Cryptanalysis of TETRA Encryption Algorithms

Episode 1: TEA-3

Jens Alich[1], Amund Askeland[2], Subhadeep Banik[3], Tim Beyne[4], Anne Canteaut[5], Patrick Felke[6], Gregor Leander[1], Willi Meier[7] and Lukas Stennes[1]

[1] Ruhr University Bochum, Bochum, Germany
firstname.lastname@rub.de
[2] University of Bergen, Bergen, Norway
amund.askeland@uib.no
[3] Universita della Svizzera Italiana, Lugano, Switzerland
subhadeep.banik@usi.ch
[4] COSIC, KU Leuven, Leuven, Belgium
tim.beyne@esat.kuleuven.be
[5] Inria, Paris, France
anne.canteaut@inria.fr
[6] University of Applied Sciences Emden/Leer, Emden, Germany
patrick.felke@hs-emden-leer.de
[7] University of Applied Sciences and Arts Northwestern Switzerland, Windisch, Switzerland
willimeier48@gmail.com

Abstract. We present the first public and in-depth cryptanalysis of TEA-3, a stream cipher used in TETRA radio networks that was kept secret until recently. While the same also holds for the six other TETRA encryption algorithms, we pick TEA-3 to start with as (i) it is *not* obviously weakened as TEA-{1,4,7} but (ii) in contrast to TEA-2 it is approved only for *extra-European* emergency service, and (iii) as already noted by [MBW23] the TEA-3 design surprisingly contains a non-bijective S-box. Most importantly, we show that the 80-bit non-linear feedback shift register operating on the key decomposes into a cascade of two 40-bit registers. Although this hints at an intentional weakness at first glance, we are not able to lift our results to a practical attack. Other than that, we show how the *balanced* non-linear feedback functions used in the state register of TEA-3 can be constructed.

Keywords: TETRA · TEA-3 · Stream cipher · Cryptanalysis · ETSI

1 Introduction

TETRA (Terrestrial Trunked Radio) is a standardized trunked radio system that automates channel selection, enabling multiple users to communicate efficiently—a critical feature for many applications. TETRA is the European counterpart to North America's Project 25 and was specifically developed for government agencies, emergency services (such as police, fire departments, and ambulance services), public safety networks, rail transport staff, transport services, and the military. Standardized by the European Telecommunications Standards Institute (ETSI) in 1995, TETRA is now used in over 100 countries and is the most widely adopted police radio communication system outside of the U.S. While Project 25 supports widely recognized encryption algorithms such as DES, Triple-DES, and AES, TETRA specifies several proprietary cryptographic algorithms as its core security components. These proprietary algorithms were kept secret for an extended period, a practice that can compromise national security and public safety, as it prevents potential design flaws

from being found, reported, and fixed. This approach is particularly concerning given ETSI's history of standardizing proprietary cryptographic primitives with weaknesses that may enable practical attacks. Examples in other ETSI standards, such as A5/1 [BSW00] and GEA [BDL+21], have raised concerns and mistrust regarding TETRA's security foundations. Only after the reverse engineering effort by the Dutch security researchers Carlo Meijer, Wouter Bokslag, and Jos Wetzels in 2022 [MBW23], ETSI decided to publish the details of the encryption algorithms.

Originally, i.e. in the mid 1990s, four algorithms TEA-1 to TEA-4, were designed for TETRA [ETS24j]. They are all stream ciphers with an 80-bit key and 29-bit IV.

- TEA-1: Has an internal state of 12 bytes, 8 in the state and 4 in the key register. It is primarily intended for commercial use. It is widely used by critical infrastructure sectors worldwide, such as pipelines, railways, and the electric grid. Its key size is reduced to 32 bits which we discuss in more detail below.

- TEA-2: Has an internal state of 18 bytes, split into one non-linear feedback register for the key that feeds into another non-linear feedback register producing the key stream. Designed for exclusive use within Europe, it is deployed in radios and walkie-talkies by police, military, intelligence agencies, and emergency personnel.

- TEA-3: Has an internal state of 18 bytes, split into one non-linear feedback register for the key that feeds into another non-linear feedback register producing the key-stream. This is the export version of TEA-2, serving the same user groups but intended for use outside of Europe.

- TEA-4: Has an internal state of 15 bytes, again split into a state and key register of 8 and 7 bytes respectively. TEA-4 is intended for commercial use outside Europe, but reportedly has seen minimal practical use [MBW23]. Similar to TEA-1, the effective key size is reduced, in this case to 56 bits.

In 2022, ETSI introduced the TETRA encryption Set B containing three more stream ciphers TEA-{5,6,7}. They were made public in 2024 [ETS24h]. All of them take a 192-bit key and an 80-bit IV as input and all of them are built in the same way: they expand the IV to 192 bits and then apply an 8-bit S-box 48 times in parallel, each time taking 4 bits from the key and 4 bits from the IV. Thereby, a different key and IV are derived which are then used to run Rijndael in counter mode. Moreover,

- TEA-5: Is the analog to TEA-2.

- TEA-6: Is the analog to TEA-3 and, interestingly, again uses a non-bijective S-box (which reduces the complexity of a brute force attack by 9 bits).

- TEA-7: Has an effective key size of 56 bits [ETS24k] and is the counterpart to TEA-{1,4}. The reduction of the key size is achieved by defining the aforementioned S-box [ETS24h, Table 6] such that, for a given IV, the *collision entropy* of the key and hence the security against a brute force attack is only 56 bits.

For the full details of which cipher is allowed to use in which context, we refer to the official ETSI rules in [ETS24a, ETS24b, ETS24c, ETS24d, ETS24e, ETS24f, ETS24g].

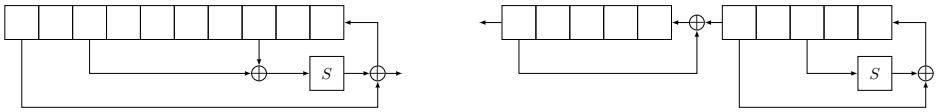
As mentioned above, in 2022 Dutch security researchers managed to extract TEA-{1,2,3} from a TETRA radio device and reverse-engineered them, revealing several critical vulnerabilities [MBW23]. The most significant cryptographic weakness discovered was in TEA-1, which initializes its internal state by essentially clocking the 80-bit key into a 32-bit register. This approach allows the algorithm to be practically broken within seconds on a standard modern laptop. As a result, TEA-1 appears intentionally weakened.

Whether this, or the parallels in TEA-{4,7}, constitutes a "backdoor" is debatable. On one hand, the use of an 80-bit key that is then effectively reduced internally to 32 bits might suggest a backdoor, as the design obfuscates the true key length. On the other hand, the weakness is apparent once the algorithm's description is known. ETSI has claimed this design choice was made to comply with export regulations at the time [ETS24k, Zet24, WAS24]. However, a straightforward reduction to a 32-bit key length would have achieved compliance without obfuscation, raising further concerns over the design's transparency and security.

It is important to note that, as with mobile networks, end-to-end encryption should always be added on top of TETRA whenever possible to enhance security. Despite TETRA's widespread and critical applications, ETSI only released the specification of TETRA to the public two years after [MBW23] reverse-engineered the algorithms. This release, along with the original research, remains one of the only sources of information on the TEA algorithms, with the cryptanalytic security of TEA entirely unexplored in published literature. This lack of analysis is concerning given (i) the extensive global use of these algorithms, (ii) their deployment in critical applications, and (iii) ETSI's questionable track record in designing robust ciphers. In response to this situation, a thorough investigation into the security of TEA is both necessary and urgent.

Our Contribution. We conduct an in-depth analysis of the TEA-3 algorithm, which appears to be an especially pertinent target for reasons discussed before. While we do not succeed in breaking the cipher, we identify several unconventional and unnecessarily weak design choices that warrant further scrutiny.

As mentioned above, TEA-3 consists of two nonlinear feedback registers. While the benefit of using non-linear shift registers is obvious, they provide the necessary non-linearity for a secure cipher, their behavior is less understood. This is in particular true for word-based non-linear shift registers as used here. Thus, as our first contribution, we transfer and generalize the theory known for the binary NFSRs to the word-based case. This theory about cascades of NFSRs and their cycle structure becomes of particular interest as the key-register decomposes into two registers, an NFSR with a 5-byte state feeding into a linear LFSR with a 5-byte state as well.



This decomposition is highly unexpected and leads to very short periods for this register of roughly 2^{40} instead of the 2^{80} that would have been possible. The theory adopted and generalized in Section 3 is key to explain this behavior and is given in Corollary 3. This reduces the computation of cycles to the computation of the possible cycles of the small non-linear register and we are able to compute the entire cycle structure of that.

While it is not clear if this property can be lifted to an attack, it certainly does not increase the trust in the cipher. Indeed, based on the short and known cycle length, it would be possible to mount a key recovery attack for around 75% of the key space (in time less than generic key/state recovery attack) as explained in Section 6. However, this attack would require the IV to be larger or equal to 32 bits, while the actual IV is only 29 bits.

The other, much more obvious observation that was already discussed in [MBW23] is that the function S is actually not a permutation, even if it is referred to as such in the standard. Note that the update function of the key register is still invertible. Thus there is no immediate entropy loss arising from the non-bijectivity of the S function. Given our understanding of the cycle structure we performed an experiment, reported in Section 4.2 to understand if the choice of the S function significantly influences the cycle structure,

but that does not seem to be the case. So, again, we could not lift this property to an attack, but it is a very unexpected property as well.

The other component where the design is unclear given only the specification are the non-linear functions F_{31} and F_{32}, mapping 16 bits to 8 bits in a balanced manner. Those functions consist of 8 Boolean functions that take 4 bits as input to produce the 8 output bits. It is not trivial to construct such a function in a way that ensures that the overall function is balanced and our main contribution here is to give a general construction that can be used to create those functions.

Besides generic attacks, that are possible due to the way the IV and the key are handled and that we present in the last part of Section 6, we discuss the resistance of TEA-3 against linear cryptanalysis. It is intriguing to observe that the best linear trails between consecutive keystream bytes have correlation $\pm 2^{-32}$. Given the 64 bit state, this fact can be interpreted as a motivation for using a decimation factor of 19. A rough analysis of the resources required for a key-recovery attack based on this observation suggests a time-complexity of slightly less than 2^{80} memory accesses and arithmetic operations, for a data-complexity of 2^{67} keystream bytes. Using more modern techniques of linear cryptanalysis, such as using additional linear approximations obtained from multiple linear trails with lower correlation and more efficient methods for key recovery, it might be possible to further reduce the time- and data-complexity. However, we do not expect a linear attack that outperforms a brute force attack in its overall practical costs.

Related Work. TETRA is not the first set of ETSI-standardized ciphers to exhibit critical security issues. Similar concerns have previously arisen with the A5/1 cipher, used for voice encryption, and the GEA-1 cipher, used for data encryption in mobile networks. Both ciphers share a comparable history: they were proprietary, kept secret, and later revealed to be weak—arguably by design—once their specifications became public. In addition to the specifications of the TETRA algorithms [ETS24j, ETS24h] and reverse engineering efforts such as [MBW23], a recent interview [Zet24] with Brian Murgatroyd, who manages the technical body responsible for TETRA and critical broadband technology standardization, provides valuable insights into the process.

For foundational theory on NFSRs, the primary references are Golomb's seminal work [Gol81] and additional sources as [MST79, GD70], which develop the theory required to analyze the cycle structure of cascaded bit-based constructions.

Outline. We start by describing TEA-3 in Section 2. In Sections 3 and 4 we present necessary theory and the details of the decomposition of the key register respectively. Then, in Section 5, we study the non-linear functions F_{31} and F_{32} which are used in the feedback of the state register. We give some more general cryptanalytic results in Section 6 and conclude the paper in Section 7.

2 Description of TEA-3

In this section, we describe TEA-3 based on the reverse-engineered source code provided by Midnight Blue [MBW23][1], the original specification by ETSI [ETS24j] and private communication [Bok]. In terms of notation, we stick with [MBW23] and hence deviate from the ETSI document which, in contrast to our work, has the least significant bit/byte on the right and starts counting from 1 from the left. TEA-3 is a stream cipher that takes as input an 80-bit key k, a 29-bit initialization vector $\texttt{IV}$ and a 32-bit tweak t. We denote the keystream of m bytes generated by TEA-3 with these inputs as $\texttt{TEA-3}_k(\texttt{IV}, t, m)$.

[1] See https://github.com/MidnightBlueLabs/TETRA_crypto/blob/main/tea3.c.

Figure 1 gives the high-level structure of TEA-3. It consists of two registers, namely the *key* register (top) and the *state* register (bottom).

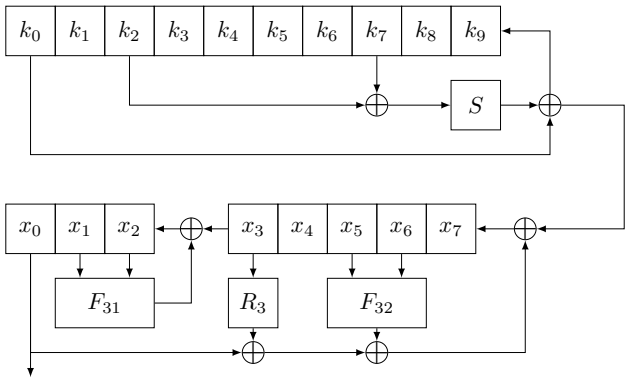


Figure 1: TEA-3 with the 10-byte key register on top and 8-byte state register below.

2.1 Internals of TEA-3

We omit the full description of the S-Box S here for brevity. Remarkably, the S-Box is not bijective as $S(\texttt{0x14}) = S(\texttt{0x9e}) = \texttt{0xc2}$ and $\texttt{0xd2}$ is not in the image of S. Notice that $S(\texttt{0xc2}) = 0$. That is, the value appearing twice in the image is mapped to zero. Notice also that the S-box is of algebraic degree 8. This degree would not be reachable if the S-Box would be a permutation.

The bit permutation $R_3 \colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ maps a byte $X = (x_0, x_1, \ldots, x_7)$, with x_0 being the least significant bit in X, to the byte $(x_3, x_7, x_6, x_1, x_2, x_4, x_0, x_5)$. The functions $F_{31} \colon \mathbb{F}_2^{16} \to \mathbb{F}_2^8$ and $F_{32} \colon \mathbb{F}_2^{16} \to \mathbb{F}_2^8$ are built similarly and both are balanced. For $F \in \{F_{31}, F_{32}\}$, and for each $0 \leq i < 8$, the coordinate functions F_i depend only on four of the sixteen input bits, two from each input byte. We give their algebraic normal forms in Appendix B.

2.2 Initialization and Keystream Generation

TEA-3 is initialized with the 80-bit cipher key $\texttt{CK}$, the so-called frame numbers (29 bits), and the network information (32 bits). We identify the frame number with a four-byte initialization vector $\texttt{IV} = (\texttt{IV}_0, \texttt{IV}_1, \texttt{IV}_2, \texttt{IV}_3)$ where the three most significant bits of $\texttt{IV}_3$ are filled with zeros, and the network information with a four-byte tweak $t = (\texttt{LA}, \texttt{CN}, \texttt{CC})$ with the 14-bit location area $\texttt{LA}$, the 12-bit carrier number $\texttt{CN}$, and the 6-bit color code $\texttt{CC}$. For initialization, both the frame numbers and the network information are expanded with

$$\texttt{expand}_\texttt{K}(t) \coloneqq \texttt{LA}||\texttt{CN}||\texttt{CC}||\texttt{CN}||\texttt{CC}||\texttt{CN}||\texttt{CC}||\texttt{CN} \quad \text{and}$$
$$\texttt{expand}_\texttt{S}(\texttt{IV}) \coloneqq (\texttt{IV}_3 \oplus \texttt{0xC4})||\texttt{IV}_3||\texttt{IV}_2||\texttt{IV}_1||\texttt{IV}_0||(\texttt{IV}_2 \oplus \texttt{0x3A})||(\texttt{IV}_1 \oplus \texttt{0x7D})||(\texttt{IV}_0 \oplus \texttt{0x51}).$$

Then, the initial state of the TEA-3 *state* register is given by $\texttt{expand}_\texttt{S}(\texttt{IV})$. That is, x_0 in Figure 1 is filled with $\texttt{IV}_3 \oplus \texttt{0xC4}$, x_1 with $\texttt{IV}_3$ and so on. Curiously enough, the used constants sum up to $\texttt{0xd2}$, i.e., the value not appearing in the image of the S-box S. The encryption cipher key $\texttt{ECK}$, i.e., the initial state of the key register is given by $\texttt{CK} \oplus \texttt{expand}_\texttt{K}(t)$. After this, both registers are synchronously clocked 32 times to bring TEA-3 in a state where keystream generation can start. Keystream bytes are generated by clocking the cipher 19 times and then taking the value held in register x_0 as the output byte (the first output byte takes 51 clocks).

Remark 1. The original specification [ETS24j, Sect. 7.2.3] of the IV expansion is clearly erroneous. Hence, our description here relies on [MBW23]. Further, strictly speaking, TEA-3 is initialized with the `ECK` directly. The computation of `ECK` is done in a different function, namely `TB5`, which is part of a different specification [ETS24i, Section 5.24]. We merged this here for simplicity.

2.3 Class-2 networks

Here, we give details on so-called Class-2 networks based on [MBW23, Bok].

Tetra devices communicate via connecting with a cell tower. Each cell tower is thereby a location area. It may have several radios operating on a few different frequencies, but in general, a single base station counts as a single location area. With a 5-20km radius, there are thus hundreds of location areas for a country-spanning network and thus, hundreds of different `ECK`s which differ. A base station may have multiple base radios, operating on different frequencies. While the location area will be identical for each base radio in the same cell, the frequency and thus carrier number differs. As already shown, the IV consists of four numbers: hyperframe, multiframe, frame and slot. These increment with time and are generally synchronized throughout the entire country-spanning network, usually using GPS time for reference. These numbers do roll over within less than a month, but the key should have changed before that happens.

In Class-2 the `ECK` is derived from the 80-bit static cipher key denoted by `SCK` in the standard (see [MBW23, Figure 4], i.e. in Class-2 networks `CK` in [ETS24j] is instantiated with the `SCK`) which is used network-wide and intended to be changed every few weeks. In practice, many network operators tend to never change the `SCK`.

3 The Theory of Word-Oriented NFSRs

This section is devoted to the theory of word-oriented NFSRs. This theory will be used to decompose the key-register in TEA-3 and afterwards to compute its possible cycle length. While there exist several papers on *bit*-oriented NFSRs, see e.g. [MST79] or [Gol81], to the best of our knowledge the theory of *word*-oriented NFSRs is not covered by common literature. However, it is quite similar to the theory of bit-oriented NFSRs. Therefore most of the results presented in this section are generalizations which we formally provide for completeness. By $\mathbb{F}_{2^n}$ we denote the finite field of degree n and even characteristic and by $\mathbb{F}_2^n$ the $\mathbb{F}_2$-vectorspace of dimension n. Let R denote the set of sequences $r = (r_i)_{i\in\mathbb{N}}, r_i \in \mathbb{F}_{2^n}$. The all-zero sequence is denoted by (0). Let $\mathcal{M}_n$ be the set of all mappings from $\mathbb{F}_{2^n}$ to itself. We consider $\mathbb{F}_{2^n}$ as a subset of $\mathcal{M}_n$ by identifying $A \in \mathbb{F}_{2^n}$ with the mapping $x \mapsto A \cdot x$. Hence for $M \in \mathcal{M}_n$ the map AM is the composition $x \mapsto A \cdot M(x)$. By $\mathbb{F}_{2^n}[X]$ we denote the univariate polynomial ring, where $\mathbb{F}_{2^n}$ is considered as a subset of $\mathcal{M}_n$ by the above identification. A *generalized polynomial* is a polynomial of the form $\sum_{i=0}^{m} H_i(G_i), H_i \in \mathcal{M}_n$ and G_i in $\mathbb{F}_{2^n}[X]$, where $H_i(G_i)$ is considered as a formal expression. Later we define how to apply this expression as a mapping on sequences over $\mathbb{F}_{2^n}$.

Let f be of the form $X^m + F(X)$, where $F(X) = \sum_{i=0}^{m-1} H_i(G_i)$, $H_i \in \mathcal{M}_n$, $G_i \in \mathbb{F}_{2^n}[X]$, and every G_i is of degree strictly less than m. We call f a register polynomial. The mapping associated to X^0 is called the constant of F. The *degree* of f is defined as m. Let R denote the set of sequences over $\mathbb{F}_{2^n}$. For a register polynomial f of degree m we define a sequence operator θ by

$$\begin{aligned}&\theta(f) : R \to R,\\ &\theta(f)(r) := r', r'_i = f(r_i, \ldots, r_{i+m}), i \in \mathbb{N},\end{aligned}$$

i.e. each $X^j, 0 \le j \le m$ is substituted by r_{i+j} and each S_i is applied to the corresponding

value in $\mathbb{F}_{2^n}$. If $\theta(f)(r) = (0)$ then r is called a *register sequence*. The set of all register sequences of f is denoted by $\Omega(f)$.

Definition 1. Let $g = X^m + \sum_{i=0}^{m-1} G_i X^i$ be a register polynomial and $h = X^k + \sum_{j=0}^{k-1} H_j X^j \in \mathbb{F}_{2^n}[X]$. We define the composition by setting

$$(g \cdot h) \coloneqq \sum_{i=0}^{m} G_i \left(\sum_{j=0}^{k} H_j X^{j+i} \right) \quad \text{or} \quad (h \cdot g) \coloneqq \sum_{i=0}^{m} H_i \left(\sum_{j=0}^{k} G_j X^{j+i} \right),$$

respectively.

The following theorem is a direct consequence of the above definition.

Theorem 1. *The following assertions hold for g, h as above:*

1. *The compositions gh and hg are again register polynomials.*
2. *The degree of gh and hg is in both cases $m + k$.*
3. *The composition hg simplifies to $\sum_{\ell=0}^{m+k} F_\ell X^\ell$, where F_ℓ denotes the mapping $\sum_{i,j,i+j=\ell} H_i(G_j)$.*
4. *It is $\theta(gh) = \theta(g) \circ \theta(h)$ and $\theta(hg) = \theta(h) \circ \theta(g)$.*
5. *r is a register sequence of gh or hg if and only if $\theta(h)(r)$ is a register sequence of g or $\theta(g)(r)$ is a register sequence of h.*

We will see that Item 5 is, although it may appear rather trivial, very useful. This fact is not restricted to register polynomials and could be generalized. The same is true as well for some of the subsequent results. As these kinds of generalizations are not of interest for this paper we restricted Item 5 and the rest of this paper to register polynomials.

Definition 2. The *period* of a register sequence r is the smallest $p \in \mathbb{N}$ such that there exist a $k \geq 0$ with $r_{k+i+p} = r_{i+k}, i \geq 0, i \in \mathbb{N}$. The minimal $k \geq 0$ with this property is called the preperiod of r.

Remark 2. For a register sequence r with period p, it is well-known that if l is another value with $r_{k+i+l} = r_{i+k}, i \geq 0$ then p divides l.

Corollary 1. *Given gh and $a \in \Omega(g)$ of period p. If $a = \theta(h)(r), r \in \Omega(gh)$ then p divides the period of r.*

Proof. If r is periodic with period ℓ then obviously $a_{k+i+\ell} = a_{k+i}$ for a proper chosen k. Hence a is periodic and the period of a divides ℓ by Remark 2. □

For a register polynomial $f(X) = X^m + F(X)$ we denote by $T_f \colon (\mathbb{F}_{2^n})^m \to (\mathbb{F}_{2^n})^m$ the mapping $(r_0, \ldots, r_{m-1}) \mapsto (r_1, r_2, \ldots, r_{m-1}, F(r_0, \ldots, r_{m-1}))$. We then have the following for the period of a register sequence.

Theorem 2. *Given a register polynomial f. If T_f is bijective then, for any sequence $r \in \Omega(f)$ of period p, we have $r_{i+p} = r_i, i \geq 0$, i.e. the preperiod is zero.*

Proof. By the definition of the preperiod k we have that $r_{k+i+p} = r_{k+i}, i \geq 0$ but $r_{k-1+p} \neq r_{k-1}$. Assume that $k \geq 1$. We then have

$$\begin{aligned} (r_{k+p}, \ldots, r_{k+p+m}) &= T_f(r_{k-1+p}, \ldots, r_{k-1+p+m-1}) \\ &= (r_k, \ldots, r_{k+m}) = T_f(r_{k-1}, \ldots, r_{k-1+m-1}), \end{aligned}$$

but $(r_{k-1+p}, \ldots, r_{k-1+p+m-1}) \neq (r_{k-1}, \ldots, r_{k-1+m-1})$. This is a contradiction. □

Remark 3. Note that T_f being bijective is equivalent to T_f being onto.

Corollary 2. *Let f be a register polynomial of the form $X^m+F(X)+C$ with $C \in \mathbb{F}_{2^n} \setminus \{0\}$, where*

$$F(X) = \sum_{i=1}^{m-1} H_i(G_i), H_i \in \mathcal{M}_n, G_i \in \mathbb{F}_{2^n}[X]$$

and every G_i is of degree strictly less than m and a constant equal to 0. Let $r \in \Omega(f)$ be a sequence with period p. Then, $r_{i+p} = r_i$ for every $i \geq 0$.

Proof. Since $C \neq 0$ and all G_i have 0 as a constant the mapping T_f is onto. Thus the result follows from Theorem 2. □

Theorem 3. *Given the composition gh of register polynomials, where $g = X^d + G(X)$ and $h = X^d + H(X)$ are of degree d. For $a \in \Omega(g)$ there exist 2^{nd} register sequences r of gh with $\theta(h)(r) = a$.*

Proof. For $r \in \Omega(gh)$ we have $a = \theta(h)(r)$ if and only if $a_i = r_{i+d} + H(r_i, \ldots, r_{i+d-1})$ for all $i \geq 0$. Hence $r_0, \ldots, r_{d-1}$ can be chosen arbitrarily but then to become an element of $\Omega(gh)$ the sequence r is determined. As the first d elements of $\theta(h)(r)$ coincide with the one of a it is mapped to a. Hence for each a there exist 2^{nd} preimages. □

Definition 3. The cascade connection of two register polynomials $g = X^d + G(X), h = X^m + H(X)$ of degree d, m, where at least one is a polynomial, is denoted by $g < h$ and defined as follows. Let r be a register sequence of h. The sequence a of $g < h$ fulfills the identity $a_{i+d} = r_i + \theta(G)(a)_i = r_i + G(a_i, \ldots, a_{i+d-1})$.

Figure 2 shows the cascade connection of $g < h$, where $g = X^5+1$ and $h = X^5+S(X^2)+1$ for the key register of TEA-3. The following theorem is well-known for cascade connections. For completeness we give a proof.

Theorem 4. *The sequences of the cascade connection $g < h$ of two register polynomials $g = X^d + G(X) \in \mathbb{F}_{2^n}[X], h = X^m + H(X)$ of degree d and m respectively (where at least one of g, h is in $\mathbb{F}_{2^n}[X]$) are exactly the register sequences of hg.*

Proof. Let r be a register sequence of h. The output sequence of the cascade connection a fulfills the identity $a_{i+d} = r_i + G(a_i, \ldots, a_{i+d-1}), i \geq 0$. So $r_i = a_{i+d} + G(a_i, \ldots, a_{i+d-1})$ and finally $\theta(g)(a) = r$. Thus, by Theorem 1, $\theta(hg)(a) = 0$. The converse is analog. □

In the following, we study the periods of a specific decomposition useful later.

Theorem 5. *Let h be a register polynomial and $g = X^m + 1$. Then, the period of a sequence in $\Omega(hg)$ is dp, where d is a divisor of $2m$ and p is a period of a sequence in $\Omega(h)$ with preperiod 0.*

Proof. We make use of the fact that $\Omega(g < h) = \Omega(hg)$ by Theorem 4. Let $a_0, \ldots, a_{m-1}$ denote the initial values of g and $r_0, \ldots, r_{d-1}$ those of h. Let k denote the output sequence and p the period of $\theta(g)(k) \in \Omega(h)$. The i-th entry of the register g at clock $0, m, 2m, \ldots$ is of the form $a_i, a_i + r_i, a_i + r_i + r_{i+m}, \ldots$. Hence, after $2pm$ clocks, the i-th entry is of the form $a_i + \sum_{j=0}^{2p-1} r_{i+mj} = a_i + \sum_{j=0}^{p-1} r_{i+mj} + \sum_{j=0}^{p-1} r_{i+mj+mp}$. Since the period of r is p, clearly $r_j = r_{j+mp}$ holds as the preperiod is 0, so each element from r appears twice in the sum (for large enough values of i). Thus, after $2mp$ clocks, the i-th entry of the register g is again a_i. As by Corollary 1 p is a divisor of the period of k, so it follows that the period can only be tp for $t = 1, \ldots, 2m$. Suppose the period is tp. Clearly, we have $k_{\ell 2mp+i} = k_i$ for any ℓ, and also $k_{utp+i} = k_i$ for any u. By substitution, we get $k_{(\ell 2m-ut)p+i} = k_i$. By Bezout's identity, we have ℓ, u such that $\gcd(2m, t) = \ell 2m - ut$, hence the period is a divisor of $\gcd(2m, t)p$ and divided by p. □

For the polynomial basis $(1, \alpha, \ldots, \alpha^{n-1})$ of $\mathbb{F}_{2^n}$ it is well-known that the map $\Phi_n : \mathbb{F}_2^n \to \mathbb{F}_{2^n}, (a_0, \ldots, a_{n-1}) \mapsto \sum_{i=0}^{n-1} a_i \alpha^i$ is an isomorphism. Applying Φ_n it is straightforward to see that the theory developed in this section applies 1-to-1 to generalized polynomials, where the set of mappings $\mathcal{M}'_n$ is over $\mathbb{F}_2^n$. For the sake of clarity, we introduced the theory for $\mathcal{M}_n$. In the next section, we consider S as an element of $\mathcal{M}'_n$ as it fits better to common descriptions of TEA-3.

4 Decomposition of the Key Register

In this section we apply the theory developed so far to give properties of TEA-3, which are the foundation for our analysis. As stated in the introduction, the key register in TEA-3 decomposes. More precisely, the original and the decomposed register depicted in the introduction are equivalent up to initialization, i.e., they generate the same output stream. To see this, consider $(k_0, \ldots, k_9)$ as the initial state of the non-decomposed key register. Then, clearly the output stream s of the key register is

$$s_0 = k_0 \oplus S(k_2 \oplus k_7),\ s_1 = k_1 \oplus S(k_3 \oplus k_8),\ s_2 = k_2 \oplus S(k_4 \oplus k_9),\ s_3 = k_3 \oplus S(k_5 \oplus s_0),$$
$$s_4 = k_4 \oplus S(k_6 \oplus s_1),\ s_5 = k_5 \oplus S(k_7 \oplus s_2),\ s_6 = k_6 \oplus S(k_8 \oplus s_3),\ s_7 = k_7 \oplus S(k_9 \oplus s_4),$$
$$s_8 = k_8 \oplus S(s_0 \oplus s_5),\ s_9 = k_9 \oplus S(s_1 \oplus s_6).$$

From this, it directly follows that the initial state $a = (a_0, \ldots, a_4)$ of the linear part of the decomposed key register must be $a = (s_0, \ldots, s_4)$. Further, for $5 \leq i \leq 9$ we have $s_i = a_{i-5} \oplus r_{i-5}$. Hence, for $(r_0, r_1, r_2, r_3, r_4) = (s_5 \oplus s_0, s_6 \oplus s_1, s_7 \oplus s_2, s_8 \oplus s_3, s_9 \oplus s_4)$ it is easy to verify that the two registers indeed produce the same output stream.

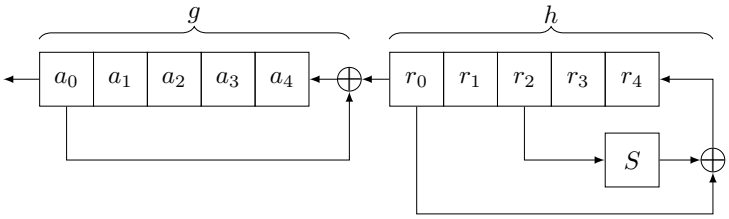


Figure 2: The decomposition of the key register of TEA-3 into the cascade connection with feedback polynomial $hg = (X^5 + S(X^2) + 1)(X^5 + 1)$. a and r are the initial states.

Remark 4. Alternatively, we can initialize g with $a = (k_0, k_1, k_2, k_3, k_4)$ and h with $r = (k_0 \oplus k_5, k_1 \oplus k_6, k_2 \oplus k_7, k_3 \oplus k_8, k_4 \oplus k_9)$ and clock g and h 10 times *without* clocking the state register. Then, g and h are in the same state as described above.

In terms of the theory established in Section 3, this means that the key register of TEA-3 is the composition of two generalized polynomials $hg = (X^5 + S(X^2) + 1)(X^5 + 1)$. Hence the key stream is a register sequence of the cascade connection $g < h$. This cascade connection is depicted in Figure 2. As the register polynomial $X^5 + S(X^2) + X$ fulfills the conditions from Theorem 2 we get the following corollary from Theorem 5.

Corollary 3 (Period lengths of the TEA-3 key register)**.** *The periods of a keystream k generated by hg where $h = (X^5 + S(X^2) + 1)$ and $g = (X^5 + 1)$ are $p, 2p, 5p$ or $10p$, where p is a period of a sequence in $\Omega(h)$.*

Remark 5. Note, that $\mathbb{F}_{2^8}$ contains the subgroup of 5-th roots of unity. Hence $X^5 + 1 = (X^4 + X^3 + X^2 + X + 1)(X + 1) = \prod_{i=0}^{4} \left(X + \zeta^i\right)$ over $\mathbb{F}_{2^8}$, ζ a generator the above subgroup. Our attacks do not use this fact.

4.1 Sequences of the Decomposed Key Register

Since the length of the sequences generated by the TEA-3 key register is largely determined by the period of the sequences in $\Omega(h)$, we are interested in identifying these sequences and their periods. Some of these are notably short. For example, if all the bytes of the register with feedback polynomial h are initialized with the value `0xc2`, it will be stuck and produce a period of length 1 as $S(\texttt{0xc2}) = 0$. We find another short sequence by noting that $S(\texttt{0x81}) \oplus \texttt{0x81} = \texttt{0xc2}$, which combined with $S(\texttt{0xc2}) = 0$ means that if this register is filled with any combination of the byte values `0xc2` and `0x81` it will only ever produce more of those same values. In this case, the register behaves like a 5-bit binary feedback shift register with feedback polynomial $X^5 + X^2 + 1$. Since this is a primitive polynomial, the sequence containing combinations of `0x81` and `0xc2` has period $2^5 - 1 = 31$.

The total number of states in the register with feedback polynomial h is 2^{40}, which is sufficiently low for us to find all the sequences formed by this feedback shift register by straightforward computation. We find that there are a total of 28 distinct sequences, with the longest having a period of approximately $2^{39.245}$. The period of all the sequences together with an initial value for each sequence is listed in Table 1. The initial values should be read as five consecutive bytes from right to left. The C-code used to find these sequences and to build Table 1 can be found in Appendix A.

Table 1: The cycle structure of output streams of the register with feedback polynomial h. For every cycle length, there is exactly one cycle of that length.

Initial value	length	$\log_2$(length)	Initial value	length	$\log_2$(length)
`0xc2c2c2c2c2`	$p_0 = 1$	0	`0x00006261e3`	$p_{14} = 2 \cdot 86113$	17.394
`0x05586fe1a3`	$p_1 = 2^2 \cdot 3$	3.585	`0x000039a19c`	$p_{15} = 5 \cdot 137869$	19.395
`0x8181818181`	$p_2 = 31$	4.954	`0x0000126e10`	$p_{16} = 3 \cdot 163 \cdot 3259$	20.604
`0x0041ec11a6`	$p_3 = 5 \cdot 7$	5.129	`0x00000c6ba3`	$p_{17} = 3 \cdot 439 \cdot 2371$	21.574
`0x093edd24b1`	$p_4 = 103$	6.687	`0x0000000039`	$p_{18} = 2 \cdot 3 \cdot 31 \cdot 117427$	24.381
`0x0096090772`	$p_5 = 2^2 \cdot 5 \cdot 7$	7.129	`0x00000037e0`	$p_{19} = 2^8 \cdot 5 \cdot 53 \cdot 1009$	26.029
`0x0006725628`	$p_6 = 2 \cdot 3 \cdot 103$	9.271	`0x000000027a`	$p_{20} = 13 \cdot 23 \cdot 43 \cdot 89 \cdot 967$	30.043
`0x0021253386`	$p_7 = 829$	9.695	`0x00000000fc`	$p_{21} = 7 \cdot 23 \cdot 14533049$	31.124
`0x00058dba94`	$p_8 = 5 \cdot 11 \cdot 139$	12.900	`0x000000007c`	$p_{22} = 2^2 \cdot 11 \cdot 29 \cdot 563 \cdot 3359$	31.168
`0x0001771efd`	$p_9 = 2 \cdot 7309$	13.835	`0x0000000002`	$p_{23} = 13 \cdot 3689014783$	35.481
`0x0004795bfa`	$p_{10} = 2 \cdot 3 \cdot 59 \cdot 61$	14.398	`0x0000000009`	$p_{24} = 67121 \cdot 755903$	35.562
`0x0003da68e1`	$p_{11} = 2^3 \cdot 3 \cdot 1283$	14.910	`0x0000000006`	$p_{25} = 7 \cdot 109 \cdot 3373 \cdot 43283$	36.697
`0x0000747c73`	$2p_{12} =^3 \cdot 19 \cdot 293$	15.443	`0x000000000a`	$p_{26} = 37 \cdot 101 \cdot 1021 \cdot 60793$	37.755
`0x00001d60ac`	$p_{13} = 2 \cdot 3 \cdot 19 \cdot 983$	16.774	`0x0000000000`	$p_{27} = 25033 \cdot 26026229$	39.245

4.2 S-box Influence on Cycle Structure

The number of distinct sequences generated by the key register, and their periods, depends on the S-box which is part of the feedback function. Since the TEA-3 S-box is unusual in that it is not a permutation, one could ask if has been chosen in order to lead to some desired properties in the number of sequences or their lengths. In order to analyze this, we considered several alternative bijective S-boxes. For those, we again computed all sequences produced by this register with the same method that we used to build Table 1.

The first two alternative S-boxes we considered are generated by replacing one of the two `0xc2` entries with the missing `0xd2` entry. For these two alternatives, the register produces 28 and 24 distinct sequences where the largest one has a period of approximately $2^{38.89}$ and $2^{39.62}$. Next, replacing the S-box with the AES S-box we get 30 sequences where the largest has an approximate period of $2^{39.90}$. We also try with one of the S-boxes used in Camellia, which results in 26 sequences where the largest has an approximate period of $2^{38.92}$. From these experiments, we cannot see that the unusual S-box leads to any special properties in the cycle structure.

5 On the functions F_{31} and F_{32}

The two functions F_{31} and F_{32} are balanced functions of degree 3 from $\mathbb{F}_2^{16}$ to $\mathbb{F}_2^8$. It is obvious from the specifications and the algebraic normal forms given in Appendix B that, up to a reordering of the input variables, both are such that their i-th coordinate f_i depends on four variables $(x_i, x_{i+1}, y_i, y_{i+1})$, where $0 \leq i < 8$ and all indices are computed modulo 8. Notice that the same applies to the feedback functions used in TEA-{1,2,4}.

However, one may wonder how these functions have been chosen since constructing a balanced function having this property is not trivial. In this section, we show that these two functions share the same specific construction which guarantees that they are balanced. Let $G : \mathbb{F}_2^{16} \to \mathbb{F}_2^8$ be a function derived from F_{31} or F_{32} by composing its input with the bijection $(x, y) \mapsto (x, x \oplus y)$.

Then, it can be checked from their algebraic normal forms given in Appendix B that the middle coordinates of G have the following form, for $1 \leq i \leq 6$:

$$f_i(x_i, x_{i+1}, y_i, y_{i+1}) = \begin{cases} y_{i+1} \oplus c_i & \text{if } y_i = \varepsilon_i \\ y_{i+1}\ell_i^1(x_i, x_{i+1}) \oplus \ell_i^0(x_i, x_{i+1}) & \text{if } y_i = 1 \oplus \varepsilon_i \end{cases} \tag{1}$$

where $\varepsilon_i, c_i \in \mathbb{F}_2$ and ℓ_i^0 and ℓ_i^1 are 2 functions of degree 1 such that $(\ell_i^0 \oplus \ell_i^1)$ has degree 1.

Then, this construction guarantees that the function

$$F^{(6)} : (x_1, \ldots, x_7, y_1, \ldots, y_7) \mapsto (f_1(x_1, x_2, y_1, y_2), \ldots, f_6(x_6, x_7, y_6, y_7))$$

is balanced as shown by the following proposition.

Proposition 1. *Let n be an integer and $F^{(n)}$ be the function from $\mathbb{F}_2^{2n+2} \to \mathbb{F}_2^n$ defined by*

$$F^{(n)}(x_1, \ldots, x_{n+1}, y_1, \ldots, y_{n+1}) = (f_1(x_1, x_2, y_1, y_2), \ldots, f_n(x_n, x_{n+1}, y_n, y_{n+1}))$$

with f_i, $1 \leq i \leq n$, defined by (1) where ε_i for $2 \leq i \leq n$ is such that $\ell_i^0 \oplus \ell_i^1 \oplus \varepsilon_{i+1}\ell_i^1 \oplus x_{i+1}$ has degree 1. Then, $F^{(n)}$ is balanced.

Notice, for any i, such an ε_{i+1} always exists as at least one of the two functions $(\ell_i^0(x_i, x_{i+1}) \oplus x_{i+1})$ and $((\ell_i^0 \oplus \ell_i^1)(x_i, x_{i+1}) \oplus x_{i+1})$ has degree 1. Indeed, since ℓ_i^0, ℓ_i^1 and $\ell_i^0 \oplus \ell_i^1$ all have degree 1, the linear parts of ℓ_i^0 and of $(\ell_i^0 \oplus \ell_i^1)$ cannot both equal x_{i+1}.

The fact that $F^{(n)}$ is balanced equivalently means that its components $\gamma \cdot F^{(n)}$ are balanced for all nonzero $\gamma \in \mathbb{F}_2^n$. This is deduced from the fact that the sum of any k consecutive coordinates of $F^{(n)}$, $g_{u,k} := f_u + \ldots + f_{u+k-1}$, is balanced. Indeed, as detailed in Appendix C, it can be proved by induction on k that $g_{u,k}$ is balanced and the restriction of $(g_{u,k} \oplus x_{u+k})$ to $y_{u+k} = 1 \oplus \varepsilon_{u+k}$ is balanced. This proof relies on the following key lemma.

Lemma 1. *Let g be a balanced Boolean function of $2k$ variables $(x_1, \ldots, x_k, y_1, \ldots, y_k)$ such that the restriction of $g \oplus x_k$ to all inputs with $y_k = 1 \oplus \varepsilon$ is balanced for some $\varepsilon \in \mathbb{F}_2$. Then, the Boolean function of $2(k+1)$ variables*

$$f(x_1, \ldots, x_{k+1}, y_1, \ldots, y_{k+1}) := g(x_1, \ldots, x_k, y_1, \ldots, y_k) \oplus f_k(x_k, x_{k+1}, y_k, y_{k+1})$$

is balanced where f_k is defined by (1) with $\varepsilon_k = \varepsilon$. Moreover, f satisfies the following additional properties:

(i) *If $g \oplus y_k$ is balanced, then $f \oplus y_{k+1}$ is balanced.*

(ii) *If $\ell_k^0(x_k, x_{k+1}) \oplus x_{k+1}$ has degree 1, then the restriction of $f \oplus x_{k+1}$ to all inputs with $y_{k+1} = 0$ is balanced;*

(iii) *If* $(\ell_k^0 \oplus \ell_k^1)(x_k, x_{k+1}) \oplus x_{k+1}$ *has degree* 1*, then the restriction of* $f \oplus x_{k+1}$ *to all inputs with* $y_{k+1} = 1$ *is balanced.*

Proof. • Let us first prove that f is balanced. Let f_ε (resp. $f_{1\oplus\varepsilon}$) be the restriction of f to $y_k = \varepsilon$ (resp. to $y_k = 1 \oplus \varepsilon$). Then,

$$f_\varepsilon(x_1, \ldots, x_{k+1}, y_1, \ldots, y_{k+1}) = g(x_1, \ldots, x_k, y_1, \ldots, y_{k-1}, \varepsilon) \oplus y_{k+1} \oplus c_k \quad (2)$$

which is balanced since g does not depend on y_{k+1}. Moreover,

$$f_{1\oplus\varepsilon}(x_1, \ldots, y_{k+1}) = g(x_1, \ldots, x_k, y_1, \ldots, y_{k-1}, 1\oplus\varepsilon) \oplus y_{k+1}\ell_k^1(x_k, x_{k+1}) \oplus \ell_k^0(x_k, x_{k+1}) \ .$$

Then, the two restrictions of $f_{1\oplus\varepsilon}$ to $y_{k+1} = 0$ and to $y_{k+1} = 1$ are of the form

$$g(x_1, \ldots, x_k, y_1, \ldots, y_{k-1}, 1\oplus\varepsilon) \oplus \ell(x_k, x_{k+1}) \text{ with } \ell(x_k, x_{k+1}) = \begin{cases} \ell_k^0(x_k, x_{k+1}) \\ (\ell_k^0 \oplus \ell_k^1)(x_k, x_{k+1}) \end{cases}$$

Since ℓ_k^0 and $(\ell_k^0 \oplus \ell_k^1)$ have degree 1, the linear part of ℓ belongs to $\{x_k, x_k \oplus x_{k+1}, x_{k+1}\}$. Because g does not depend on x_{k+1}, this function is balanced when the linear part of ℓ involves x_{k+1}. Moreover, if the linear part of ℓ equals x_k, it is balanced too since the restriction of $g \oplus x_k$ to $y_k = 1 \oplus \varepsilon$ is balanced by hypothesis. Therefore, the 2 restrictions of $f_{1\oplus\varepsilon}$ are balanced, implying that $f_{1\oplus\varepsilon}$ is balanced.

- For proving (i), we first use the fact that the restrictions of $f_{1\oplus\varepsilon}$ to $y_{k+1} = 0$ and to $y_{k+1} = 1$ are balanced, implying that $f_{1\oplus\varepsilon} \oplus y_{k+1}$ is balanced. Moreover, we know from (2) that, when y_{k+1} is fixed, f_ε equals $g(x_1, \ldots, x_k, y_1, \ldots, y_{k-1}, \varepsilon)$ up to a constant. When both g and $(g \oplus y_k)$ are balanced, the restriction of g to $y_k = \varepsilon$ (and to $y_k = \varepsilon \oplus 1$) is balanced, implying that the restrictions of f_ε to $y_{k+1} = 0$ and to $y_{k+1} = 1$ are balanced. It follows that the restrictions of f to $y_{k+1} = 0$ and to $y_{k+1} = 1$ are balanced. Hence $(f \oplus y_{k+1})$ is balanced.
- For proving (ii) and (iii), we consider the restriction h_α of $(f \oplus x_{k+1})$ to $y_{k+1} = \alpha$, $\alpha \in \mathbb{F}_2$. Then, if $y_k = \varepsilon$,

$$h_\alpha(x_1, \ldots, y_{k+1}) = g(x_1, \ldots, x_k, y_1, \ldots, y_{k-1}, \varepsilon) \oplus x_{k+1} \oplus (\alpha \oplus c_k)$$

which is obviously balanced. If $y_k = 1 \oplus \varepsilon$,

$$h_\alpha(x_1, \ldots, y_{k+1}) = g(x_1, \ldots, x_k, y_1, \ldots, y_{k-1}, 1 \oplus \varepsilon) \oplus (\alpha\ell_k^1 \oplus \ell_k^0)(x_k, x_{k+1}) \oplus x_{k+1} \ ,$$

which is balanced if $(\alpha\ell_k^1 \oplus \ell_k^0) \oplus x_{k+1}$ has degree 1. The result then directly follows. □

Remark 6. Prop. 1 similarly holds if the first coordinate of $F^{(n)}$ is replaced by any balanced function f_1' which satisfies the hypothesis of Lemma 1, i.e., the restriction of $(f_1' \oplus x_2)$ to $y_2 = 1 \oplus \varepsilon_2$ is balanced.

Remark 7. We also deduce from the proof of Prop. 1 that, for any $\gamma \in \mathbb{F}_2^n$, the restriction of $(\gamma \cdot F^{(n)} \oplus x_{n+1})$ to $y_{n+1} = 0$ (resp. to $y_{n+1} = 1$) is balanced if $\ell_n^0 \oplus x_{n+1}$ (resp. $\ell_n^0 \oplus \ell_n^1 \oplus x_{n+1}$) has degree 1.

Functions F_{31} and F_{32} in TEA-3 then use the construction described in Prop. 1, and more specifically its generalization in Remark 6, for $n = 6$ for defining their 6 middle coordinates. Two additional coordinates, g_0 and g_7, that have to be carefully chosen, then need to be appended. To this aim, we can take advantage of the degrees of freedom we have in choosing the first coordinate of $F^{(n-2)}$, as explained in Remark 6, so that the resulting function has some symmetries.

Corollary 4. *Let*

$$f_1'(x_1,x_2,y_1,y_2) = \begin{cases} (y_2 \oplus \varepsilon_2 \oplus 1)\ell_1(x_1,x_2) \oplus c_1 & \text{if } y_1 = 0 \\ (y_2 \oplus \varepsilon_2 \oplus 1)\ell_1'(x_1,x_2) \oplus c_1 \oplus 1 & \text{if } y_1 = 1 \end{cases} \tag{3}$$

where ℓ_1 and ℓ_1' are two functions of degree 1 such that $\deg(\ell_1 \oplus x_1) = 1$ and $\deg(\ell_1' \oplus x_1) = 1$. Let n be an integer and $\widetilde{F}^{(n)}$ be the function from $\mathbb{F}_2^{2n+2} \to \mathbb{F}_2^n$ defined by

$$\widetilde{F}_n^{(n)}(x_1,\ldots,y_{n+1}) = (f_1'(x_1,x_2,y_1,y_2), f_2(x_2,x_3,y_2,y_3),\ldots,f_n(x_n,x_{n+1},y_n,y_{n+1}))$$

with f_i, $2 \le i \le n$, defined by (1) where ε_i for $2 \le i \le n-1$ is such that $\ell_i^0 \oplus \ell_i^1 \oplus \varepsilon_{i+1}\ell_i^1 \oplus x_{i+1}$ has degree 1, $\deg(\ell_n^0 \oplus x_{n+1}) = 1$ and $(\ell_n^1 \oplus x_{n+1})$ is constant. Then, $\widetilde{F}^{(n)}$ is balanced and, for every fixed values $a, b \in \mathbb{F}_2$ and every $z \in \mathbb{F}_2^n$,

$$N_{a,b}(u,v,z) = \#\{(x_2,\ldots,x_n,y_2,\ldots y_n) : \widetilde{F}^{(n)}(u,x_2,\ldots,x_n,v,a,y_2,\ldots,y_n,b) = z\}$$

does not depend on u and v.

Proof. The fact that $\widetilde{F}^{(n)}$ is balanced is derived from Remark 6, by observing that f_1' is balanced and that the restriction of $(f_1' \oplus x_2)$ to $y_2 = 1 \oplus \varepsilon_2$ is balanced as it has degree 1.

From now on, we set y_1 and y_{n+1} to some fixed values a and b. We define the $2n$-variable Boolean function

$$g_\omega(x_1,\ldots,x_{n+1},y_2\ldots,y_n) = \omega \cdot \widetilde{F}^{(n)}(x_1,\ldots,x_{n+1},a,y_2,\ldots,y_n,b),\ \omega \in \mathbb{F}_2^n\ .$$

Then, for any $\omega \in \mathbb{F}_2^n$ and any $\alpha, \beta \in \mathbb{F}_2$, we have

$$\begin{aligned} \widehat{N}_{a,b}(\alpha,\beta,\omega) &= \sum_{z \in \mathbb{F}_2^n} \sum_{u,v \in \mathbb{F}_2} N_{a,b}(u,v,z)(-1)^{\omega \cdot z \oplus \alpha u \oplus \beta v} \\ &= \sum_{x,y \in \mathbb{F}_2^{n-1} \times \mathbb{F}_2^{n-1}} \sum_{u,v \in \mathbb{F}_2} (-1)^{g_\omega(u,x,v,y) \oplus \alpha u \oplus \beta v}\ . \end{aligned}$$

The inverse Fourier transform leads to

$$N_{a,b}(u,v,z) = 2^{-n} \sum_{\alpha,\beta \in \mathbb{F}_2} \sum_{\omega \in \mathbb{F}_2^n} \widehat{N}_{a,b}(\alpha,\beta,\omega)(-1)^{\alpha u \oplus \beta v \oplus \omega \cdot z}\ , \forall u,v \in \mathbb{F}_2, z \in \mathbb{F}_2^n\ .$$

Since $\widehat{N}_{a,b}(\alpha,\beta,\omega)$ is equal to the correlation of the function $g_\omega \oplus \alpha x_1 \oplus \beta x_{n+1}$, we derive that $N_{a,b}(u,v,z)$ does not depend on (u,v) if all functions $(g_\omega \oplus \alpha x_1 \oplus \beta x_{n+1})$, for $(\alpha,\beta) \neq (0,0)$ are balanced.

Let us now prove this last property. When the first coordinate of ω vanishes, then $(g_\omega \oplus x_1 \oplus \beta x_{n+1})$ is obviously balanced since g_ω does not depend on x_1. Furthermore, the fact that $(g_\omega \oplus x_{n+1})$ is balanced is derived from Remark 7, because in this case, g_ω is a sum of functions f_k, implying that the restriction of $(\omega \cdot \widetilde{F}^{(n)} \oplus x_{n+1})$ to $y_{n+1} = b$ is balanced for any b because $\ell_n^0 \oplus x_{n+1}$ and $\ell_n^0 \oplus \ell_n^1 \oplus x_{n+1}$ have degree 1 by hypothesis.

When the first coordinate of ω equals 1, then we have to prove that $(f_1'(x_1,x_2,a,y_2) \oplus \alpha x_1 \oplus g_{\omega'} \oplus \beta x_{n+1})$ is balanced, where ω' is obtained by changing the first coordinate of ω to 0. This can be deduced from Prop. 1 and Remark 6, starting from $\widetilde{f}_1 = f_1'(x_1,x_2,a,y_2) \oplus \alpha x_1$, for $\alpha, a \in \mathbb{F}_2$. Indeed, it can be checked that these four functions are such that the restrictions of $(\widetilde{f}_1 \oplus x_2)$ to $y_2 = 1 \oplus \varepsilon_2$ are balanced, since they have degree 1. It follows that, as noticed in Remark 7, the restriction $(\widetilde{f}_1 \oplus \omega' \cdot \widetilde{F}^{(n)} \oplus x_{n+1})$ to $y_{n+1} = b$, i.e. the function $(g_\omega \oplus \alpha x_1 \oplus x_{n+1})$ is balanced for all $b \in \mathbb{F}_2$ because $\deg(\ell_n^0 \oplus x_{n+1}) = 1$ and $\deg(\ell_n^0 \oplus \ell_n^1 \oplus x_{n+1}) = 1$.

Finally, we have to handle the case $\alpha = 1$ and $\beta = 0$, i.e. prove that $(f_1' \oplus x_1 \oplus g_{\omega'})$ is balanced. This is derived from the recursive application of Property (i) in Lemma 1, starting from the observation that $(f_1' \oplus x_1 \oplus y_2)$ is balanced both on $y_1 = 0$ and on $y_1 = 1$ because $\deg(\ell_1 \oplus x_1) = 1$ and $\deg(\ell_1' \oplus x_1) = 1$. □

Theorem 6. *Let n be an integer and $G^{(n)}$ be the function from $\mathbb{F}_2^{2n}$ to $\mathbb{F}_2^n$ defined by*

$$G^{(n)}(x_0,\ldots,x_{n-1},y_0,\ldots,y_{n-1}) = (z_0,\ldots,z_{n-1})$$

$$\text{where } \begin{cases} z_0 & = g_0(x_0,x_1,y_0,y_1) \\ z_1 & = f_1'(x_1,x_2,y_1,y_2) \\ z_i & = f_i(x_i,x_{i+1},y_i,y_{i+1}) \text{ for } 2 \le i < n-1 \\ z_{n-1} & = g_{n-1}(x_{n-1},x_0,y_{n-1},y_0) \end{cases}$$

and f_1' and f_i are defined by (1) and (3) and satisfy the hypotheses of Corollary 4. Assume that g_0 and g_{n-1} are such that the functions

$$(x_0,x_1,x_{n-1},y_0) \mapsto (g_0(x_0,x_1,y_0,a), g_{n-1}(x_{n-1},x_0,b,y_0))$$

are balanced for all $a,b \in \mathbb{F}_2$. Then, $G^{(n)}$ is balanced.

Proof. Let $z \in \mathbb{F}_2^n$ and $\mathcal{N}(z) = \#\{(x,y) \in \mathbb{F}_2^{2n} : G^{(n)}(x,y) = z\}$. Then,

$$\begin{aligned} \mathcal{N}(z) &= \sum_{a,b\in\mathbb{F}_2} \sum_{u,v\in\mathbb{F}_2} \#\{(x_0,x_2\ldots,x_{n-2},y_0,y_2,\ldots,y_{n-2}) : g_0(x_0,u,y_0,a) = z_0, \\ &\qquad \widetilde{F}^{(n-2)}(u,x_2,\ldots,x_{n-2},v,a,y_2,\ldots,y_{n-2},b) = z' \text{ and } g_{n-1}(v,x_0,b,y_0) = z_{n-1}\} \\ &= \sum_{a,b\in\mathbb{F}_2} \sum_{u,v\in\mathbb{F}_2} \#\{(x_0,y_0) : g_0(x_0,u,y_0,a) = z_0, g_{n-1}(v,x_0,b,y_0) = z_{n-1}\} N_{a,b}(u,v,z') \end{aligned}$$

where $z' = (z_1,\ldots,z_{n-2})$, $\widetilde{F}^{(n-2)}$ is defined as in Corollary 4 and

$$N_{a,b}(u,v,z') = \#\{(x_2,\ldots,x_n,y_2,\ldots y_n) : \widetilde{F}^{(n-2)}(u,x_2,\ldots,x_{n-2},v,a,y_2,\ldots,y_{n-2},b) = z'\}.$$

We know from Corollary 4 that $N_{a,b}(u,v,z')$ does not depend on (u,v). It follows that

$$\begin{aligned} \mathcal{N}(z) &= \sum_{a,b\in\mathbb{F}_2} N_{a,b}(0,0,z') \left(\sum_{u,v\in\mathbb{F}_2} \#\{(x_0,y_0) : g_0(x_0,u,y_0,a) = z_0, g_{n-1}(v,x_0,b,y_0) = z_{n-1}\} \right) \\ &= 4 \sum_{a,b\in\mathbb{F}_2} N_{a,b}(0,0,z') = \#\left(\widetilde{F}^{(n-2)}\right)^{-1}(z') \end{aligned}$$

because the mapping $(x_0,y_0,u,v) \mapsto (g_0(x_0,u,y_0,a), g_{n-1}(v,x_0,b,y_0))$ is balanced. Since $\widetilde{F}^{(n-2)}$ is balanced, we deduce that $\mathcal{N}(z) = \#\left(\widetilde{F}^{(n-2)}\right)^{-1}(z') = 2^n$. □

Theorem 6 then provides a simple algorithm for constructing many balanced functions from $\mathbb{F}_2^{2n}$ to $\mathbb{F}_2^n$ with a low-cost implementation. Determining whether this particular construction, especially the fact that each of the middle coordinates is linear on a hyperplane, introduces some weaknesses remains open.

6 Cryptanalysis

For attacking stream ciphers, a number of methods are known for state recovery, analysis of the initialization mechanism, and distinguishers of the keystream. Established methods include algebraic and correlation attacks as well as TMDTO attacks. Our focus is on generic as well as dedicated attacks. To begin with, it can be easily deduced that a key-recovery attack on TEA-3 using exhaustive search requires around $2^{85.67}$ iterations of the cipher (see Appendix E.3.1). Also a simple TMDTO attack to recover the internal state of the cipher would need $\approx 2^{79.83}$ iterations of TEA-3, with $T_{\mathsf{insertions}} = 2^{70.42}$ table-insertions, $T_{\mathsf{lookups}} = 2^{73.58}$ table-lookups and $M = 6 \cdot 2^{72}$ bytes of memory (see Appendix E.3.2).

6.1 Observation for Shifted Keystreams

Let us analyze a generic design resembling TEA-3, that employs an initial vector of length ℓ bits that is expanded into a 64 bit string using an affine map $\mathsf{Aff} : \{0,1\}^\ell \to \{0,1\}^{64}$, which can be seen as the tuple consisting of full-rank matrix M of size $64 \times \ell$ over $\mathbb{F}_2$ and a vector $\vec{b} \in \{0,1\}^{64}$, so that $\mathsf{Aff}(\vec{x}) = M \cdot \vec{x} \oplus \vec{b}$.

Consider a key $\texttt{ECK}$ and an initial vector $\texttt{IV}_1$. If TEA-3 is initialized with $(\texttt{ECK}, \texttt{IV}_1)$, then after 51 iterations, denote the internal state it produces as $S_1 = (\mathcal{K}, \mathcal{V}_1)$ where $\mathcal{K} \in \mathbb{F}_2^{80}$ is the value in the key register and $\mathcal{V}_1 \in \mathbb{F}_2^{64}$ is the value of the state register. Let $\mathcal{K}$ be a vector for which the TEA-3 registers produce a key sequence of period $10p$ for some p (since we know that all keys produce sequences of period $10p$). Run the cipher for $190p$ iterations. Let the new state after $190p$ iterations be denoted by $S_2 = (\mathcal{K}, \mathcal{V}_2)$. We want to determine if it is possible that for some other initial vector $\texttt{IV}_2$ and key $\texttt{ECK}$, we have that S_2 is the state of the cipher after 51 iterations itself? To answer this question, we can clock the cipher backwards from S_2 for 51 iterations and check if the resulting state $S = (\texttt{ECK}, V)$ is a valid initial state of TEA-3. This only happens if V is in the image of the affine map Aff, i.e. there exists an $\texttt{IV}_2 \in \{0,1\}^\ell$ such that $M \cdot \texttt{IV}_2 \oplus \vec{b} = V$. For example if $\ell = 32$ and we had used the original $\texttt{expand}_\texttt{S}$ function, then the above condition would be satisfied if $V = v_0, v_1, \cdots, v_7$ satisfied the following bit-conditions.

$$v_0 = v_1 \oplus \texttt{0xC4} \qquad v_2 = v_5 \oplus \texttt{0x3A} \qquad v_3 = v_6 \oplus \texttt{0x7D} \qquad v_4 = v_7 \oplus \texttt{0x51} \tag{4}$$

When this happens $\texttt{IV}_2$ is an initial vector such that $(\texttt{ECK}, \texttt{IV}_1)$ and $(\texttt{ECK}, \texttt{IV}_2)$ generate exactly $10p$ shifted keystream bytes. The process is explained pictorially in Figure 3. The probability that V lies in the image of Aff is naturally equal to $\rho = \frac{|\mathsf{Im}(\mathsf{Aff})|}{2^{64}} = 2^{\ell-64}$. So the probability that the above exercise will succeed for any randomly chosen IV is small. However if we exhaust all the 2^ℓ initial vectors in the space, then the above set of experiments is expected to have around $2^{2\ell-64}$ successes, i.e. we find on average $2^{2\ell-64}$ initial vector pairs $\texttt{IV}_1, \texttt{IV}_2$ after the back-tracking. If $\ell = 32$, this value is equal to 1.

However consider a second exercise: let $S_3 = (\mathcal{K}, \mathcal{V}_3)$ denote the state after $380p$ iterations of the cipher. We want to determine if one of the following situations is possible:

- For some other initial vector $\texttt{IV}_2$: S_2 is the state of the cipher after 51 iterations or
- For some other initial vector $\texttt{IV}_3$: S_3 is the state of the cipher after 51 iterations.

Assuming that the events are disjoint, this probability can be roughly estimated to be $2 \cdot \rho = 2^{\ell-63}$. Let α be an integer parameter. In general the probability that any one of the α states S_i (for $2 \le i \le \alpha+1$) gives a valid state after backtracking is $\alpha \cdot 2^{\ell-64}$. This means that for any key $\texttt{ECK}$ there exist on average around $(\alpha-1) \cdot 2^{2\ell-64}$ IV pairs that generate up to $10p\alpha$ byte-shifted keystream sequences. For a fixed key $\texttt{ECK}$, now we consider the space

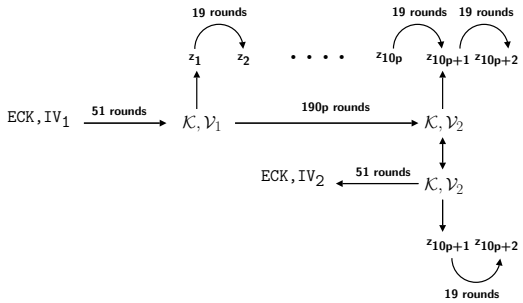


Figure 3: Generating IV pairs that produce shifted keystream bytes.

of IVs as a graph in which the IVs are nodes and $(\texttt{IV}_1, \texttt{IV}_2)$ are connected if $(\texttt{ECK}, \texttt{IV}_1)$ and

$(\texttt{ECK}, \texttt{IV}_2)$ generate either $10p, 20p, \cdots, 10p\alpha$ byte-shifted keystream bytes. From the above discussion we know that the cardinality of the set of edges is around $\mathsf{Epair} = \alpha \cdot 2^{2\ell-64}$, indicated by the blue edges in Figure 4. Assume that we know a priori the value of p, and α is chosen such that $\mathsf{Epair} \geq 1$. Our task is to query random initial vectors $\texttt{IV}$ and collect keystream sequences for the unknown key $\texttt{ECK}$ and $\texttt{IV}$. We want to determine many random IV's we need to query before we get two initial vectors that produce up to $10p\alpha$ shifted keystream bytes. In other words we do the following experiment outlined in Algorithm 1. In this random experiment we query random IVs and check if there exist

Algorithm 1: Finding IV pairs that generate shifted keystream bytes.

FindIVpair(p, α)
Input: p: The value such that the Secret Key $\texttt{ECK}$ produces sequence of period $10p$
Input: α: Integer parameter larger than 1
Output: $(\texttt{IV}_1, \texttt{IV}_2, i)$: $(\texttt{ECK}, \texttt{IV}_1), (\texttt{ECK}, \texttt{IV}_2)$ give $10pi$ byte shifted keystream,$i \leq \alpha$

Initialize an empty hash-table **Tab**
while *True* **do**
  Choose random initial vector $\texttt{IV} \xleftarrow{\$} \mathbb{F}_2^\ell$
  Generate keystream bytes $z_1, \ldots, z_{10p\alpha+18}$ with the key-IV pair $(\texttt{ECK}, \texttt{IV})$
  for $r \leftarrow 0$ *to* α **do**
    Denote $Z_r := [z_{10pr+1}, z_{10pr+2}, \ldots, z_{10pr+18}]$
    if ***Tab***$[Z_r]$ *is empty* **then** Store $(\texttt{IV}, r)$ in **Tab**$[Z_r]$
    else return $(\texttt{IV}_1 = \textbf{Tab}[Z_r].\texttt{IV}, \texttt{IV}_2 = \texttt{IV}, i = |r - \textbf{Tab}[Z_r].r|)$
  end
end

initial vectors $\texttt{IV}_1, \texttt{IV}_2$ and integers r_1, r_2 such that the r_1-th block of 18 keystream bytes produced by $\texttt{ECK}, \texttt{IV}_1$ equals the r_2-th block produced by $\texttt{ECK}, \texttt{IV}_2$. This indicates with high probability that the internal states produced by $(\texttt{ECK}, IV_1)$ at instance r_1 and that produced by $(\texttt{ECK}, \texttt{IV}_2)$ at instance r_2 are the same and thus they produce $10p|r_1 - r_2|$ byte shifted keystream. If we query X initial vectors we are actually testing all the $\binom{X}{2}$ edges that are formed between them (the red edges in Figure 4). The blue edges are those that connect the shifted IVs. A collision occurs when the product of number of red and blue edges equals the total possible number of edges. Thus we have

$$\binom{X}{2} \cdot \alpha \cdot 2^{2\ell-64} = \binom{2^\ell}{2} \Rightarrow X = \frac{2^{32}}{\sqrt{\alpha}}$$

We know that p can only have 28 values the largest of which p_{27} is just above 2^{39}. So to begin with we can generate $10p_{27}\alpha + 18$ bytes from $X = 2^{32}/\sqrt{\alpha}$ random initial vectors. Since the IV space has 2^ℓ vectors, for the attack to make sense we need both $\mathsf{Epair} = \alpha \cdot 2^{2\ell-64} \geq 1$ and $2^{32} \leq \sqrt{\alpha} \cdot 2^\ell$, both of which lead to $\alpha \geq 2^{64-2\ell}$.

The algorithm requires $T_{\mathsf{complexity}} = (51 + 190p_{27}\alpha + 342) \cdot X \approx 2^{79}\sqrt{\alpha}$ iterations of TEA-3. Now for $i = 27$ downto 0, we run the algorithm **FindIVpair**(p_i, α), for $X = 2^{32}/\sqrt{\alpha}$ initial vectors (without generating anymore keystream) and move to the next lower value of i, if the algorithm does not output anything. We should get a collision whp for some value of i after which we get the value of p_i to which the Key belongs to. The number of hash insertions is around $T_{\mathsf{insertions}} = \alpha \cdot 28 \cdot X \approx 2^{38}\sqrt{\alpha}$. If $\ell = 32$, then by choosing $\alpha = 2$ this results in a distinguisher to determine the period p_i with computational complexity lower than state recovery of Appendix E.3.2, and with table access complexity significantly lower than state recovery. Indeed, if the period is wrongly guessed the probability of such a collision is $\frac{2^\ell}{2^{144}}$ and thus is not expected to occur when conducting the above experiment.

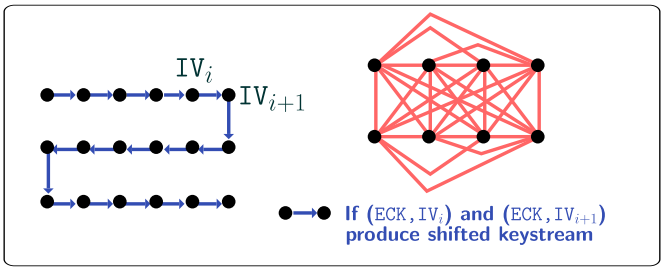


Figure 4: Space of initial vectors as a graph. $\mathtt{IV}_i, \mathtt{IV}_{i+1}$ are connected with a blue edge if they produce upto $10p\alpha$ byte-shifted keystream sequences.

However this does not give us the actual key, but only tells us in which class the secret key lies in. However when $\ell = 29$, this forces us to choose $\alpha \geq 2^6 = 64$, which results in $T_{\text{complexity}} \approx 2^{82}$. This gives some justification as to why the length of the IV has been somewhat awkwardly set at 29 bits.

6.1.1 Forcing Short Periods

We extend our observations on shifted keystreams by considering the tweak which is used to derive ECK. For this, recall from Remark 4 that initializing the decomposed register in Figure 2 with $(a_0, a_1, a_2, a_3, a_4) = (k_0, k_1, k_2, k_3, k_4)$ and $(r_0, r_1, r_2, r_3, r_4) = (k_0 \oplus k_5, k_1 \oplus k_6, k_2 \oplus k_7, k_3 \oplus k_8, k_4 \oplus k_9)$ produces the same sequence as the original un-decomposed key register in Figure 1. The h-register produces sequences of one of 28 periods $p_i,\ i \in [0, 27]$ as tabulated in Table 1 and it has also been shown that the key sequence (s_i) has period either $p_i, 2p_i, 5p_i$ or $10p_i$. Indeed, the 28 periods divide the state space of the register into 28 disjoint partitions or orbits. Given a 10-byte master key $K = (k_0, k_1, k_2, \ldots, k_9)$, define its short-key as the 5-byte vector $\mathtt{SL} = (k_0 \oplus k_5, k_1 \oplus k_6, k_2 \oplus k_7, k_3 \oplus k_8, k_4 \oplus k_9)$, corresponding to the initial state of the h-register. If SL happens to reside in an orbit of smaller size, then we might be able to leverage an attack on the keystream generator.

However, the short-key corresponding to an arbitrary master key Kdoes not necessarily lie in a small orbit. An active attacker can however freely choose the 14-bit location area (LA), 12-bit carrier number (CN) and the 6-bit color code (CC) to force the h-register into a low period orbit. The 32-bit vector $t = \mathtt{LA}||\mathtt{CN}||\mathtt{CC}$ is expanded to 80 bits by $E = \mathtt{expand}_{\mathtt{K}}(t) = (\mathtt{LA} \parallel \mathtt{CN} \parallel \mathtt{CC} \parallel \mathtt{CN} \parallel \mathtt{CC} \parallel \mathtt{CN} \parallel \mathtt{CC} \parallel \mathtt{CN})$ which are xored to the K to give the encryption key $\mathtt{ECK} = K \oplus E$. If e_i's denote the bits of E, i.e $\mathtt{LA} = (e_0, e_1, \cdots, e_{13})$, $\mathtt{CN} = (e_{14}, e_{15}, \cdots, e_{25})$ and $\mathtt{CC} = (e_{26}, e_{27}, \cdots, e_{31})$ then the short-key SL is given as

$$
\begin{aligned}
\mathtt{SL}_0 &= k_0 \oplus k_5 \oplus (e_0, e_1, \cdots e_7) \oplus (e_{22}, e_{23}, \cdots, e_{29}) \\
\mathtt{SL}_1 &= k_1 \oplus k_6 \oplus (e_8, e_9, \cdots e_{15}) \oplus (e_{30}, e_{31}, e_{14}, e_{15}, \cdots, e_{19}) \\
\mathtt{SL}_2 &= k_2 \oplus k_7 \oplus (e_{16}, e_{17}, \cdots e_{23}) \oplus (e_{20}, e_{21}, \cdots, e_{27}) \\
\mathtt{SL}_3 &= k_3 \oplus k_8 \oplus (e_{24}, e_{25}, \cdots e_{31}) \oplus (e_{28}, e_{29}, \cdots, e_{31}, e_{14}, e_{15}, \cdots, e_{17}) \\
\mathtt{SL}_4 &= k_4 \oplus k_9 \oplus (e_{14}, e_{15}, \cdots e_{21}) \oplus (e_{18}, e_{19}, \cdots, e_{25}).
\end{aligned}
$$

Since the keystream generator is initialized with the session key ECK, we want to see if the corresponding SL can be part of an orbit with small size. If we keep e_8 to e_{31} fixed to some constant, and by varying $e_0, \ldots, e_7$ to all the 256 strings of 8 bits, we obtain 256 short session keys in which the first byte takes all the byte values and the 2nd to 5th bytes are constant. With high probability, one of the SL's belongs to a small orbit of size $\leq p_{22}$.

In fact a simple experiment can be done as follows: we consider all the orbits of sizes p_0 to p_{22} as shown in Table 1. For each of the i orbits $(0 \leq i \leq 22)$, we denote the p_i elements of the orbits as $X_{i,j},\ j \in [0, p_i - 1]$, where each $X_{i,j}$ is a 5-byte string. We want

to find out what values the last 4 bytes of X_j can take i.e. $X_{i,j} \wedge \texttt{0x00ffffffff}$. We iterate over all the p_i elements of the orbit and insert in a set S, all the possible values of $X_{i,j} \wedge \texttt{0x00ffffffff}$ for $i \in [0, 22]$. Doing this for all elements of all the i orbits of size upto p_{22}, i.e. for $0 \leq i \leq 22$, we find that $|S| = 3225630085 = 2^{31.5869}$. Therefore we have $|S^c| = 2^{32} - |S| = 1069337211 = 2^{29.9941}$.

Although a given master key may be such that its short-key does not lie in a small orbit, we can try to see if any of the short-keys of the derived session keys lie in a small orbit by varying the e_i's. If we generate 2^8 session keys by varying $e_0, e_1, \ldots, e_7$, then the corresponding short-keys will take all the 256 values in the first byte i.e. since only $\texttt{SL}_0$ changes as per the expressions listed earlier. Now if $[\texttt{SL}_1, \texttt{SL}_2, \texttt{SL}_3, \texttt{SL}_4] \in S$, then it is certain that one of the 256 short-keys lie in a period of size less than p_{22}, since we are generating all the values in the first byte $\texttt{SL}_0$. Now $[\texttt{SL}_1, \texttt{SL}_2, \texttt{SL}_3, \texttt{SL}_4] \in S$ with probability $\frac{|S|}{2^{32}} \approx 0.751$, which is therefore the probability that at least one of the session keys is such that its short-key lies in an orbit of size less than or equal to $p_{22} \approx 2^{31.168}$.

To perform the attack we repeat the attack of the previous section for each of the 256 session keys, but instead of p_{28} we generate only till $10p_{22}\alpha + 18$ bytes and search for a collision. Thus in terms of number of iterations, the computational complexity is given by

$$T_{\mathsf{complexity}} = 2^8 \cdot (190 \cdot \alpha \cdot p_{22}) \cdot 2^{32}/\sqrt{\alpha} \approx 2^{78.7} \cdot \sqrt{\alpha}.$$

The number of hash insertions is $T_{\mathsf{insertions}} = 2 \cdot 23 \cdot \frac{2^{32}}{\sqrt{\alpha}} \cdot 2^8 \approx 2^{45.5} \cdot \alpha^{-1/2}$. For $\ell = 32$ and $\alpha = 2$, we get $T_{\mathsf{complexity}} = 2^{79.2}$, $T_{\mathsf{insertions}} = 2^{45}$, i.e. the computational complexity is slightly lower and the table access complexity is significantly lower than the corresponding figures of the generic state-recovery detailed in Appendix E.3.2. However when $\ell = 29$, $\alpha = 64$, we get $T_{\mathsf{complexity}} = 2^{81.7}$, $T_{\mathsf{insertions}} = 2^{42.5}$ which is worse than state recovery complexity. This is another justification for the length of the IV to be limited to well below 32 bits.

Once we establish that for some known value of $(e_0, \ldots, e_7) = b^*$, the session key is in a short period orbit p_i for $i \leq 22$, we have effectively reduced the entropy of the master key to $\log_2 p_i + 40 < 71.17$ bits. The remaining key can be found by exhaustive search. From Appendix E.3.1, we know that this should take less than $2^{71.17+5.67} \approx 2^{76.84}$ iterations. So the total computational complexity if $\ell = 32$ is still $2^{79.2}$ which is lower than the complexity of generic state recovery, and results in a significantly smaller table-access complexity.

With probability around $1 - 0.751 \approx 0.249$ we will not obtain a collision in any one of the 256 cases listed above. If so we reduced the last 4 bytes of the short-key to a set of cardinality S^c which is just lower than 2^{30}, and the entire short-key to 2^{38}. Thus the remaining entropy of the key is around 78 bits, which can still be exhausted in complexity approximately $2^{83.67}$.

Remark 8. The high complexity comes from the fact that the decimation factor 19 is co-prime with 10. If instead the decimation factor had been 30, which is still larger than 19, the attack complexity would have a factor $30p_{22}$ instead of $190p_{22}$ which is around 6 times smaller. Thus the decimation factor being co-prime with 10 is necessary for the security.

6.2 Linear Cryptanalysis

For a well-designed stream cipher, there should be no linear approximations with high absolute correlation between the inputs (in the case of TEA-3, these are the 29-bit IV and the 32-bit tweak) and the keystream. The overall methodology of linear cryptanalysis is identical for stream- and block ciphers, although for historical reasons different terminology is sometimes used. For example, correlation attacks are covered as a special case of the analysis below. The main difference is that, for stream ciphers, there are linear approximations with all-zero input masks – and these are particularly important.

To emphasize the influence that the decomposition discussed in Section 4 can have on linear cryptanalysis, for a moment, consider a modified TEA-3 with a *linear* state register.

In this setting, a guess-and-determine key-recovery attack with complexity roughly 2^{40} TEA-3 evaluations (computing 10 key stream bytes each) becomes easily possible. Consider Figure 2 again. The idea of the attack is to guess the 40-bit state $r = (r_0, \ldots, r_4)$ and then uniquely determine $a = (a_0, \ldots, a_4)$ from the output keystream. We can do so rather easily as, after guessing the initialization of h, only *linear* components remain.

In regards to the original TEA-3, we argue that the most interesting scenarios for linear cryptanalysis of TEA-3 are the following two extreme cases: (i) targeting only the initialization (output mask nonzero only for the first keystream byte), or (ii) using consecutive keystream bytes (zero masks on the inputs). Due to the relatively heavy initialization phase of TEA-3 (key and state registers are clocked 32 times for mixing and an additional 19 times to output the first keystream byte), targeting the initialization phase seems less promising. However, the constraints on the masks of the linear approximations are more flexible. Also, because the tweak is injected through the key register, it is possible to skip a few initialization rounds by fixing the IV and part of the tweak. However, our analysis indicates that this does not lead to better attacks. Hence, we focus on scenario(ii).

For linear approximations with masks nonzero only on consecutive keystream bytes, we first analyze the construction with a fixed tweak. This is a natural starting point, since in this case the key-update function imposes no additional constraints on the masks of linear trails through the state-update function. By solving a (simplied) Satisfiability Modulo Theories model of TEA-3, we found a linear trail between two consecutive keystream bytes (19 iterations of the state-update function) with correlation $\pm 2^{-32}$. This trail is depicted in Figure 5a in Appendix E.1. For the same linear approximation as in Figure 5a, there exist four linear trails with correlation $\pm 2^{-35}$ (see Figure 5b). These trails have the same key-dependency and always cancel each other. In addition to the trails in Figure 5, we found several trails (for different approximations) with correlation $\pm 2^{-34}$ and $\pm 2^{-35}$.

Based on the above, one might be tempted to conclude that there are linear approximations between consecutive keystream bytes with correlation $\pm 2^{-32}$. However, this is incorrect because the IV only consists of 29 bits, so the initial state is not uniform random. This will lead to an estimation error (due to trails with nonzero masks on the initial state) on the order of $\pm 2^{-14.5}$. Heuristically, multiple linear cryptanalysis based on the approximations with correlation $c = \pm 2^{-32}$ is still feasible, but the data-complexity would be around $2^{99} = 2^{29} \cdot 2^{70}$ so that $q\sqrt{l} = 1/c^2$ with $q = 2^{29}$ the number of IVs and $l = 2^{70}$ the number of keystream bytes for each IV.

The data-complexity may be reduced to $2^{67} = 2^{29+32} \cdot 2^6$ if multiple tweaks are used. However, this would require guessing 40 bits of the key and modifying the test-statistic based on the output of the nonlinear part of the key-update function. This can be done in roughly 2^{80} table accesses as follows. First, distill the data into a table of length 2^{32+6} indexed by 32-bit tweaks and six bits indicating the linear approximation. For every 40 bit value of the xor between the two halves of the 80 bit key (so the initial state of the nonlinear part of the key-update function can be computed for a given tweak), compute a standard test statistic for multiple linear cryptanalysis (such as the sum of squares). This requires $2^{40} \cdot 2^{32+6} = 2^{78}$ table reads and a similar number of arithmetic operations. In practice, the memory accesses are likely to dominate. It is not clear if the computational cost of this approach can be brought significantly below 2^{80} evaluations of TEA-3.

It is also worth looking at TEA-3 with a smaller decimation factor, r, so that $r = 19$ for TEA-3. In Appendix E.2, it is shown that for $r \leq 8$ there are no linear trails with nonzero correlation – assuming that the mask on the state before the extraction of the first keystream byte is zero.

6.3 Generic Attacks

These attacks are applicable to TEA-{2,4} as well.

6.3.1 Attack on Ciphers Xoring Key and Tweak

It is well-known that xoring a tweak to the key of a secure block cipher does not yield a secure tweakable block cipher in general [LRW02]. Indeed, there is a generic time-data-memory trade-off attack [DH77] that, e.g., was recently described in the context of the tweakable block cipher HALFLOOP [DDLS22, LRS23]. Essentially the same attack can be applied to the stream cipher TEA-3. However, here the 32-bit tweak is expanded to 80 bits using $\texttt{expand}_\texttt{K}$ which acts as the identity on the first 32-bits. Hence, we only vary the last 48 bits of the key. More precisely, for a fixed initialization vector $\texttt{IV}^*$, in an offline phase we compute a table

$$T = \left[\left(0^{32}||k, \texttt{TEA-3}_{0^{32}||k}(\texttt{IV}^*, 0, 10)\right) \text{ for } k \in \mathbb{F}_2^{48}\right]$$

and sort it by the second component. In the online phase, we query $\texttt{TEA-3}_{k^*}(\texttt{IV}^*, t, 10)$ for *every* tweak $t \in \mathbb{F}_2^{32}$ where k^* is the secret key we wish to recover. Using binary search, we check if there exists a tuple (k, t) such that $\texttt{TEA-3}_{0^{32}||k}(\texttt{IV}^*, 0, 10) = \texttt{TEA-3}_{k^*}(\texttt{IV}^*, t, 10)$. If so, we know that (ignoring one expected false positive) $k^* = \texttt{expand}_\texttt{K}(t) \oplus (0^{32}||k)$.

This attack has an offline complexity of 2^{48} TEA-3 encryptions (plus sorting the table) and roughly $2^{48} \cdot 16$ bytes = 4 PB (petabytes) of memory. For the online complexity, we need 2^{32} queries with *chosen* IV and tweak and as many binary searches in the table. At least for passive attackers, we do *not* consider this attack practical, simply because the need of *chosen* data is too high, especially considering that the key should be changed before the IV, which corresponds to a time stamp, rolls over.

6.3.2 A TMTO Attack for a Fixed IV

We describe a TMTO attack, with a practical online and query complexity that requires 1 PB more of memory than the previous attack. We follow Hellman's approach and assume that the reader is familiar with TMTO attacks. [Hel06] is an excellent reference for those who need help. Other approaches as e.g. the one by Oechslin [Oec03] might be equally appropriate. Again we fix an initial value $\texttt{IV}^*$. As we recover the ECK directly we do not have to deal with the tweak except after recovery to compute the CK or SCK from it respectively. Hence we adjust the notation a bit and denote with $\texttt{TEA-3}_k(\texttt{IV}^*, 10)$ the first 10 bytes after the initialization phase of TEA-3 with initial value $\texttt{IV}^*$. We consider keys of the form $0^8||k, k \in \mathbb{F}_2^{72}$, i.e. elements from the subspace $K_{72} := \{0\}^8 \times \mathbb{F}_2^{72}$. According to [GMAM16, Table 1] we build $2^{\frac{72}{3}} = 2^{24}$ tables with 2^{24} rows and 2^{24} columns to get a coverage of 80%. Thereby the starting points for each table T_l are chosen uniformly at random from K_{72}. Then $\texttt{TEA-3}_k(\texttt{IV}^*, 10)$ is applied, i.e. the transition function $f : K_{72} \to K_{72}$ for the l-th table is as follows. From the current entry the 10-byte output stream $\texttt{TEA-3}_k(\texttt{IV}^*, 10)$ is considered as an element $k \in K_{72}$ by setting the first 8 bits to zero. Then $\texttt{TEA-3}_k(\texttt{IV}^*, 10)$ is applied which gives the next output stream. The end points are stored and sorted as well. The transition function is over $\mathbb{F}_2^{80}$ as 10-byte output stream has to be compared to prevent false positives on average. With a probability of $\frac{1}{2^8}$ we have that the tweaked key $k^* = \texttt{expand}_\texttt{K}(t) \oplus k$ lies in K_{72}. Taking into account that the coverage is 80%, we get that only $2^8 \cdot 1.25$ queries are required on average to expect that k^* is contained in one of the T_l. So with a complexity of $2^8 \cdot 1.25 \cdot 2^{48} \approx 2^{56.3}$ the key k^* and thus k can be recovered.

To study the practicality of this attack, we roughly compute its complexity and cost. If for all $2^8 \cdot 1.25 = 320$ queries the search is done in parallel, the complexity is 2^{48} and the key can be recovered within 3 days on average. From Section 2.3 it is known that in Class-2 networks the CK tends to be used for along time. Hence if one sets up a fake base station or wiretaps 320 base stations, a TMTO attack based on a fixed IV should be successful as then it can be expected that there is a device where the ECK lies in K_{72}. From this, CK can be computed and then used for a long time. In the latter attack scenario it might take up to a month to be successful and it is less realistic. To build the tables T_l

we have to go through 2^{72} states. It is possible to generate $2^{29.10}$ keystreams of 10 bytes in a second [Tez]. Hence it is possible to generate 2^{55} entries in a year with one GPU. So with 2^{18} GPUs the corresponding tables can be built within a year. Due to the amount of GPUs required an optimistic assumption is that each of them costs 100 €. Hence for 70,000,000 € these tables can be built. The required memory is $19 \cdot 2^{48}$ bytes ≈ 5 PB. These costs are estimated with 10,000,000 €. So in total such an attack costs 80,000,000 € and might then be considered as somewhere between purely academic and practical.

7 Discussion

We presented a first public and in-depth cryptanalysis of TEA-3. Although we identified some rather unconventional and arguably unsound design choices, most importantly the decomposing key register, we were not able to give a practical attack. Nevertheless, we advise using for instance AES instead of TEA-3 and TEA-2. AES is more secure (higher key length) and better studied. Moreover, even though we did not study them in detail, we also advise against using TEA-{1,4} (as their effective key size is 32 or 56 bit respectively). And, again without an in-depth analysis, we also recommend using AES instead of TEA-{5,6,7}. While Rijndael in counter mode is a good step, modifying it by adding an obscure function to derive the key for it, which in TEA-7 is used to intentionally weaken the design [ETS24k], is questionable at best. Still, we believe that studying all these designs to unveil the design choices made without an open evaluation process is thrilling future work.

References

[BDL+21] Christof Beierle, Patrick Derbez, Gregor Leander, Gaëtan Leurent, Håvard Raddum, Yann Rotella, David Rupprecht, and Lukas Stennes. Cryptanalysis of the GPRS encryption algorithms GEA-1 and GEA-2. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 155–183. Springer, 2021.

[Bok] Wouter Bokslag. Private communication.

[BSW00] Alex Biryukov, Adi Shamir, and David A. Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[DDLS22] Marcus Dansarie, Patrick Derbez, Gregor Leander, and Lukas Stennes. Breaking HALFLOOP-24. *IACR Trans. Symmetric Cryptol.*, 2022(3):217–238, 2022.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.

[ETS24a] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 1: TEA1. https://www.etsi.org/deliver/etsi_TS/101000_101099/10105301/02.01.01_60/ts_10105301v020101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24b] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 2: TEA2. https://www.etsi.org/deliver/etsi_TS/101000_101099/10105302/02.05.01_60/ts_10105302v020501p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24c] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 3: TEA3. https://www.etsi.org/deliver/etsi_ts/101000_101099/10105303/02.01.01_60/ts_10105303v020101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24d] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 4: TEA4. https://www.etsi.org/deliver/etsi_ts/101000_101099/10105304/02.01.01_60/ts_10105304v020101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24e] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 5: TEA5. https://www.etsi.org/deliver/etsi_ts/101000_101099/10105305/01.01.01_60/ts_10105305v010101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24f] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 6: TEA6. https://www.etsi.org/deliver/etsi_ts/101000_101099/10105306/01.01.01_60/ts_10105306v010101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24g] ETSI. Rules for the management of the TETRA standard encryption algorithms; Part 7: TEA7. https://www.etsi.org/deliver/etsi_ts/101000_101099/10105307/01.01.01_60/ts_10105307v010101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24h] ETSI. TETRA Air Interface Security, Algorithms Specification; Part 2: TETRA Encryption Algorithms TEA Set B. https://www.etsi.org/deliver/etsi_ts/104000_104099/10405302/01.01.01_60/ts_10405302v010101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24i] ETSI. TETRA Air Interface Security, Algorithms specification; Part 3: TETRA and Authentication and Key Management Algorithms TAA1. https://www.etsi.org/deliver/etsi_ts/104000_104099/10405303/01.01.01_60/ts_10405303v010101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24j] ETSI. TETRA Air Interface Security, Algorithms Specifications; Part 1: TETRA Encryption Algorithms Set A. https://www.etsi.org/deliver/etsi_ts/104000_104099/10405301/01.01.01_60/ts_10405301v010101p.pdf, 2024. [Online; accessed 11-November-2024].

[ETS24k] ETSI. Tetra security disclosures. https://tcca.info/documents/Research-Disclosures-v2-301023.pdf, 2024. [Online; accessed 11-November-2024].

[GD70] D.H. Green and K.R. Dimond. Nonlinear product-feedback shift registers. *Proceedings of the Institution of Electrical Engineers*, 117:681–686, 1970.

[GMAM16] Nasser Gharavi, Abdolrasoul Mirghadri, Mohammad Abdollahi Azgomi, and Ahmad Mousavi. Expected coverage of perfect chains in the Hellman time memory trade-off. *Journal of Computing and Security*, 3:155–162, 07 2016.

[Gol81] Solomon W. Golomb. *Shift Register Sequences*. Aegean Park Press, USA, 1981.

[Hel06] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theor.*, 26(4):401–406, sep 2006.

[LRS23] Gregor Leander, Shahram Rasoolzadeh, and Lukas Stennes. Cryptanalysis of HALFLOOP block ciphers destroying HALFLOOP-24. *IACR Trans. Symmetric Cryptol.*, 2023(4):58–82, 2023.

[LRW02] Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[MBW23] Carlo Meijer, Wouter Bokslag, and Jos Wetzels. All cops are broadcasting: TETRA under scrutiny. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 7463–7479, 2023.

[MST79] Johannes Mykkeltveit, Man-Keung Siu, and Po Tong. On the cycle structure of some nonlinear shift register sequences. *Information and control*, 43(2):202–215, 1979.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 617–630, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Tez] Cihangir Tezcan. Private communication.

[WAS24] The Wassenaar Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies. https://www.wassenaar.org/, 2024. [Online; accessed 11-November-2024].

[Zet24] Kim Zetter. Interview with the ETSI Standards Organization That Created TETRA "Backdoor". https://www.zetter-zeroday.com/interview-with-the-etsi-standards/, 2024. [Online; accessed 11-November-2024].

A A C-program to Find the Sequences of the Decomposed Key Register

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

/*
    This program, as is, will find all cycles in the decomposed
    key register of the TEA-3 cipher and print their periods.
    We do this by marking every state we visit in a table. We get
    a state belonging to an unknown cycle by consulting the table.
    The full table requires 2^40 bits corresponding to 128 GiB.
    In order to be able to run this program om machines with
    less than 128 GiB of RAM, we only consider a subset of the
    states controlled by a "subset key". With a subset key size of
    4 bits the table only requires 8 GiB - at the expense of
    possibly missing some cycles. If any cycles are missed,
    the program can be rerun with another value for the subset key.

    Compile with gcc -O3 filename.c -o find_cycles
    Takes ~3 hours to complete on a typical laptop.
*/

uint8_t tea3_sbox[256] = {
    0x7d, 0xbf, 0x7b, 0x92, 0xae, 0x7c, 0xf2, 0x10,
    0x5a, 0x0f, 0x61, 0x7a, 0x98, 0x76, 0x07, 0x64,
    0xee, 0x89, 0xf7, 0xba, 0xc2, 0x02, 0x0d, 0xe8,
    0x56, 0x2e, 0xca, 0x58, 0xc0, 0xfa, 0x2a, 0x01,
    0x57, 0x6e, 0x3f, 0x4b, 0x9c, 0xda, 0xa6, 0x5b,
    0x41, 0x26, 0x50, 0x24, 0x3e, 0xf8, 0x0a, 0x86,
    0xb6, 0x5c, 0x34, 0xe9, 0x06, 0x88, 0x1f, 0x39,
    0x33, 0xdf, 0xd9, 0x78, 0xd8, 0xa8, 0x51, 0xb2,
    0x09, 0xcd, 0xa1, 0xdd, 0x8e, 0x62, 0x69, 0x4d,
    0x23, 0x2b, 0xa9, 0xe1, 0x53, 0x94, 0x90, 0x1e,
    0xb4, 0x3b, 0xf9, 0x4e, 0x36, 0xfe, 0xb5, 0xd1,
    0xa2, 0x8d, 0x66, 0xce, 0xb7, 0xc4, 0x60, 0xed,
    0x96, 0x4f, 0x31, 0x79, 0x35, 0xeb, 0x8f, 0xbb,
    0x54, 0x14, 0xcb, 0xde, 0x6b, 0x2d, 0x19, 0x82,
    0x80, 0xac, 0x17, 0x05, 0xff, 0xa4, 0xcf, 0xc6,
    0x6f, 0x65, 0xe6, 0x74, 0xc8, 0x93, 0xf4, 0x7e,
    0xf3, 0x43, 0x9f, 0x71, 0xab, 0x9a, 0x0b, 0x87,
    0x55, 0x70, 0x0c, 0xad, 0xcc, 0xa5, 0x44, 0xe7,
    0x46, 0x45, 0x03, 0x30, 0x1a, 0xea, 0x67, 0x99,
    0xdb, 0x4a, 0x42, 0xd7, 0xaa, 0xe4, 0xc2, 0xd5,
    0xf0, 0x77, 0x20, 0xc3, 0x3c, 0x16, 0xb9, 0xe2,
    0xef, 0x6c, 0x3d, 0x1b, 0x22, 0x84, 0x2f, 0x81,
    0x1d, 0xb1, 0x3a, 0xe5, 0x73, 0x40, 0xd0, 0x18,
    0xc7, 0x6a, 0x9e, 0x91, 0x48, 0x27, 0x95, 0x72,
    0x68, 0x0e, 0x00, 0xfc, 0xc5, 0x5f, 0xf1, 0xf5,
    0x38, 0x11, 0x7f, 0xe3, 0x5e, 0x13, 0xaf, 0x37,
    0xe0, 0x8a, 0x49, 0x1c, 0x21, 0x47, 0xd4, 0xdc,
    0xb0, 0xec, 0x83, 0x28, 0xb8, 0xf6, 0xa7, 0xc9,
    0x63, 0x59, 0xbd, 0x32, 0x85, 0x08, 0xbe, 0xd3,
    0xfd, 0x4c, 0x2c, 0xfb, 0xa0, 0xc1, 0x9d, 0xb3,
    0x52, 0x8c, 0x5d, 0x29, 0x6d, 0x04, 0xbc, 0x25,
```

```
    0x15, 0x8b, 0x12, 0x9b, 0xd6, 0x75, 0xa3, 0x97
};

#define SUBSET_KEY 0ULL
#define SUBSET_KEY_SIZE 4
#define SR_WIDTH 40
#define TABLE_SIZE (1ULL << (SR_WIDTH - SUBSET_KEY_SIZE - 6))
#define MASK_FULL ((1ULL << SR_WIDTH) - 1)
#define MASK_PARTIAL ((1ULL << (SR_WIDTH - SUBSET_KEY_SIZE)) - 1)

uint64_t *visited_states;

// Initial states that we know belong to short cycles
// these could be missed due to the subset_key part of the code
uint64_t known_short[] = {0xc2c2c2c2c2, 0x8181818181};
uint64_t num_short_cycles = sizeof(known_short) /
    sizeof(known_short[0]);

// Finds the period of the cycle containing init_val
uint64_t map_cycle(uint64_t init_val) {
    init_val = init_val & MASK_FULL; \
    uint64_t state = init_val & MASK_FULL;
    uint64_t fb;
    uint64_t cycle_length = 0;
    uint64_t state_key;

    while (1) {
        if ((cycle_length & 0xfffffff) == 0) { // Print progress
            printf("\rNumber of visisted states in current cycle:
                0x%010lx", cycle_length);
            fflush(stdout);
        }
        cycle_length++;
        // Mark the state as visited
        state_key = state >> (SR_WIDTH - SUBSET_KEY_SIZE);
        if (state_key == SUBSET_KEY) {
            uint64_t partial_state = state & MASK_PARTIAL;
            uint64_t index = (partial_state >> 6);
            uint64_t bitmask = 1ULL << (partial_state & 0x3f);
            if (visited_states[index] & bitmask) {
                break; // We have been here before, break
            }
            visited_states[index] |= bitmask;
        }
        // Calculate feedback and advance the shift register
        fb = tea3_sbox[(state >> 16) & 0xff];
        fb ^= state & 0xff;
        state = (state >> 8) | (fb << (SR_WIDTH - 8));
        // Break if we have completed the cycle
        if (state == init_val) {
            break;
        }
    }
    return cycle_length;
}

// Returns the first unvisited state in the table
```

```
109 uint64_t get_unvisited_state(uint64_t *unvisited_state) {
110     *unvisited_state = SUBSET_KEY << (SR_WIDTH - SUBSET_KEY_SIZE);
111     for (uint64_t i = 0; i < TABLE_SIZE; i++) {
112         if (visited_states[i] != 0xffffffffffffffff) {
113             for (uint64_t j = 0; j < 64; j++) {
114                 if (!(visited_states[i] & (1ULL << j))) {
115                     *unvisited_state |= (i << 6) | j;
116                     *unvisited_state &= MASK_FULL;
117                     return 1;
118                 }
119             }
120         }
121     }
122     return 0;
123 }
124
125 int main() {
126     // Allocate memory for the states table. Calloc initializes to 0
127     visited_states = calloc(TABLE_SIZE, sizeof(uint64_t));
128     if (visited_states == NULL) {
129         printf("Memory allocation failed\n");
130         return -1;
131     }
132     uint64_t total_states_mapped = 0;
133     uint64_t num_found_cycles = 0;
134
135     while (1) {
136         uint64_t unvisited_state;
137         if (num_found_cycles < num_short_cycles) {
138             unvisited_state = known_short[num_found_cycles];
139             num_found_cycles++;
140         } else if (!get_unvisited_state(&unvisited_state)) {
141             break;
142         }
143         printf("\nMapping the cycle containing state 0x%010lx\n",
                unvisited_state);
144         uint64_t cycle_length = map_cycle(unvisited_state);
145         total_states_mapped += cycle_length;
146         // Print progress
147         printf("\nCycle found with period %lu\n", cycle_length);
148         printf("Number of covered states: 0x%010lx / 0x%010llx\n",
                total_states_mapped, MASK_FULL + 1);
149     }
150
151     free(visited_states); // Cleanup
152     return 0;
153 }
```

B Algebraic Normal Forms of F_{31} and F_{32} and of the Equivalent Functions Studied in Section 5

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

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

The functions G_{31} and G_{32} derived from F_{31} and F_{32} by composing the input with the bijection $(x, y) \mapsto (x, x \oplus y)$ and by reordering the input variables by $(x_0, \dots, x_7, y_0, \dots, y_7) \mapsto (x_3, \dots, x_2, y_3, \dots, y_2)$ have the following coordinates:

$$
\begin{aligned}
G_{31}(x_0,\dots,y_7)_0 &= x_1 \oplus y_0 \oplus x_0y_0 \oplus y_1 \oplus x_0y_1 \oplus x_0x_1y_1 \oplus y_0y_1 \oplus x_0y_0y_1 \oplus x_1y_0y_1\\
G_{31}(x_0,\dots,y_7)_1 &= 1 \oplus y_1 \oplus y_2 \oplus x_1y_2 \oplus x_2y_2 \oplus y_1y_2 \oplus x_1y_1y_2\\
G_{31}(x_0,\dots,y_7)_2 &= x_2 \oplus x_3 \oplus x_2y_2 \oplus x_3y_2 \oplus y_3 \oplus x_3y_3 \oplus x_3y_2y_3\\
G_{31}(x_0,\dots,y_7)_3 &= x_4y_3 \oplus y_4 \oplus x_3y_3y_4 \oplus x_4y_3y_4\\
G_{31}(x_0,\dots,y_7)_4 &= 1 \oplus y_4 \oplus x_4y_4 \oplus x_5y_4 \oplus y_5 \oplus y_4y_5 \oplus x_5y_4y_5\\
G_{31}(x_0,\dots,y_7)_5 &= 1 \oplus x_5 \oplus x_6 \oplus y_5 \oplus x_5y_5 \oplus x_6y_5 \oplus y_6 \oplus x_6y_6 \oplus x_6y_5y_6\\
G_{31}(x_0,\dots,y_7)_6 &= 1 \oplus x_6 \oplus x_6y_6 \oplus x_7y_7 \oplus y_6y_7 \oplus x_7y_6y_7\\
G_{31}(x_0,\dots,y_7)_7 &= 1 \oplus x_7 \oplus y_0 \oplus x_0y_7 \oplus x_7y_7 \oplus x_0x_7y_7 \oplus x_0y_0y_7 \oplus x_7y_0y_7
\end{aligned}
$$

$$
\begin{aligned}
G_{32}(x_0,\dots,y_7)_0 &= 1 \oplus x_1 \oplus x_0x_1 \oplus y_0 \oplus x_0y_0 \oplus x_1y_0 \oplus y_1 \oplus x_0x_1y_1 \oplus x_0y_0y_1 \oplus x_1y_0y_1\\
G_{32}(x_0,\dots,y_7)_1 &= 1 \oplus y_1 \oplus x_2y_2 \oplus x_1y_1y_2\\
G_{32}(x_0,\dots,y_7)_2 &= x_3 \oplus x_3y_2 \oplus y_3 \oplus x_2y_3 \oplus x_3y_3 \oplus x_2y_2y_3 \oplus x_3y_2y_3\\
G_{32}(x_0,\dots,y_7)_3 &= x_4y_3 \oplus y_4 \oplus y_3y_4 \oplus x_3y_3y_4 \oplus x_4y_3y_4\\
G_{32}(x_0,\dots,y_7)_4 &= x_4y_4 \oplus y_5 \oplus y_4y_5 \oplus x_4y_4y_5 \oplus x_5y_4y_5\\
G_{32}(x_0,\dots,y_7)_5 &= x_5 \oplus x_6 \oplus x_5y_5 \oplus x_6y_5 \oplus y_6 \oplus x_6y_6 \oplus x_6y_5y_6\\
G_{32}(x_0,\dots,y_7)_6 &= y_6 \oplus x_6y_6 \oplus x_7y_6 \oplus y_7 \oplus y_6y_7 \oplus x_7y_6y_7\\
G_{32}(x_0,\dots,y_7)_7 &= x_0x_7 \oplus y_0 \oplus x_7y_0 \oplus y_7 \oplus x_0y_7 \oplus x_7y_7 \oplus x_0x_7y_7 \oplus x_0y_0y_7 \oplus x_7y_0y_7
\end{aligned}
$$

Then, it can be easily checked that these functions satisfy the construction described in Theorem 6. For instance, focusing on G_{31}, the coordinate of index 1 has the form (3). Indeed,

$$G_{31,1} = \begin{cases} y_2(x_1 \oplus x_2 \oplus 1) \oplus 1 & \text{if } y_1 = 0 \\ y_2 x_2 & \text{if } y_1 = 1 \end{cases}$$

and the next 5 coordinates then follow Construction (1)':

$$\begin{aligned} G_{31,2} &= \begin{cases} y_3 & \text{if } y_2 = 1 \\ y_3(x_3 \oplus 1) \oplus x_2 \oplus x_3 & \text{if } y_2 = 0 \end{cases} \\ G_{31,3} &= \begin{cases} y_4 & \text{if } y_3 = 0 \\ y_4(x_3 \oplus x_4 \oplus 1) \oplus x_4 & \text{if } y_3 = 1 \end{cases} \\ G_{31,4} &= \begin{cases} y_5 \oplus 1 & \text{if } y_4 = 0 \\ y_5 x_5 \oplus x_4 \oplus x_5 & \text{if } y_4 = 1 \end{cases} \\ G_{31,5} &= \begin{cases} y_6 & \text{if } y_5 = 1 \\ y_6(x_6 \oplus 1) \oplus x_5 \oplus x_6 \oplus 1 & \text{if } y_5 = 0 \end{cases} \\ G_{31,6} &= \begin{cases} y_7 \oplus 1 & \text{if } y_6 = 1 \\ y_7 x_7 \oplus x_6 \oplus 1 & \text{if } y_6 = 0 \end{cases} \end{aligned}$$

C Proof of Proposition 1

We will start by proving that any sum of k consecutive coordinates, ranging from u to $u+k-1$, is balanced. Let $g_{u,k}$ where $1 \le u < u+k \le n$ denote the Boolean function of $2(k+1)$ variables defined by

$$g_{u,k}(x_u, \ldots, x_{u+k}, y_u, \ldots, y_{u+k}) = \sum_{i=u}^{u+k-1} f_i(x_i, x_{i+1}, y_i, y_{i+1}) \ .$$

We can prove by induction on $k \ge 1$ that $g_{u,k}$ is balanced and the restriction of $(g_{u,k} \oplus x_{u+k})$ to $y_{u+k} = 1 \oplus \varepsilon_{u+k}$ is balanced.

- For $k = 1$, we only have to check that each function f_u is balanced, which comes from the fact that its restrictions to $y_u = \varepsilon_u$ and to $y_u = 1 \oplus \varepsilon_u$ are both balanced. Indeed, ℓ_u^0 and $(\ell_u^0 \oplus \ell_u^1)$ both have degree 1. Moreover, the restriction of $(f_u \oplus x_{u+1})$ to $y_{u+1} = 1 \oplus \varepsilon_{u+1}$ is given by

$$\begin{cases} x_{u+1} \oplus 1 \oplus \varepsilon_{u+1} \oplus c_u & \text{if } y_u = \varepsilon_u \\ ((1 \oplus \varepsilon_{u+1})\ell_u^1 \oplus \ell_u^0)(x_u, x_{u+1}) \oplus x_{u+1} & \text{if } y_u = 1 \oplus \varepsilon_u \ . \end{cases}$$

  It is then balanced on the set of inputs with $y_u = \varepsilon_u$. Moreover, it is also balanced on $y_u = 1 \oplus \varepsilon_u$, because ε_{u+1} satisfies $\deg(\ell_u^0 \oplus \ell_u^1 \oplus \varepsilon_{u+1}\ell_u^1 \oplus x_{u+1}) = 1$ for all $1 \le u < n$.

- The induction step is derived from Lemma 1. Indeed, by induction hypothesis, $g_{u,k}$ is balanced and the restriction of $(g_{u,k} \oplus x_{u+k})$ to $y_{u+k} = 1 \oplus \varepsilon_{u+k}$ is balanced. Then Lemma 1 with $\varepsilon = \varepsilon_{u+k}$ implies that $g_{u,k+1}$ is balanced and the restriction of $g_{u,k+1} \oplus x_{u+k+1}$ to $y_{u+k+1} = 1 \oplus \varepsilon_{u+k+1}$ is balanced because ε_{u+k+1} has been chosen such that $\deg(\ell_{u+k}^0 \oplus \ell_{u+k}^1 \oplus \varepsilon_{u+k+1}\ell_{u+k}^1 \oplus x_{u+k+1}) = 1$.

Now, the vectorial function $F^{(n)}$ is balanced if and only if its components $\gamma \cdot F^{(n)}$ are balanced for all nonzero $\gamma \in \mathbb{F}_2^n$. If $\{i : \gamma_i = 1\}$ consists of consecutive integers, then $\gamma \cdot F^{(n)}$ equals some $g_{u,k}$ and is therefore balanced. Otherwise, $\{i : \gamma_i = 1\}$ can be decomposed into $I_1 \cup I_2 \ldots \cup I_v$ where the sets I_j are disjoint sets of consecutive integers that satisfy $\max I_j < \min I_{j+1} + 1$ for $1 \leq j < v$. It follows that $\gamma \cdot F^{(n)}$ is a sum of v functions g_{u_j,k_j} with distinct input variables. Since all g_{u_j,k_j} are balanced and are in direct sum, $\gamma \cdot F^{(n)}$ is balanced. □

D Equivalent Representations

For completeness, we briefly describe some equivalent ways of representing TEA-3.

Key Register Output First, notice that instead of taking the feedback value of the key register as an input into the state register, we can also take the output (the leftmost byte of the key register) as the input into the state register. Up to initialization, this is equivalent. The corresponding initialization is clocking the key register 10 times (with the state register unchanged).

Fibonacci-like Representations While the state register is originally presented in a Galois-like representation, we can change this to an (up to initialization) equivalent Fibonacci-like representation. That is, there is a feedback function $H_1 \colon (\mathbb{F}_2^8)^8 \to \mathbb{F}_2^8$ that computes a single feedback byte that is fed into the right most byte of the state register. Clearly, the initial state of this representation must be the first eight bytes of the output stream itself. The feedback function H, which we deduced by symbolically clocking the state register, is

$$H_1(x_0, \ldots, x_7) = x_0 \oplus R(F_{31}(x_1, x_2) \oplus x_3) \oplus F_{32}(F_{31}(x_3, x_4) \oplus x_5, F_{31}(x_4, x_5) \oplus x_6) \oplus F_{31}(x_6, x_7)$$

where x_0 is the leftmost byte in the Fibonacci-like representation. Furthermore, we can apply the same idea to the state and key register *jointly* (or the state and the linear part of the decomposed key register). That is, there is (up to initialization) an equivalent Fibonacci-like representation of TEA-3 that consists of a single 18-byte register with a feedback function $H_2 \colon (\mathbb{F}_2^8)^{18} \to \mathbb{F}_2^8$. Again, the initial state is simply given by the (undecimated) output stream of TEA-3. The feedback function is

$$\begin{aligned} H_2(x_0, \ldots, x_{17}) = &H_1(x_0, \ldots, x_7) \oplus x_8 \oplus H_1(x_{10}, \ldots, x_{17}) \\ &\oplus S(H_1(x_2, \ldots, x_9) \oplus x_{10} \oplus H_1(x_7, \ldots, x_{14}) \oplus x_{15}). \end{aligned}$$

Notice that in both cases, the leftmost byte x_0 is only used linearly. Hence, we can also interpret the state update of TEA-3 as a Feistel-like structure.

Further Decomposition As already mentioned in Remark 5 we can decompose the linear part of the decomposed key register even further. That is, in $\mathbb{F}_{2^n}[X]$ we have

$$(X^5 + 1) = (X^4 + X^3 + X^2 + X + 1)(X + 1) = \prod_{i=0}^{4} (X - \zeta^i)$$

and hence we can split the 5-byte register into a trivial one byte register feeding into a 4-byte register with all taps set, or vice versa. Alternatively, we can write it as the composition of 5 LFSRs with linear characteristic polynomials over $\mathbb{F}_{2^8}$. In all cases, the order does *not* matter. In the latter case, we have to consider bytes as elements of $\mathbb{F}_{2^n}$ via Φ_n.

Sum of Key Registers Finally, we note that instead of decomposing the key register, we can equivalently consider the key register where we initialize the five leftmost bytes to some fixed 40-bit value and then xor the output stream of this register with the output stream of the linear register from the decomposition of the key register which is initialized with an according 40-bit value. We can apply this idea also the aforementioned further decomposition.

E Cryptanalysis

E.1 Linear Cryptanalysis

This section contains supplementary material for Section 6.2.

E.1.1 Linear Trails

Linear trails with nonzero masks on consecutive keystream bytes are shown in Figure 5. The masks in this figure correspond to the masks on the bytes $x_0, x_1, \ldots, x_7$ (as in Figure 1 from left to right) in every round.

```
1. .. .. .. .. .. .. ..
.8 .8 2. .. .. .2 .. 1.
.8 .. .1 .. .. .2 1. .8
4. .1 .1 .. .. 1. .8 .8
.. .. .4 .. .8 1. .8 4.
.. .4 .. .8 1. .8 4. ..
.4 .2 .8 1. .8 4. .. ..
.2 .8 .. .8 41 8. .. .4
.8 .. .. 41 .. .. .4 .2
.. .8 4. .. .1 .4 .2 .8
.8 4. .. .1 .4 .2 .8 ..
4. .. .. .4 .. .8 .. .8
.. .. .. .. .. 1. .8 4.
.. .. .. .. 1. .8 4. ..
.. .. .. 1. .8 4. .. ..
.. .2 1. .8 4. .. .. ..
.. 1. .8 4. .. .. .. ..
.. .. 4. .. .. .. .. ..
.. 4. .. .. .. .. .. ..
4. .. .. .. .. .. .. ..
```

(a) Correlation $\pm 2^{-32}$.

```
1. .. .. .. .. .. .. ..
.8 .8 2. .. .4 .2 .. 1.
48 2. .1 .4 .. .2 1. .8
4. .1 .1 .. .. u8 .8 48
.. .. .4 .. .. .8 48 4.
.. .4 .. .. .8 48 4. ..
.4 .. .. .8 48 4. .. ..
.3 .. 18 48 4. 81 .. .4
.. 18 .. 4. .1 2. .4 .3
.8 1. 4. .1 2. .4 .3 ..
1. 4. .. 2. .4 .2 .. .8
4. .. .. .4 .. .. .8 1.
.. .. .. .. .. 1. 1. 4.
.. .. .. .. 1. 1. 4. ..
.. .. .. 1. 1. 4. .. ..
.. .v 1. 1. 4. .. .. ..
.. 1. 1. 4. .. .. .. ..
.. .. 4. .. .. .. .. ..
.. 4. .. .. .. .. .. ..
4. .. .. .. .. .. .. ..
```

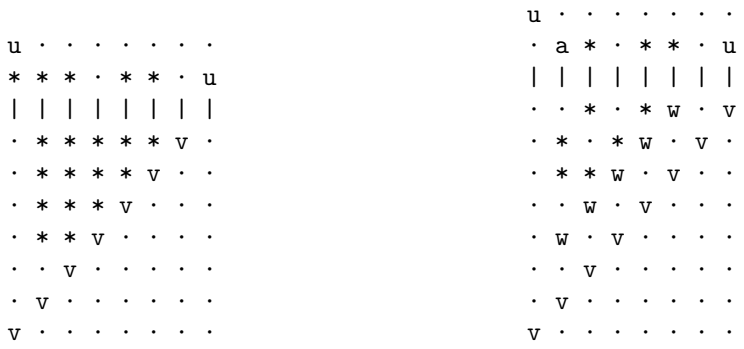
(b) Correlation $\pm 2^{-35}$.

Figure 5: Linear trails for consecutive keystream bytes, $u \in \{0, 1\}$ and $v \in \{2, 4\}$.

E.2 Reduced Number of Clocks

Let r denote the number of clocks of the state-update function between consecutive keystream bytes, so that $r = 19$ for TEA-3. A standard miss-in-the-middle argument shows that for $r \leq 7$, there are no linear trails with nonzero correlation – assuming that the mask on the state before the extraction of the first keystream byte is zero. The miss-in-the-middle argument is shown in Figure 6a, with u the input mask, v the output mask, zeros denoted by '$\cdot$' and undetermined values denoted by '$*$'. For $r = 8$, a little more

work is necessary to show that all such trails have correlation zero. If no assumptions on F_{31} and F_{32} are made, then one can see that all trails are of the form shown in Figure 6b. However, a contradiction is obtained when taking into account that if the output mask of these functions is nonzero, then the mask on at least one of the input bytes should be nonzero to obtain a nonzero correlation (due to balancedness). For $r = 9$ we found linear trails with correlation $\pm 2^{-26}$.



(a) $r = 7$ (b) $r = 8$

Figure 6: Structure of linear trails through the state-update function with $r \in \{8, 9\}$.

E.3 Generic Attacks

E.3.1 Key recovery

We first explicitly compute the complexity of exhaustive search in terms of the number of TEA-3 rounds executed by the adversary. This is an important metric to compare the feasibility of our attacks. To do an exhaustive search, first we guess the value of the key and then run an initialization phase for 51 rounds and generate keystream bytes corresponding to the guessed value of the key. We then compare the generated keystream byte with the keystream byte corresponding to the secret value of the key. If unequal, we discard the guess of the key, and restart with another guess, if not we generate the next keystream byte for comparison (this takes another 19 rounds). This way 255/256 of the key-guesses get eliminated after round 51 itself. Of the remaining 1/256 fraction, again 255/256 of the guesses get eliminated after $51 + 19 = 70$ rounds etc. Therefore on average the total number of TEA-3 rounds needed for the process is

$$\begin{aligned} T_{\mathsf{exhaustive}} &= \frac{255}{256} \cdot 2^{80} \cdot 51 + \frac{1 \cdot 255}{256 \cdot 256} \cdot 2^{80} \cdot (51 + 19) + \frac{1 \cdot 1 \cdot 255}{256 \cdot 256 \cdot 256} \cdot 2^{80} \cdot (51 + 2 \cdot 19) + \cdots \\ &= \frac{255}{256} \cdot 2^{80} \cdot \sum_{i=0}^{9} \left(\frac{1}{256}\right)^{i} \cdot (51 + 19i) \approx 2^{85.67} \end{aligned}$$

E.3.2 State recovery

The state size of the stream ciphers is less than twice the size of the key and hence the complexity of state recovery should be slightly better than that of exhaustive search. In the offline stage, we generate T randomly chosen internal states of the TEA-3 cipher, i.e. $S_i \xleftarrow{\$} \mathbb{F}_2^{144}$, for $i \in [1, T]$. We generate an 18-byte keystream sequence Z_i corresponding to each S_i and store it in a hash table indexed by S_i. Generating 18 bytes requires us to run TEA-3 for $18 \times 19 = 342$ iterations, which makes the offline complexity $T_{\mathsf{offline}} = 342T$ iterations. In general, the time to access a table could be considered to be equivalent to a typical cryptographic operation, since insertion in a large table also requires some

time/energy. Unless we have the technology or means to perform disk access in short enough time, we ought also take into consideration the number of table insertions/lookups needed as one of the costs of doing the attack. In this case we have $T_{\mathsf{insertions}} = T$, and the memory complexity is $M = 18T$ bytes.

In the online stage, we generate N keystream bytes from given secret key and IV pair. We can generate around $N - 17 \approx N$ keystream windows Y_i of 18 bytes each, and search for it in the precomputed table. If yes then recover the corresponding state S_i and clock back the cipher to the starting state to recover the key. Generating N bytes requires us to run TEA-3 for $N \times 19 = 19N$ iterations. After state is found another maximum $N \cdot 19$ reverse iterations to get the key, which makes the offline complexity $T_{\mathsf{offline}} = 38N$ iterations, and number of table-lookups is $T_{\mathsf{lookups}} = N$. Therefore the total time complexity equals

$$T_{\mathsf{final}} = 342T + 38N.$$

By Birthday considerations, the algorithm succeeds when $TN = 2^{144}$. T_{final} is minimized when $T = \frac{1}{3} \cdot 2^{72}$ and $N = 3 \cdot 2^{72}$. This makes $T_{\mathsf{final}} \approx 2^{79.83}$ iterations of TEA-3. We also have $T_{\mathsf{insertions}} = 2^{70.42}$ and $T_{\mathsf{lookups}} = 2^{73.58}$ and $M = 6 \cdot 2^{72}$ bytes.