TEA3

Sam Hadow

July 18, 2026

Contents

1 Git repositories **1**

2 Introduction **2**

3 Linear and differential bias in the keystream generation **2**
3.1 With TEA-3 original S-BOX . . . 2
3.2 With TEA-3 modified S-BOX . . . 3
3.2.1 at position [1, 4] . . . 3
3.2.2 at position [9, E] . . . 3
3.3 Comparing TEA-3 and AES S-BOX . . . 3
3.3.1 plugging the AES S-BOX instead of the TEA-3 S-BOX . . . 4
3.4 exchanging F31 and F32 . . . 4

4 F31 and F32 analysis **5**
4.1 linear and differential bias . . . 5
4.2 Walsh transform and non linearity . . . 5
4.3 Algebraic immunity . . . 6

5 Inspection with sagemath **6**
5.1 variable change . . . 7
5.1.1 first attempt . . . 7
5.1.2 second attempt . . . 7
5.1.3 third attempt . . . 8
5.1.4 fourth attempt . . . 8
5.2 XOR-ing register bits . . . 8
5.2.1 2 bits XOR . . . 8
5.2.2 3 and more bits XOR . . . 9
5.3 Cube attack (offline phase) . . . 9
5.3.1 Cube attack with the ANF . . . 9
5.3.2 Cube attack without the ANF . . . 9

6 Conclusion **10**

1 Git repositories

- `https://git.hadow.fr/sam.hadow/tea3-py`
  TEA 3 python and sagemath cryptanalysis tools.
- `https://git.hadow.fr/sam.hadow/tetra_crypto`
  TETRA cryptography implementations and tools in rust.
- `https://git.hadow.fr/sam.hadow/boolean-tools`
  Boolean tools (ANF solver and Walsh spectrum)

2 Introduction

The specifications of TETRA algorithms can be found in the ETSI document [Eur24].
As per [AAB+24] we can represent TEA-3 structure as follow:

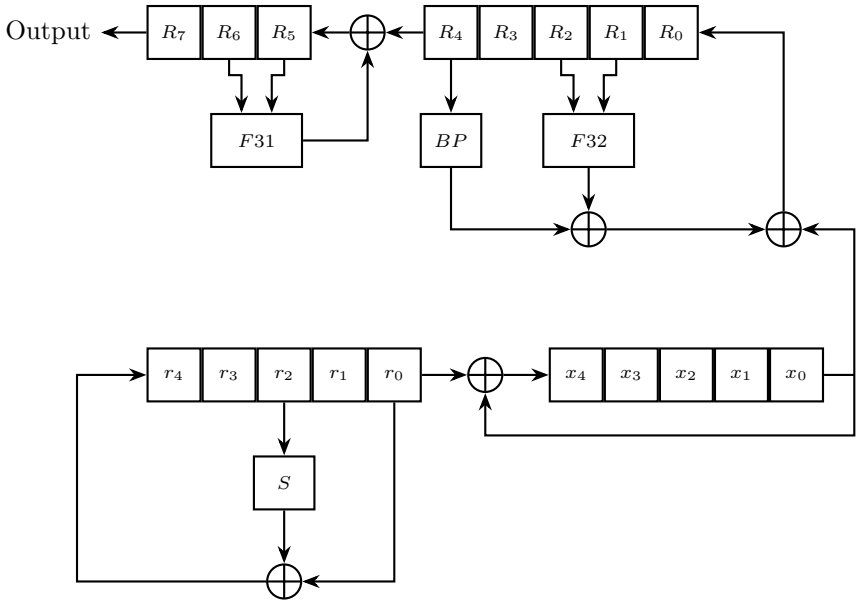

Figure 1: TEA-3 representation

BP is a bit permutation, S a S-BOX, and F31, F32 two non linear functions.
Indexing the bit from 0 to 7 BP outputs a byte 37612405, with 0 the least significant bit. The S-BOX is described in [Eur24] and the functions F31 and F32 in [AAB+24].
The S-BOX is also defined in the python sagemath code in 'constants.py'. The functions F31, F32 are also described later in section 4 or in the python sagemath code in 'tea3model.py'.

3 Linear and differential bias in the keystream generation

The linear and differential bias in the keystream generation are studied by modifying the IV each time and looking at the output byte. Modifying the IV makes sense since it can be controlled by an attacker.
For the differential bias, 2 IVs with a difference of 0x00000001 are generated.
As we can see there is no significant improvements when replacing the S-BOX with the AES S-BOX, or a modified TEA-3 S-BOX.
There is no significant difference either when exchanging F31 and F32.

Interpretation of the results:
`samples` is the number of random IVs tested.
`matches` is the number of times the expected event occurred (a 0 output bit for the linear test, or the expected output difference for the differential test).
`p_match` is the corresponding empirical probability.
`bias` is its deviation from the probability expected for a random cipher (1/2 for the linear test and 1/256 for the differential test).
`estimated_needed_samples` is an estimate of the number of samples required to reliably detect the observed bias. Since distinguishing attacks typically require a number of samples proportional to $1/\varepsilon^2$, where ε is the bias, smaller biases require significantly more observations to exploit.

3.1 With TEA-3 original S-BOX

```
$ tea3-linear-bias
black-box output bias
expected non-biased p_match: 1/2
BiasResult(samples=20000, matches=10065, p_match=0.50325, bias=0.003249999999999975,
estimated_needed_samples=94674.5562130192)
```

```
$ tea3-differential-bias
black-box differential bias
expected non-biased p_match: 1/256 = 0.00390625
DiffBiasResult(samples=20000, matches=76, p_match=0.0038, bias=0.00010625000000000001,
estimated_needed_samples=88581314.87889272)
```

3.2 With TEA-3 modified S-BOX

In the original TEA-3 S-BOX in the outputs, 0xC2 is duplicated (at positions [1, 4] and [9, E]) and 0xD2 is missing.
In the modified S-Box one OxC2 is replaced with 0xD2.
However as shown below, replacing this value doesn't have a great influence on the keystream bias.

3.2.1 at position [1, 4]

```
$ tea3-linear-bias
black-box output bias
expected non-biased p_match: 1/2
BiasResult(samples=20000, matches=10093, p_match=0.50465, bias=0.004650000000000043,
estimated_needed_samples=46248.121170076614)

$ tea3-differential-bias
black-box differential bias
expected non-biased p_match: 1/256 = 0.00390625
DiffBiasResult(samples=20000, matches=65, p_match=0.00325, bias=0.0006562500000000001,
estimated_needed_samples=2321995.4648526064)
```

3.2.2 at position [9, E]

```
$ tea3-linear-bias
black-box output bias
expected non-biased p_match: 1/2
BiasResult(samples=20000, matches=10065, p_match=0.50325, bias=0.003249999999999975,
estimated_needed_samples=94674.5562130192)

$ tea3-differential-bias
black-box differential bias
expected non-biased p_match: 1/256 = 0.00390625
DiffBiasResult(samples=20000, matches=76, p_match=0.0038, bias=0.00010625000000000001,
estimated_needed_samples=88581314.87889272)
```

3.3 Comparing TEA-3 and AES S-BOX

DDT (Difference Distribution Table): The DDT measures the differential properties of an S-box by counting how often each output difference occurs for a given input difference. It is used to evaluate the S-box's resistance to differential cryptanalysis. Lower maximum values indicate better resistance.

Differential uniformity: The largest non-trivial entry in the DDT. It represents the maximum number of input pairs that produce the same output difference for a fixed input difference. Lower values indicate stronger resistance to differential attacks.

Maximum difference probability: The highest probability that a given input difference leads to a particular output difference. It is obtained by dividing the maximum DDT coefficient by the total number of possible inputs (2^n for an n-bit S-box). Smaller probabilities correspond to better resistance against differential cryptanalysis.

LAT (Linear Approximation Table): The LAT measures the correlation between linear combinations of the input bits and linear combinations of the output bits. It is used to evaluate the S-box's resistance to linear cryptanalysis. Smaller absolute correlation values indicate better resistance.

Maximum LAT coefficient: The largest absolute entry in the LAT (excluding the trivial approximation). It measures the strongest linear correlation present in the S-box. Lower values indicate better resistance to linear cryptanalysis.

Relative bias: The maximum LAT coefficient normalized by the number of inputs (2^n). It represents the largest deviation of a linear approximation from a random Boolean function. A smaller relative bias indicates that linear approximations are less effective, improving resistance to linear cryptanalysis.

```
differential cryptanalysis resistance:

TEA3
differential uniformity: 10
max DDT coefficient: 10
max difference probability: 0.0390625

AES
differential uniformity: 4
max DDT coefficient: 4
max difference probability: 0.015625

linear cryptanalysis resistance:

TEA3
max LAT coefficient: 35
relative bias: 0.13671875

AES
max LAT coefficient: 16
relative bias: 0.0625
```

- Modifying the Byte [1, 4] from 0xC2 to 0xD2 results in max LAT coefficient changing to 36 (worse)
- Modifying the Byte [9, E] from 0xC2 to 0xD2 results in max LAT coefficient changing to 34 (better)

3.3.1 plugging the AES S-BOX instead of the TEA-3 S-BOX

```
$ tea3-linear-bias
black-box output bias
expected non-biased p_match: 1/2
BiasResult(samples=20000, matches=10039, p_match=0.50195, bias=0.0019500000000000073,
estimated_needed_samples=262984.8783694918)

$ tea3-differential-bias
black-box differential bias
expected non-biased p_match: 1/256 = 0.00390625
DiffBiasResult(samples=20000, matches=69, p_match=0.00345, bias=0.00045625000000000006,
estimated_needed_samples=4803903.1713267015)
```

3.4 exchanging F31 and F32

```
$ tea3-linear-bias
black-box output bias
expected non-biased p_match: 1/2
```

```
BiasResult(samples=20000, matches=10061, p_match=0.50305, bias=0.003049999999999997,
estimated_needed_samples=107497.98441279246)

$ tea3-differential-bias
black-box differential bias
expected non-biased p_match: 1/256 = 0.00390625
DiffBiasResult(samples=20000, matches=83, p_match=0.00415, bias=0.00024375000000000004,
estimated_needed_samples=16831032.215647593)
```

4 F31 and F32 analysis

F31 and F32 are 16 bits to 8 bits non-linear balanced functions in TEA-3.
F31 and F32 are defined as follow: (x_0 being the least significant bit)

$$F_{31}(x_0,\ldots,y_7)_0 = x_5x_6y_5 \oplus x_5x_6 \oplus x_5y_5y_6 \oplus x_5y_5 \oplus x_5y_6 \oplus x_6y_5y_6 \oplus y_5y_6 \oplus y_5 \oplus y_6$$
$$F_{31}(x_0,\ldots,y_7)_1 = x_6x_7y_6 \oplus x_6x_7 \oplus x_6y_6y_7 \oplus x_6y_7 \oplus x_6 \oplus x_7y_6 \oplus x_7y_7 \oplus y_6y_7 \oplus y_6 \oplus y_7 \oplus 1$$
$$F_{31}(x_0,\ldots,y_7)_2 = x_0x_7y_0 \oplus x_0y_0y_7 \oplus x_0y_0 \oplus x_0 \oplus x_7y_7 \oplus y_0$$
$$F_{31}(x_0,\ldots,y_7)_3 = x_0x_1y_0 \oplus x_0x_1y_1 \oplus x_0x_1 \oplus x_0y_0y_1 \oplus x_0y_1 \oplus x_1y_0y_1 \oplus x_1 \oplus y_1$$
$$F_{31}(x_0,\ldots,y_7)_4 = x_1x_2y_2 \oplus x_1x_2 \oplus x_1y_1 \oplus x_1y_2 \oplus x_2y_1y_2 \oplus x_2y_1 \oplus x_2 \oplus y_1y_2 \oplus y_1 \oplus y_2 \oplus 1$$
$$F_{31}(x_0,\ldots,y_7)_5 = x_2x_3y_3 \oplus x_2y_2 \oplus x_2 \oplus x_3y_2y_3 \oplus x_3y_3 \oplus x_3 \oplus y_2 \oplus y_3 \oplus 1$$
$$F_{31}(x_0,\ldots,y_7)_6 = x_3x_4y_4 \oplus x_3y_3 \oplus x_3y_4 \oplus x_4y_3y_4 \oplus x_4y_4 \oplus x_4 \oplus y_3y_4 \oplus 1$$
$$F_{31}(x_0,\ldots,y_7)_7 = x_4x_5y_5 \oplus x_4y_4y_5 \oplus x_4y_4 \oplus x_4y_5 \oplus x_5y_4y_5 \oplus x_5 \oplus y_5 \oplus 1$$

$$F_{32}(x_0,\ldots,y_7)_0 = x_5x_6y_5 \oplus x_5x_6 \oplus x_5y_5y_6 \oplus x_5y_5 \oplus x_5y_6 \oplus x_6y_5y_6 \oplus y_5 \oplus y_6 \oplus 1$$
$$F_{32}(x_0,\ldots,y_7)_1 = x_6x_7y_6 \oplus x_6x_7 \oplus x_6y_6y_7 \oplus x_6y_7 \oplus x_6 \oplus x_7y_7 \oplus x_7 \oplus y_6 \oplus 1$$
$$F_{32}(x_0,\ldots,y_7)_2 = x_0x_7y_0 \oplus x_0x_7y_7 \oplus x_0y_0y_7 \oplus x_0y_0 \oplus x_0 \oplus x_7y_0y_7 \oplus y_0$$
$$F_{32}(x_0,\ldots,y_7)_3 = x_0x_1y_0 \oplus x_0x_1y_1 \oplus x_0y_0y_1 \oplus x_1y_0y_1 \oplus x_1y_0 \oplus x_1 \oplus y_0y_1 \oplus y_1$$
$$F_{32}(x_0,\ldots,y_7)_4 = x_1x_2y_1 \oplus x_1x_2y_2 \oplus x_1x_2 \oplus x_1y_1y_2 \oplus x_1y_1 \oplus x_1 \oplus x_2y_1y_2 \oplus x_2 \oplus y_1y_2 \oplus y_2$$
$$F_{32}(x_0,\ldots,y_7)_5 = x_2x_3y_3 \oplus x_2y_2 \oplus x_3y_2y_3 \oplus x_3y_3 \oplus x_3 \oplus y_3$$
$$F_{32}(x_0,\ldots,y_7)_6 = x_3x_4y_4 \oplus x_3x_4 \oplus x_3y_3 \oplus x_3y_4 \oplus x_4y_3y_4 \oplus x_4y_3 \oplus x_4 \oplus y_3y_4 \oplus y_3 \oplus y_4$$
$$F_{32}(x_0,\ldots,y_7)_7 = x_4x_5y_5 \oplus x_4y_4y_5 \oplus x_4y_4 \oplus x_5y_4y_5 \oplus x_5 \oplus y_4 \oplus y_5 \oplus 1$$

4.1 linear and differential bias

```
F31
differential uniformity: 40960
max DDT coefficient: 40960
max difference probability: 160.0
max LAT coefficient: 16384
relative bias: 0.25

F32
differential uniformity: 40960
max DDT coefficient: 40960
max difference probability: 160.0
max LAT coefficient: 16384
relative bias: 0.25
```

F31 and F32 have the same coefficients. And despite being balanced functions, they are linearly and differentially very weak. F31 and F32 are not the same polynomial map, but appear to be variants of the same core construction.

4.2 Walsh transform and non linearity

The **Walsh transform** measures the correlation between a Boolean function and all possible linear Boolean functions. It is a standard tool for evaluating the resistance of a function to linear

cryptanalysis. Small Walsh coefficients indicate that the function behaves more like a random Boolean function and has weaker linear approximations.
The **Walsh spectrum** is the collection of all Walsh transform coefficients for a Boolean function. The largest absolute value in the spectrum determines the function's non-linearity: the smaller this maximum coefficient, the higher the non-linearity and the better the resistance to linear attacks.
We can therefore calculate the non linearity of the coordinates of the functions F31 and F31 using a Walsh transform. They all have a non linearity of 4.
The maximum non linearity would be 6 for a bent function, so 4 is good but not optimal. However a bent function would not be balanced unlike F31 and F32.

The Walsh spectra of the coordinates of F31 are:

```
[0, -4, 4, -8, 0, 4, 4, 0, 4, 0, 0, 4, 4, 8, 0, -4]
[0, 0, 4, 4, 0, -8, 4, -4, -4, -4, 0, 0, -4, 4, 0, -8]
[0, 0, -4, 4, 0, 8, 4, 4, -4, 4, 0, 0, 4, 4, 0, -8]
[0, -4, 4, 0, 4, 0, 0, -4, 4, 0, 8, 4, 0, -4, -4, 8]
[0, 4, 0, 4, -4, 0, -4, 0, 0, 4, -8, -4, -4, 0, 4, -8]
[0, 4, 0, -4, 4, 0, -4, 0, 0, -4, 8, -4, -4, 0, -4, -8]
[0, -4, 0, -4, -4, 0, -4, 0, 0, 4, -8, -4, 4, 0, -4, 8]
[0, 4, -4, 0, -4, 0, 0, 4, 0, -4, -4, -8, 4, 0, -8, 4]
```

The Walsh spectra of the coordinates of F32 are:

```
[0, 4, -4, 0, 0, -4, -4, 8, -4, 0, 0, 4, -4, -8, 0, -4]
[0, 0, -4, 4, 0, -8, -4, -4, 4, -4, 0, 0, 4, 4, 0, -8]
[0, 4, -4, 0, 4, 8, 0, 4, -4, 0, 0, 4, 0, 4, 4, -8]
[0, 4, 4, 0, -4, 0, 0, -4, 4, 0, 8, -4, 0, -4, 4, 8]
[0, 4, 4, 0, -4, 0, 0, -4, -4, 0, 8, 4, 0, 4, -4, 8]
[0, -4, 0, 4, -4, 0, 4, 0, 0, 4, 8, 4, 4, 0, 4, -8]
[0, -4, 0, -4, 4, 0, 4, 0, 0, -4, 8, 4, 4, 0, -4, 8]
[0, -4, -4, 8, 4, 0, 0, -4, 0, 4, -4, 0, -4, 0, -8, -4]
```

4.3 Algebraic immunity

Algebraic immunity is the minimum degree of a non-zero Boolean function that annihilates a given Boolean function (or its complement). It measures the resistance of a Boolean function to algebraic attacks: a higher algebraic immunity means that low-degree algebraic relations are more difficult to find.
All the coordinates of both F31 and F32 have an algebraic immunity of 2, meaning a quadratic annihilator exists. It is the maximum possible since each coordinates has a 4-variables input
But for a Boolean function on 16 variables, the theoretical maximum algebraic immunity is 8.

5 Inspection with sagemath

In the first git repository, there is a TEA-3 model to study polynomials evolution rounds after rounds. We cannot conclude much from the naive search. And we can only compute up to 9 rounds before the number of monomials gets too big to compute more rounds.
We're mostly interested in the polynomials representation in R_0 and R_5 where the feedback happens.

The naive search, named "classic inspection" in the python sagemath code allows us to compute n rounds. We can then see polynomials indicating how the bits in the R registers depends on the initial values of all the registers.
The advanced search, named "Advanced inspection with forced 0 or 1 bits" in the python sagemath code allows us to compute n rounds forcing chosen initial bits to 0 or 1.
The initial values of the registers are named R_{ij}, x_{ij} and r_{ij}, with i being the index of the register and j being the bit in this register. The bits are indexed from 0 to 7. 0 being the least significant bit.
After n rounds, each bit of the R registers is a polynomial depending on the R_{ij}, x_{ij} and r_{ij}. The R_{ij} and r_{ij} are abstracted by a variable g and a variable h respectively.

As said earlier, after 9 rounds, even when abstracting the values depending on a function f(Ri) and

a function f'(ri) with a variable named g and a variable named h respectively, there are too many monomials to compute further rounds.
We therefore look at the polynomial in R_5 after 8 rounds because that's when the feedback from F_{31} happens. And we look at the polynomial in R_0 after 6 rounds because that's when the feedback from BP happens.

5.1 variable change

In this section we're trying variable changes in an attempt to simplify the polynomials representation in the registers. We considered 4 matrices but no interesting variable change was found.

5.1.1 first attempt

In the python sagemath code, this corresponds to the menu entry "exhaustive variable change search".
We first consider the following variable changes inside each 8-bit register:

$$\forall R \in \{0,\dots,7\}, \quad \begin{cases} y_{R,1} = x_{R,1} \\ y_{R,2} = x_{R,2} \\ y_{R,3} = x_{R,3} \\ y_{R,4} = x_{R,4} \\ y_{R,5} = x_{R,5} \\ y_{R,6} = x_{R,6} \\ y_{R,7} = x_{R,7} \\ y_{R,0} = x_{R,0} \oplus a_1 x_{R,1} \oplus a_2 x_{R,2} \oplus a_3 x_{R,3} \oplus a_4 x_{R,4} \oplus a_5 x_{R,5} \oplus a_6 x_{R,6} \oplus a_7 x_{R,7} \end{cases} \qquad a_i \in \mathbb{F}_2$$

Equivalently, for each register $R \in \{0,\dots,7\}$, we have

$$\begin{pmatrix} y_{R,0} \\ y_{R,1} \\ y_{R,2} \\ y_{R,3} \\ y_{R,4} \\ y_{R,5} \\ y_{R,6} \\ y_{R,7} \end{pmatrix} = \begin{pmatrix} 1 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{R,0} \\ x_{R,1} \\ x_{R,2} \\ x_{R,3} \\ x_{R,4} \\ x_{R,5} \\ x_{R,6} \\ x_{R,7} \end{pmatrix} \qquad a_i \in \mathbb{F}_2$$

Unfortunately, there does not seem to be a simplification in the bits of the registers R_0 or R_5 with these variable changes.
The best one is $(a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (0, 0, 0, 0, 0, 0, 0)$ which corresponds to no variable change.
This corresponds to 2365 monomials in the bits in R_0 after 6 rounds. And 386 total monomials in the bits in the register R_5 after 8 rounds.

5.1.2 second attempt

for each register $R \in \{0,\dots,7\}$, we now consider the following variable changes (identity + part of a cyclic upper bidiagonal matrix):

$$\begin{pmatrix} y_{R,0} \\ y_{R,1} \\ y_{R,2} \\ y_{R,3} \\ y_{R,4} \\ y_{R,5} \\ y_{R,6} \\ y_{R,7} \end{pmatrix} = \begin{pmatrix} 1 & a_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & a_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & a_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & a_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & a_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & a_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & a_7 \\ a_8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{R,0} \\ x_{R,1} \\ x_{R,2} \\ x_{R,3} \\ x_{R,4} \\ x_{R,5} \\ x_{R,6} \\ x_{R,7} \end{pmatrix} \qquad a_i \in \mathbb{F}_2$$

Unfortunately, there does not seem to be a simplification in the bits of the registers R_0 or R_5 with these variable changes either.
The best one is $(a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8) = (0, 0, 0, 0, 0, 0, 0, 0)$ which corresponds to no variable change.

5.1.3 third attempt

For each register $R \in \{0, \ldots, 7\}$, we now consider the following variable changes (identity + part of a cyclic lower bidiagonal matrix):

$$\begin{pmatrix} y_{R,0} \\ y_{R,1} \\ y_{R,2} \\ y_{R,3} \\ y_{R,4} \\ y_{R,5} \\ y_{R,6} \\ y_{R,7} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & a_1 \\ a_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_6 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_7 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_8 & 1 \end{pmatrix} \begin{pmatrix} x_{R,0} \\ x_{R,1} \\ x_{R,2} \\ x_{R,3} \\ x_{R,4} \\ x_{R,5} \\ x_{R,6} \\ x_{R,7} \end{pmatrix} \qquad a_i \in \mathbb{F}_2$$

We tried this matrix because F31 and F32 are cyclic.
Unfortunately, like with the 2 previous attempts, there does not seem to be a simplification in the bits of the registers R_0 or R_5 with these variable changes either.

5.1.4 fourth attempt

For each register $R \in \{0, \ldots, 7\}$, we now try the following variable changes which corresponds to the identity matrix + a part of the permutation BP:

$$\begin{pmatrix} y_{R,0} \\ y_{R,1} \\ y_{R,2} \\ y_{R,3} \\ y_{R,4} \\ y_{R,5} \\ y_{R,6} \\ y_{R,7} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & a_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & a_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & a_3 & 0 \\ 0 & a_4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_6 & 1 & 0 & 0 \\ a_7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_8 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{R,0} \\ x_{R,1} \\ x_{R,2} \\ x_{R,3} \\ x_{R,4} \\ x_{R,5} \\ x_{R,6} \\ x_{R,7} \end{pmatrix} \qquad a_i \in \mathbb{F}_2$$

We also tried this matrix because F31 and F32 are cyclic.
But there does not seem to be a simplification in the bits of the registers R_0 or R_5 with these variable changes either.

5.2 XOR-ing register bits

The python sagemath code includes a "variable XOR" menu entry to XOR bits in a register.
The code also includes an "Exhaustive XOR search" entry in the menu to find the n-bits XOR resulting in the least number of monomials.
Finding a XOR simplifying the polynomials in the R-registers is interesting to potentially find weak IVs.

5.2.1 2 bits XOR

Looking at R_0 and R_5, there doesn't seem to be a 2 bits XOR combination giving a more readable output after many rounds without fixing bits to 0.
We recall that without the XORs, we have 2365 monomials in the bits in R_0 after 6 rounds. And 386 total monomials in the bits in the register R_5 after 8 rounds.
And for reference R_{51} has 33 monomials, R_{52} 36 monomials after 8 rounds. R_{00} 269 monomials and R_{01} 237 monomials after 6 rounds.

- For R_0 the best XOR is with the bits 0 and 1 resulting in 458 monomials after 6 rounds.

- For R_5 the best XOR is with the bits 1 and 2 resulting in 49 monomials after 8 rounds.
  We have the following polynomial:

```
[Step 8]
XOR of R_bits[5][[1, 2]] =
1 + x00*x04*x05*x10 + x00*x04*x05 + x00*x04*x10 + x00*x04 + x00*x05*x10 + x00*x05
+ x00*x07*x10 + x00*x07 + x00*x10*x17 + x00*x17 + x03*x04*x06*x07 + x03*x04*x06*x17
+ x03*x04 + x04*x05*x06*x16 + x04*x06*x16 + x05*x06*x16 + x06*x07*x16
+ x06*x16*x17 + x06*x16 + x07 + x10 + x16 + x21 + x22 + x00*x04*x05*x06*g
+ x00*x04*x05*g + x00*x04*x06*g + x00*x04*g + x00*x05*x06*x07*g + x00*x05*x06*x17*g
+ x00*x05*x06*g + x00*x05*g + x00*x06*x07*g + x00*x06*x17*g + x00*x06*g + x00*x07*g
+ x00*x10*g + x00*x17*g + x03*x04*x05*x06*g + x03*x05*x06*g + x03*x06*x07*g
+ x03*x06*x17*g + x03*g + x04*x05*x06*g + x04*x06*g + x06*x16*g + x06*x17*g + g
```

So if x00 = 0 we have 25 monomials. If x06 = 0 we also have 25 monomials. If both x00 and x06 = 0 we have 9 monomials:

```
P = x03*x04 + x03*g + x07 + x10 + x16 + x21 + x22 + g + 1
```

For reference, if we force x00 and x06 to 0, R_{51} has 15 monomials, and R_{52} 10 monomials after 8 rounds.

5.2.2 3 and more bits XOR

When XOR-ing more bits, the number of monomials keeps increasing.
With a 3 bits XOR for example:

- For R_0 the best XOR is with the bits 0, 1 and 5 resulting in 703 monomials after 6 rounds.
- For R_5 the best XOR is with the bits 1, 2 and 4 resulting in 85 monomials after 8 rounds.

And with a 4 bits XOR for example:

- For R_0 the best XOR is with the bits 0, 1, 5 and 6 resulting in 927 monomials after 6 rounds.
- For R_5 the best XOR is with the bits 1, 2, 3 and 4 resulting in 118 monomials after 8 rounds.

5.3 Cube attack (offline phase)

[DS08]
The offline phase consists in searching, over many candidate cubes of public IV bits, for output coordinates whose summed value over the cube has unusually low algebraic degree.
We evaluate the cipher for all assignments of the cube variables, XOR the resulting output bit, and then check whether the obtained superpoly is constant, linear, or simple enough to be useful in the online phase. Cubes leading to low-degree superpolys are kept as attack candidates.

5.3.1 Cube attack with the ANF

A menu entry "Cube attack search (symbolic)" exists in the sagemath CLI to search for cubes usable in a cube attack (offline phase). It searches for small sets of public R variables (IV variables) whose XOR makes a target output bit collapse into a low-degree polynomial, ideally something linear.
It however quickly (when computing more than 4 rounds) becomes slow to run as we do not abstract R variables (the IV variables) in this part, otherwise we would only find degree 0 superpolys. So no promising cube and superpoly have been found by lack of computing power.

5.3.2 Cube attack without the ANF

A menu entry "Cube attack search (oracle)" exists in the sagemath CLI. It allows the user to choose an IV and a key and perform the offline phase of the cube attack treating TEA-3 as an oracle and not computing the ANF.
In this black-box version of the offline phase, candidate cubes are formed by selecting a small set of public IV bits and evaluating the cipher on all 2^k assignments of these bits, while keeping the other IV bits fixed. The keystream output bit is then XOR-ed over all assignments to obtain the cube

sum. Cubes sum which are non-zero are kept as candidates, since they may reveal a low-degree superpoly useful in the online phase.
This approach is empirical, it searches for useful cubes directly on the cipher, but it does not recover the exact polynomial degree.

6 Conclusion

TEA-3 was explored from various angles: keystream bias measurements, analysis of the nonlinear functions F_{31} and F_{32}, polynomial inspection with SageMath, variable changes, XOR combinations, and cube attack searches. However, the results do not reveal an obvious weakness that would immediately lead to a practical attack.
The experiments on linear and differential bias show only small deviations from the expected random behaviour, and replacing the original S-box or swapping F_{31} and F_{32} does not significantly change anything. The analysis of F_{31} and F_{32} also shows that, although these functions are balanced and have moderate nonlinearity, they remain weak to direct differential and linear cryptanalysis.
Several attempts were made to simplify the polynomial representation with variable changes or by XORing register bits, but none of them produced a meaningful reduction in complexity.

References

[AAB+24] Jens Alich, Amund Askeland, Subhadeep Banik, Tim Beyne, Anne Canteaut, Patrick Felke, Gregor Leander, Willi Meier, and Lukas Stennes. Observations on TETRA encryption algorithm TEA-3. Cryptology ePrint Archive, Paper 2024/2045, 2024.

[DS08] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. Cryptology ePrint Archive, Paper 2008/385, 2008.

[Eur24] European Telecommunications Standards Institute. TETRA Air Interface Security, Algorithms Specifications; Part 1: TETRA Encryption Algorithms Set A. Technical Specification TS 104 053-1 V1.1.1, ETSI, 2024.