It will automatically find the round that causes incompatibility, if any. By default, we fix the key difference to `0` and require nonzero differences in the plaintext and in the ciphertext.

If the `intermediate_components` flag is set to `True`, the model will not only check for an inconsistency only on the round outputs, but also on all components. This can be useful for permutations like Ascon or Chacha, which have the notion of half rounds. For instance, Ascon has an ID on 3.5 + 1.5 rounds.

[15]:
```
trail = cp.find_one_impossible_xor_differential_trail(solver_name='Chuffed',␣
 ↪intermediate_components=False)
trail_pretty_print(trail)
```

```
key         : …
|
plaintext   : …???…?1…
|
Round 0     : ???1…???1…
| ??????????????????????????????
```

```
Round 1     : ??????????1…??????????1…??
| ??????????????????????1..?????
Round 2     : ????????????????????????????????
| ??????1..???????…1…??1..
Round 3     : ????????????????????????????????
| …??1…1…
Round 4     : ????????????????????????????????
| …1…
Output      :
| 1…1…1.
```

Let us verify it with the SAT Xor differential model:

[16]:
```
verify_differential(speck6, trail)
```

```
The differential(00000000022200000021000000000000,
10000000000000001000000000000010) is impossible.
```

It is also possible to specify the round at which the contradiction occur. Here let's try 3:

[17]:
```
trail = cp.find_one_impossible_xor_differential_trail(middle_round=3,␣
 ↪solver_name='Chuffed', intermediate_components=False)
trail_pretty_print(trail)
```

```
No impossible differential found.
```

In the previous trail we found, the incompatibility was at the end of the second round, so setting `middle_round` to 2 should return an impossible differential:

[19]:
```
trail = cp.find_one_impossible_xor_differential_trail(middle_round=2,␣
 ↪solver_name='Chuffed', intermediate_components=False)
trail_pretty_print(trail)
```

```
key         : …
|
plaintext   : …?…1…
|
Round 0     : ??1…??1…
|
Round 1     : ??????????1…??????????1…??
| ??????????????????????1..?????
Round 2     :
| ??????1..???????…1…??1..
Round 3     :
| …??1…1…
Round 4     :
| …1…
Output      :
| 1…1…1.
```

All impossible differentials can be enumerated using the following command. Since there may be

many, we will partially fix the ciphertext difference to reduce the search space:

[20]:
```
fixed_variables = [set_fixed_variables('key', 'equal', range(64), [0]*64),
                   set_fixed_variables('plaintext', 'not_equal', range(32),␣
 ↪[0]*32),
                   set_fixed_variables('inverse_cipher_output_5_12', 'equal',␣
 ↪range(28), integer_to_bit_list(0x8000800, 28, 'big'))
                  ]
trails = cp.
 ↪find_all_impossible_xor_differential_trails(fixed_values=fixed_variables,␣
 ↪middle_round=2, solver_name='Chuffed', intermediate_components=False)
if trails:
    print(f"Found {len(trails)} impossible differentials. Printing the first␣
 ↪two")
    trail_pretty_print(trails[0])
    print()
    trail_pretty_print(trails[1])
```

```
Found 181 impossible differentials. Printing the first two
key         : …
|
plaintext   : …?…1…
|
Round 0     : ??1…??1…
|
Round 1     : ??????????1…??????????1…??
| ??????????????????????1..?????
Round 2     :
| ??????1..???????…1…??1..
Round 3     :
| …??1…1…
Round 4     :
| …1…
Output      :
| 1…1…1.

key         : …
|
plaintext   : …1…1…
|
Round 0     : ??1…??1…
|
Round 1     : ??????????1…??????????1…??
| ??????????????????????1..?????
Round 2     :
| ??????1..???????…1…??1..
Round 3     :
| …??1…1…
```

```
Round 4     :
| …1…
Output      :
| 1…1…1.
```

You may also want to search for clusters of impossible differentials, by maximizing the number of unknown bits in the input and output differences:

[21]:
```
speck4 = SpeckBlockCipher(number_of_rounds=4)
cp = MznImpossibleXorDifferentialModel(speck4)
trail = cp.find_one_impossible_xor_differential_cluster(fixed_values=[],␣
 ↪solver_name='Chuffed', intermediate_components=False)
trail_pretty_print(trail)
```

```
key         : …
|
plaintext   : …???1…?1…
|
Round 0     : ???…???…
| ????????????????????????????????
Round 1     : ??????????…??????????…??
| ????????????????????????????1?????
Round 2     : ????????????????????????????????
| ????????.???????????????1??????1
Output      :
| ??????1??????1?1??????.??????.??
```

We can also try on Present to see if the undisturbed bits bring any interesting results:

[22]:
```
from claasp.cipher_modules.models.cp.mzn_models.
 ↪mzn_impossible_xor_differential_model import␣
 ↪MznImpossibleXorDifferentialModel
from claasp.ciphers.block_ciphers.present_block_cipher import PresentBlockCipher
present =PresentBlockCipher(number_of_rounds=6)
cp = MznImpossibleXorDifferentialModel(present)
impossible_trail = cp.
 ↪find_one_impossible_xor_differential_trail(fixed_values=[],␣
 ↪solver_name='Chuffed', intermediate_components=False)
trail_pretty_print(impossible_trail)
```

```
key         :
…
|
plaintext   : 1..11..11..11..11..11..11..11..11..11..11..11..11..11..11..1
|
Round 0     : ????????????????????????????????????????????????…
| ????????????????????????????????????????????????????????????????
Round 1     : ???????????…???????????…???????????…???????????…
| ????????????????????????????????????????????????????????????????
```

```
Round 2    : ???.???.???.???.???.???.???.???.???.???.???.???.???.???.???.???.
| ????????????????????????????????????????????????????????????1
Round 3    : ????????????????????????????????????????????????????????????
| ???????????…???????????…???????????…???????????????1
Round 4    : ????????????????????????????????????????????????????????????
| ???.???.???.???.???.???.???.???.???.???.???.???.???.???.???.???1
Round 5    : ????????????????????????????????????????????????????????????
| …111111111111111…1111111111111111
Output     :
| …111111111111111…1111111111111111
```

We can use the SAT Xor Differential model to verify it:

[23]:
```
verify_differential(present, impossible_trail)
```

```
The
differential(1001100110011001100110011001100110011001100110011001100110011001,
0000000000000000111111111111111100000000000000001111111111111111) is impossible.
```

---

### 1.2.2 2. Word-Oriented Model

**Data Representation:**

The $m \times s$-bit internal state difference is expressed as

$$\Delta X = (\Delta X_0, \Delta X_1, \dots, \Delta X_{m-1}),$$

where each word $\Delta X_i$ is a vector in $\mathbb{F}_2^s$.

For each cell, two variables are introduced: $\delta X_i \in \{0, 1, 2, 3\}$ representing the differential pattern.

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_i = 0 (Z) \\ 1 & \text{fixed nonzero } (N) \\ 2 & \text{nonzero } (N^*) \\ 3 & \text{unknown } (U) \end{cases}$$

$\zeta X_i$ representing the actual value of the difference:

$$\zeta X_i \in \begin{cases} \{0\} & \text{if } \delta X_i = 0 \\ \{1, \dots, 2^s - 1\} & \text{if } \delta X_i = 1 \\ \{-1\} & \text{if } \delta X_i = 2 \\ \{-2\} & \text{if } \delta X_i = 3 \end{cases}$$

*Example (S-box application $Y = S(X)$)*

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

**Incompatibility detection**

At a round $r$, if the forward ($\delta X_i^F$) and backward ($\delta X^B{}_i$) patterns satisfy:

$$\delta X_i^F + \delta X_i^B < 3 \wedge \zeta X_i^F \neq \zeta X_i^B$$

---

The word-based model offers similar options to its bitwise counterpart. The best results on LBlock in single-key setting reach 14 rounds. The differential is $(00000000, 00\alpha00000) \rightarrow (0\beta000000, 00000000)$ and can be retrieved as follows:

[24]:
```
from claasp.ciphers.block_ciphers.lblock_block_cipher import LBlockBlockCipher
lblock = LBlockBlockCipher(number_of_rounds=14).remove_key_schedule()
from claasp.cipher_modules.models.utils import set_fixed_variables
from claasp.cipher_modules.models.milp.milp_models.
 ↪milp_wordwise_impossible_xor_differential_model import␣
 ↪MilpWordwiseImpossibleXorDifferentialModel
milp =  MilpWordwiseImpossibleXorDifferentialModel(lblock)
milp.init_model_in_sage_milp_class()
key_bits = []
for i in range(14):
    key_bits.append(set_fixed_variables(component_id=f'key_{i}_0',␣
 ↪constraint_type='equal', bit_positions=range(32), bit_values= [0]*32))
pt = set_fixed_variables(component_id='plaintext', constraint_type='equal',␣
 ↪bit_positions= range(16), bit_values= [0] * 10 + [2] + [0] * 5)
ct = set_fixed_variables(component_id='intermediate_output_13_12',␣
 ↪constraint_type='equal', bit_positions= range(16), bit_values= [0]*4 +[2] +␣
 ↪[0] * 11)
trail = milp.find_one_wordwise_impossible_xor_differential_trail(7,␣
 ↪fixed_bits=key_bits, fixed_words=[pt, ct], external_solver_name='SCIP_EXT')
```

```
Adding inequalities for truncated words of size 4 bits in pre-saved dictionary
Adding wordwise xor inequalities between 2 inputs of size 4 in pre-saved
dictionary
```

Note that in this example, we have to remove the key schedule of LBlock to analyze it under the word-based model because its key schedule contains a rotation by 29. It also simplifies the model.

[25]:
```
trail_pretty_print(trail, wordbased=True)
```

```
Plaintext                         : …*…
Round 0                           : *…
Round 1                           : ..*…*…
Round 2                           : …*..*…*…
Round 3                           : **…*…*..*.
Round 4                           : ***.**..**…*
Round 5                           : *.****?****.**..
Round 6                           : **??**??*.****?*
-----------------------------------
Round 6                           : **.**?**?*?*?*?*
Round 7                           : *.***.*.**.**?**
```

```
Round 8                         : …***..*.***.*.
Round 9                         : *.*…***..
Round 10                        : .*…*.*…
Round 11                        : …*..*…
Round 12                        : …*.
Round 13                        : …*…
Ciphertext                      : …*…
```

### 1.3 Hybrid Model for the Distinguisher Search

**Limitations of the Bit-Oriented Model:**

When a nonzero input difference produces no undisturbed bits, the model assigns an entirely unknown output difference, even if the operation is bijective. This loss of information can reduce the effectiveness of the model. The hybrid model extends the bit-wise model, tracking active but undetermined bit groups in outputs from nonlinear bijective operations.

**Idea:**

Introduce extra variables to represent the output bits of each non-linear operation and keep the information that at least one of them is nonzero.

**Data Representation:**

The hybrid model assigns an integer variable to each bit difference:

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_i = 0 \\ 1 & \text{if } \Delta X_i = 1 \\ 2 & \text{unknown} \\ id_{s,r'} & \text{if } \Delta X_{i,r} \text{ produced by S-box } s \text{ at round } r' \text{ with nonzero input} \end{cases}$$

*Example (S-box)*

For S-box $Y = S(X)$ with input difference $\Delta X$:

- Introduce bit-wise constraints variables $\beta X_i \in \{0, 1, 2\}$.
- Propagate output differences considering undisturbed bits:

$$\beta Y_i = \begin{cases} b_i & \text{if bit } i \text{ is undisturbed} \\ 2 & \text{otherwise} \end{cases}$$

Output patterns satisfy:

$$\delta Y = \begin{cases} \beta Y \text{ or } id_{s,r}, & \text{if some } \delta X_i = 1 \\ id_{s,r}, & \text{if all } \delta X_i = id_{s',r'} \\ 2, & \text{otherwise} \end{cases}$$

**Incompatibility detection**

Contradictions are detected as either:

- **Bit-based contradictions**: For some round $r$, $\delta X_{u,r,i} + \delta X_{l,r,i} = 1$.
- **Word-based contradictions**: For some round $r$, bits $i_0, ..., i_{n-1}$, and S-box index $s$:

$$\forall j \in \{i_0, ..., i_{n-1}\}, (\delta X_{u,r,j} = id_{s,r} \wedge \delta X_{l,r,j} = 0)$$

---

**Toy example** Consider an 8-bit Feistel cipher with state $(x_l||x_r)$, where the round function $F$ applies a key addition followed by an S-box:

$$F(x_l, k_i) = S_0(x_l \oplus k_i)$$

The S-box $S_0$ from LBlock is given by:

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

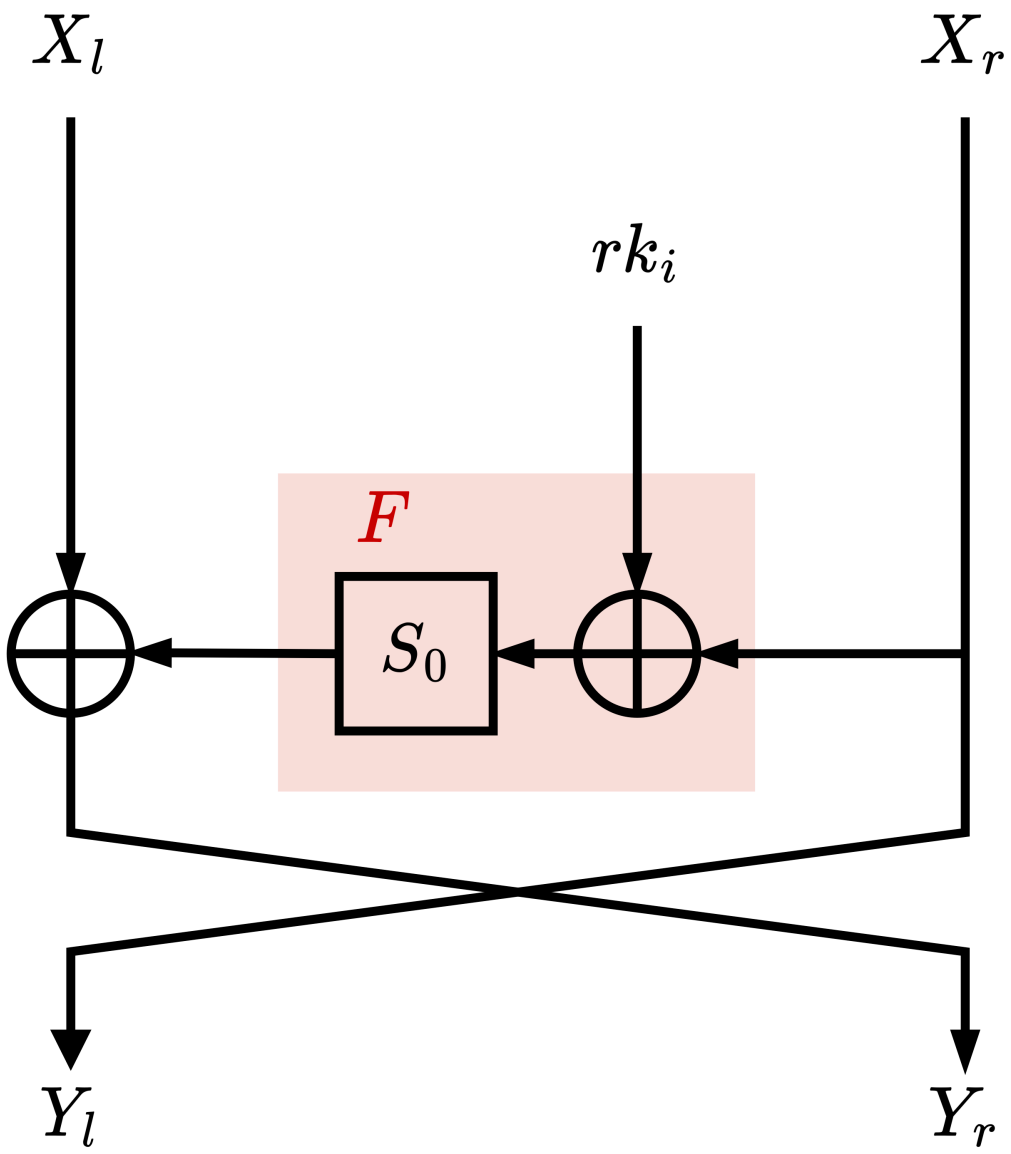
This S-box has the following undisturbed bits:

$$0000 \to 0000, \quad 0001 \to ???1, \quad 0010 \to ???1, \quad 0011 \to ??10, \quad 1000 \to ??1?, \quad 1011 \to ??0?$$

```
16  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  4  .  2  .  .  .  2  .  4  .  2  .  .  .  2
 .  4  .  2  .  .  .  2  .  4  .  2  .  .  .  2
 .  .  4  .  .  .  4  .  .  .  4  .  .  .  4  .
 .  .  .  2  4  2  4  .  .  .  .  2  .  2  .  .
 .  .  .  .  4  2  .  2  .  .  4  .  .  2  .  2
 .  .  4  .  4  2  .  2  .  .  .  .  .  2  .  2
 .  .  .  2  4  2  .  .  .  .  .  2  .  2  4  .
 .  .  2  2  .  .  2  2  .  .  2  2  .  .  2  2
 .  .  .  2  .  .  .  2  4  .  .  2  4  .  .  2
 .  4  2  .  .  .  2  .  .  .  2  .  .  4  2  .
 .  4  .  .  .  .  .  .  4  .  .  .  4  4  .  .
 .  .  2  2  .  2  2  .  2  2  .  .  2  .  .  2
 .  .  .  2  .  2  .  .  2  2  2  .  2  .  2  2
 .  .  2  .  .  2  2  2  2  2  .  2  2  .  .  .
 .  .  .  .  .  2  .  2  2  2  2  2  2  .  2  .
```

The key schedule is linear and updates the key $K = k_8 ... k_0$ and outputs the round key $rk_r$ as follows:

$$K = K \oplus (K \lll 5), \quad rk_r = k_8 k_7 k_6 k_5$$




Let us fix the key difference to `0x4` and the plaintext difference to `0` and compare the 3 models.

[26]:
```
from claasp.cipher_modules.models.utils import set_fixed_variables
fixed_variables = [set_fixed_variables('key', 'equal', range(8),␣
 ↪[0,0,0,0,0,1,0,0]), set_fixed_variables('plaintext', 'equal', range(8),␣
 ↪[0]*8)]
```

[27]:
```
from claasp.ciphers.toys.toyfeistel import ToyFeistel
toyfeistel = ToyFeistel(number_of_rounds=5)
from claasp.cipher_modules.models.cp.mzn_models.
 ↪mzn_impossible_xor_differential_model import␣
 ↪MznImpossibleXorDifferentialModel
mzn = MznImpossibleXorDifferentialModel(toyfeistel)
```

```
trail = mzn.
 ↪find_one_impossible_xor_differential_trail(fixed_values=fixed_variables,␣
 ↪solver_name='Chuffed', middle_round=3, intermediate_components=False)
trail_pretty_print(trail)
```

```
No impossible differential found.
```

[28]:
```
from claasp.ciphers.toys.toyfeistel import ToyFeistel
toyfeistel = ToyFeistel(number_of_rounds=5)
from claasp.cipher_modules.models.milp.milp_models.
 ↪milp_wordwise_impossible_xor_differential_model import␣
 ↪MilpWordwiseImpossibleXorDifferentialModel
milp =  MilpWordwiseImpossibleXorDifferentialModel(toyfeistel)
milp.init_model_in_sage_milp_class()
trail = milp.find_one_wordwise_impossible_xor_differential_trail(3,␣
 ↪fixed_bits=fixed_variables)
trail_pretty_print(trail)
```

```
GLPK: Problem has no feasible solution
No impossible differential found.
```

Due to the bitwise rotation of 5 bits in the key schedule, the word-based model is losing a lot of information. We can try to run it with manually

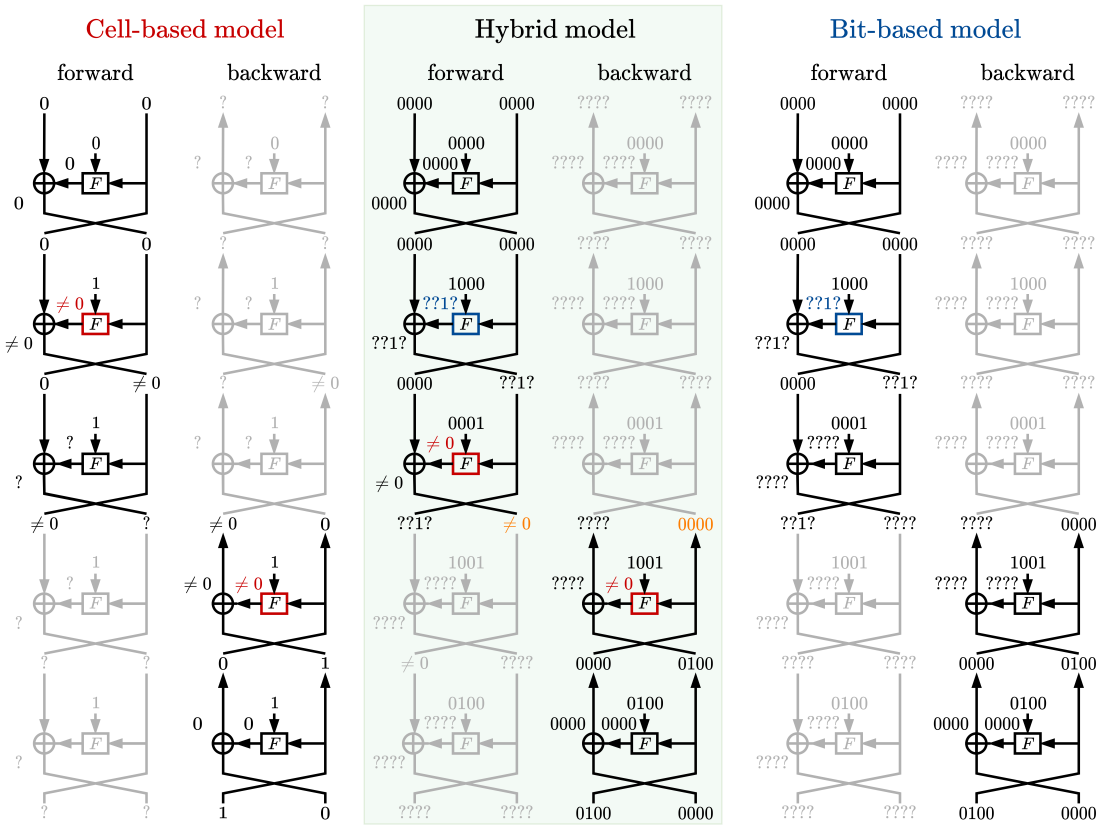
[29]:
```
from claasp.ciphers.toys.toyfeistel import ToyFeistel
toyfeistel = ToyFeistel(number_of_rounds=5).remove_key_schedule()
from claasp.cipher_modules.models.milp.milp_models.
 ↪milp_wordwise_impossible_xor_differential_model import␣
 ↪MilpWordwiseImpossibleXorDifferentialModel
milp =  MilpWordwiseImpossibleXorDifferentialModel(toyfeistel)
milp.init_model_in_sage_milp_class()
fixed_keys = [
    set_fixed_variables(component_id=f'key_0_0', constraint_type='equal',␣
 ↪bit_positions=range(4), bit_values= [0,0,0,0]),
    set_fixed_variables(component_id=f'key_1_0', constraint_type='equal',␣
 ↪bit_positions=range(4), bit_values= [1,0,0,0]),
    set_fixed_variables(component_id=f'key_2_0', constraint_type='equal',␣
 ↪bit_positions=range(4), bit_values= [0,0,0,1]),
    set_fixed_variables(component_id=f'key_3_0', constraint_type='equal',␣
 ↪bit_positions=range(4), bit_values= [1,0,0,1]),
    set_fixed_variables(component_id=f'key_4_0', constraint_type='equal',␣
 ↪bit_positions=range(4), bit_values= [0,1,0,0]),
    set_fixed_variables('plaintext', 'equal', range(8), [0]*8)
]
trail = milp.find_one_wordwise_impossible_xor_differential_trail(3,␣
 ↪fixed_bits=fixed_keys, external_solver_name='SCIP_EXT')
trail_pretty_print(trail)
```

```
No impossible differential found.
```

[30]:
```
import re
from claasp.ciphers.toys.toyfeistel import ToyFeistel
toyfeistel = ToyFeistel(number_of_rounds=5)
from claasp.cipher_modules.models.cp.mzn_models.
 ↪mzn_hybrid_impossible_xor_differential_model import␣
 ↪MznHybridImpossibleXorDifferentialModel
mzn = MznHybridImpossibleXorDifferentialModel(toyfeistel)
trail = mzn.
 ↪find_one_impossible_xor_differential_trail(number_of_rounds=5,fixed_values=fixed_variables,
 ↪solver_name='Chuffed', initial_round=1, middle_round=3,␣
 ↪final_round=5,intermediate_components=False)
trail_pretty_print(trail, hybrid=True)
```

```
Plaintext:          …
Round key 0:        …
Round output 0:     …
Round key 1:        1…
Round output 1:     …??1?
Round key 2:        …1
Round output 2:     ??1?4444
Round output 2:     ??.?..1.
Round output 3:     ..1..1..
Round output 4:     .1…1.
Round key 2:        …1
Round key 3:        1..1
Round key 4:        .1..
Cipher output:      ..1..1..
```

The hybrid model successfully detects an incompatibility while the others do not!


Cell-based model
Hybrid model
Bit-based model
forward
backward
forward
backward
forward
backward
F