Cryptanalysis of Full BEANIE

Xavier Bonnetain, Sébastien Duval, Virginie Lallemand, Thierno Mamoudou Sabaly, Thomas Sagot, and Thibault Sanvoisin

Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
firstname.name@loria.fr

Abstract. Beanie is a tweakable block cipher recently published at ToSC aiming for memory encryption of microcontroller units. In line with this goal, it handles small plaintexts of only 32 bits and has a low latency.
In this paper, we propose the first third-party analysis of the two variants of Beanie. By carefully leveraging structural properties of the cipher and taking advantage of its distinctive attack model, we manage to mount several key-recovery attacks against both the 5+5-round version and the 7+7-round version.

Keywords: BEANIE · Cryptanalysis · Yoyo · Truncated differential

1 Introduction

There has been a strong push in the last decade to design lightweight ciphers suited for various constrained applications. One such application is memory encryption for secure computing units. This scenario has strong latency constraints, and requires to encrypt very short values, typically between 10 and 32 bits. The primitive also needs to absorb a tweak, which can contain for example the address of the memory cell or a computing context, depending on the exact protocol.

As this is much smaller than the typical 64-bit block size of lightweight ciphers, it requires a dedicated design. Technically, we can trace back such designs to military standards for radio authentication, with SoDark and HALFLOOP [1,2]. They were not made by academic cryptographers, and once the cryptographic community was made aware of their specifications, they were quickly completely broken [14,15,24].

One of the first modern academic attempts was SCARF [11], which encrypts 10-bit plaintexts with a 240-bit key and a 48-bit tweak. It was broken by a sort of related-tweak differential cryptanalysis [16]. Another attempt was BipBip [7], which has a 24-bit block size and uses a 40-bit tweak and 256-bit key. It was broken with a Demirci-Selçuk MITM attack [28]. More recently, the Chilow family of tweakable block ciphers was proposed [8], specifying ciphers with 32 or 40-bit blocks, a 64-bit tweak and a 128-bit key. It has been analysed in several cryptanalysis articles [25,12] and key-recovery attacks against the full versions were presented in [22].

BEANIE [17] is a tweakable SPN block cipher designed by a team of authors from TU Graz and NXP Semiconductors with the objective of being fitted to hardware-based memory encryption for microcontroller units. To be suitable in such an environment, the cipher takes as input 32-bit plaintexts and has a competitive low latency. It uses a 128-bit master key and a 128-bit tweak.

BEANIE has several distinctive features: first, its aimed use case implies an attacker cannot access the encryption of a given plaintext (as it would require to mount a physical attack, which the authors claim is very improbable). However, a more reasonable scenario is the one where the attacker tries to read part of the internal memory they should not have access to (by overcoming logical isolation). This setting corresponds to the so-called U-shape attack setting, where an attacker can observe $E^{-1}_{K,T^1}(E_{K,T^0}(P^0))$ for a chosen plaintext P^0, and a pair of tweaks T^0, T^1.

The second specificity comes from how the round tweak-keys are produced. BEANIE does not employ an iterative tweak-key schedule (where key material is produced after each round), but uses a complex process which outputs a value seen as the concatenation of the 4 first round tweak-keys.

Technical overview of our contributions. In this article, we propose the first third-party analysis of BEANIE and the first full-round attacks of its two variants. We propose 2 attacks against the 5+5-round version that fit in the data requirements. The first one has a time cost just above the limit of 2^{80}, while the second is below the bound. Our attacks against 7+7 rounds are far below the claim of 2^{128}, which corresponds to the generic exhaustive search.

Our investigation leverages several properties of the cipher: first, properties arising from the U-shape attack model including the limited diffusion in the middle rounds, and the possibility to access queries that were not asked to the encryption oracle but are deduced from previous queries.

Second, a design flaw we identified in the round-key creation process allows us to attack the large version of BEANIE (7 rounds of encryption followed by 7 rounds of decryption) by lifting an attack or a distinguisher that targets 3 rounds of encryption and decryption only.

We made the most out of these observations by building two types of distinguishers: one relying on the yoyo technique [9,26] and the other relying on the truncated differential technique [20]. To limit the complexities of our attacks, we use several tricks including the precomputation of tables, made possible by the relative small block size in comparison to the other parameters.

Our analyses illustrate once more that designing a small-state tweakable block cipher is a difficult task. We give a critical discussion of our findings in this respect, stressing the impact of each characteristic that helped us build our attack.

We quickly notified BEANIE designers with a preliminary version of our results. They acknowledged our findings and concluded their original security rec-

ommendation was too optimistic[1]. Their recommendation is now to add one round (from 5 to 6), with the same security claim.

Our results are summarised in Table 1 and our codes validating the distinguishers are available at:

`https://gitlab.inria.fr/xbonneta/beanie-cryptanalysis`.

Outline. We start by describing BEANIE in the following section. We next detail some important properties for our attack: how we can define large SuperBoxes for BEANIE, and how we will use collision-finding (Section 3). We then focus on the properties of the U-shape model (Section 4). These observations are assembled to build yoyo attacks on 5+5-round BEANIE (Section 5), and truncated differential distinguishers on both versions (Section 6). Our critical discussion is in Section 7.

Type	R	T	D	M	Success proba	Reference
Boomerang	$4+4$	$> 2^{62}$	2^{29}	2^{30}		[17]
	$5+5$	$> 2^{94}$	2^{61}	2^{30}		[17]
Impossible diff.	$4+4$	$2^{73.4}$	2^{11}	$2^{21.38}$		[17]
	$5+5$	$2^{105.4}$	2^{43}	$2^{21.38}$		[17]
with partial coll.	$5+5$	2^{112}	2^{30}	$2^{27.6}$		[17]
Integral	$4+4$	2^{67}	2^{18}	-		[17]
	$5+5$	2^{99}	2^{50}	-		[17]
DS-MitM	$5+5$	2^{112}	2^{16}	2^{109}		[17]
Yoyo	$4+4$	$2^{59}(+2^{64})$	2^{21}	2^{32}	0.95	subsection 5.3
	$5+5$	2^{81}	$2^{38.75}$	2^{60}	0.69	subsection 5.4
Truncated diff.	$5+5$	$2^{66.2}$	2^{43}	$2^{69.2}$	0.91	subsection 6.2
	$5+5$	2^{59}	$2^{52.6}$	$2^{52.8}$	0.86	Appendix F
	$\mathbf{7+7^{\ddagger}}$	$\mathbf{2^{82.2}(\times 2)}$	$\mathbf{2^{59}(\times 2)}$	$\mathbf{2^{85.2}}$	**0.89**	subsection 6.2
(better key schedule)	$7+7$	$2^{114.7}(\times 2)$	$2^{59.5}(\times 2)$	$2^{117.7}$	0.90	subsection 6.5
with partial coll	$5+5$	$2^{58.3}$	2^{55}	$2^{60.2}$	0.90	subsection 6.3
	$\mathbf{7+7^{\ddagger}}$	$\mathbf{2^{74.3}(\times 2)}$	$\mathbf{2^{71}(\times 2)}$	$\mathbf{2^{76.6}}$	**0.90**	subsection 6.3
(better key schedule)	$7+7$	$2^{106.8}(\times 2)$	$2^{71.5}(\times 2)$	$2^{109.1}$	0.90	subsection 6.5
lower data, partial coll	$\mathbf{5+5^{\ddagger}}$	$\mathbf{2^{74.3}}$	$\mathbf{2^{39}}$	$\mathbf{2^{76.4}}$	**0.90**	subsection 6.4
Differential-linear	$7+7^{\ddagger}$	$2^{125.06}$	2^{81}	2^{81}	0.86	Appendix E

Table 1. Summary of the known attacks on BEANIE. Numbers in parenthesis are costs to recover all the round-keys. The memory is expressed in number of blocks of 32-bits. Attacks with a ‡ break the security claims of the designers. Our best attacks are in bold.

[1] See for instance the presentation of the cipher at FSE'26.

2 Specification of Beanie

Beanie [17] is a 32-bit low latency tweakable block cipher for external memory encryption of microcontroller units. It follows an SPN construction inspired by the AES [3], where each round operation was carefully selected to limit latency. The cipher is parameterised by a 128-bit tweak T and a 128-bit key K, used together in a complex tweak-key schedule to produce the round keys.

Data Path. The internal state is seen as a 4×2 matrix, where each cell contains a nibble. Cells are indexed from top to bottom and from left to right (see Figure 1). The data path is composed of r rounds (where r is either 5 or 7), with the last round R' being slightly different, that we write $R' \circ R^{r-1}$. The first $r-1$ rounds consist of the following operations:

AddRoundTweakey (ART), where the 32-bit round tweak-key is XORed with the internal state.

SubCells (SC), the non-linear part of the data path, where the 4-bit SBox detailed in Figure 1 is applied to each cell.

ShiftRows (SR), where the cells of the second and fourth rows are rotated one position to the left (note that $SR^{-1} = SR$).

MixColumns (MC), where each nibble is treated as an element of GF(2^4)/0x13 and multiplied by an MDS involutory 4×4 matrix M ($MC^{-1} = MC$).

For the last round R', the MixColumns is replaced by a final ART operation.

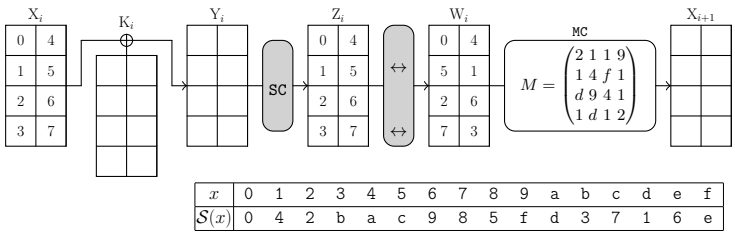


Fig. 1. One round R of the Beanie data path. State notations are taken from [17].

Tweak-key Schedule. Beanie uses a complex non-linear tweak-key schedule designed to prevent the attacker from easily building colliding round keys, as it would result in round cancellations in the U-shape attack model (see Section 4.2). The process was thus chosen to be non-linear and such that the minimum number of differentially active SBoxes is high.

As shown in Figure 2, the tweak-key schedule takes as input the 128-bit tweak T which is modified by r applications of a strong function ϕ (where r is the same as for the data path, 5 or 7) alternated with additions of the master key K and round constants. The 128-bit output is read as the concatenation of the 4 first

round tweak-keys $k_0||k_1||k_2||k_3$, providing the round tweak-keys to XOR with the data path for the first three rounds. The values for the remaining rounds are computed as follows: $k_4 = k_0 \oplus k_1$, $k_5 = k_2 \oplus k_3$, $k_6 = k_0 \oplus k_2$, $k_7 = k_1 \oplus k_3$, $k_8 = k_0 \oplus k_3$, $k_9 = k_1 \oplus k_2$. By abuse of notation we thereafter use the term of *round-key* to refer to the *round tweak-keys*. The exact details of the tweak-key schedule (which play no role in our cryptanalyses) are presented in Section A.

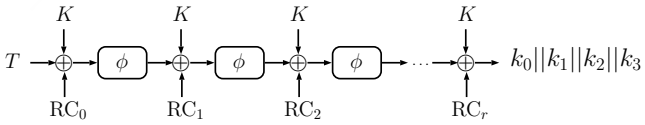


Fig. 2. High-level description of the generation of the 4 first round-keys of Beanie.

2.1 Security Claim

The designers claim security in the so-called U-shape model, where queries are made to the mapping U which does one encryption followed by one decryption. Namely, for a given unknown key K, an attacker can specify a plaintext P^0 and two tweaks T^0 and T^1 and obtain the result P^1 corresponding to P^0 going through encryption with the first tweak T^0 followed by decryption with the second tweak T^1: $P^1 = U(P^0, T^0, T^1) = E^{-1}_{K,T^1}(E_{K,T^0}(P^0))$.

For attacks and distinguishers, the notation r_0+r_1 means we target r_0 rounds of encryption followed by r_1 rounds of decryption. In what follows we use the same notations as [17] and let k_i, $i \in \{0, \cdots, r_0\}$ denote the round-keys in the encryption rounds and $\bar{k_j}$, $j \in \{0, \cdots, r_1\}$ the ones in the decryption rounds.

For the 5+5-round version, the designers claim a security in 2^{80} operations and 2^{40} U-shaped queries. For the 7+7-round version (recommended when the low-latency requirements are less stringent), the claim is increased to the standard 128 bits of security and no constraint on the U-shaped queries.

3 Cryptanalytic tools

3.1 SuperBoxes of Beanie

SuperBox. In strongly aligned ciphers such as the AES, part of the encryption process can be rewritten as parallel applications operating on independent fractions of the internal state, termed *SuperBoxes* [13]. We can define such a SuperBox S for Beanie as $S = SC \circ ART(k_i) \circ MC \circ SC$ where S applies independently on each column of the internal state.

By extension, this property implies one can compute the value of nibbles $0, 2, 5, 7$ after 1.5 rounds of Beanie (two rounds missing the last MC) from the

value of nibbles $0, 2, 5, 7$ of the input and from 16+16 bits of k_0 and k_1. The same holds for nibbles $1, 3, 4, 6$.

Middle SuperBox. The U-shaped queries $E^{-1}_{K,T^1}(E_{K,T^0}(\cdot))$ imply a similar phenomenon covering even more rounds. Recall that 2 + 2 rounds of BEANIE actually correspond to 1.5 rounds of encryption followed by 1.5 rounds of decryption, that is (we omit the round-key additions for readability):

$$SC^{-1} \circ SR^{-1} \circ MC^{-1} \circ SC^{-1} \circ SR^{-1} \circ SR \circ SC \circ MC \circ SR \circ SC.$$

As the two middle `MC` operations are missing and as $SR^{-1} = SR$ the diffusion offered by these operations is limited and the previous observation implies that 2+2 rounds of BEANIE can be seen as the application of two independent permutations applying on each half of the state (and corresponding key material).

We define the SuperBox $\hat{S}$ of BEANIE as the function:

$$\begin{aligned}\hat{S} = SC^{-1} \circ MC^{-1} \circ ART(\bar{k_{i-1}}) \circ SC^{-1} \circ SR^{-1} \circ ART(\bar{k_i}) \\ \circ ART(k_i) \circ SR \circ SC \circ ART(k_{i-1}) \circ MC \circ SC.\end{aligned}$$

Similarly to S, $\hat{S}$ computes one column of the output from one column of the input and the corresponding key material. S and $\hat{S}$ are shown in Figure 4 in Section 5.2.

3.2 Collision-finding

At multiple steps of our attacks we will need to identify collisions, i.e. distinct entries with the same value, either within a list or between 2 lists. In this section we recall some standard results on probabilities and explain the algorithms we use.

Probabilities. Assuming a uniform distribution, the probability to have no collision in a set of size N if each element can take 2^t values is:

$$\prod_{i=0}^{N-1} \left(1 - \frac{i}{2^t}\right).$$

Assuming $N \ll 2^t$, which is the regime of interest, this can be approximated by $\exp\left(-\frac{N^2}{2^{t+1}}\right)$. For two sets of N distinct values each, the probability is:

$$\left(1 - \frac{N}{2^t}\right)^N \simeq \exp\left(-\frac{N^2}{2^t}\right).$$

Assuming the number of internal collisions in a set is low, the same probability can be used for two sets of random values.

Algorithms. As we want to identify all collisions, the standard approach is to do list-merging: we sort each list, and then sequentially process each list to find equal values. As in our case we will collide on finite sets, we will use the counting sort algorithm [21]. Our strategy and the cost estimates are detailed in Section C.

In our regimes, we estimate extracting collisions using integers costs 8 memory accesses, and using sets 18 memory accesses. Furthermore, we assimilate 1 memory access and 1 round of encryption of Beanie.

4 Subtleties of the Query Model

4.1 Building Up New Queries

The U-shape model forces to chain two copies of the cipher: first one encryption with a tweak T^0, and second a decryption with an independent tweak T^1. Interestingly (and as noted by the designers of SCARF [11] which uses the same attack model) it gives the possibility to build queries that were not made.

First, since the mapping U is similar to its inverse, a query $P^1 = U(P^0, T^0, T^1)$ implies $P^0 = U(P^1, T^1, T^0)$. This has little to no interest but shows that P^0 and P^1 have similar roles, which made us call them both *plaintexts* in what follows.

Second, consider $P^1 = U(P^0, T^0, T^1)$ and $P^2 = U(P^0, T^0, T^2)$. As illustrated in Figure 3, we deduce that P^2 is also the result of the query (P^1, T^1, T^2):

$$\begin{aligned} U(P^1, T^1, T^2) &= E^{-1}_{K,T^2}(E_{K,T^1}(P^1)) = E^{-1}_{K,T^2}\left(E_{K,T^1}\left(E^{-1}_{K,T^1}(E_{K,T^0}(P^0))\right)\right) \\ &= E^{-1}_{K,T^2}(E_{K,T^0}(P^0)) = U(P^0, T^0, T^2) = P^2. \end{aligned}$$

In the most extreme case, with a single plaintext P^0 and queries of the form (P^0, T^0, T^i) for $i \in \{1, \cdots, 2^{40}\}$, the initial 2^{40} queries allow to compute about 2^{79} valid U-shaped queries to the cipher (corresponding each to a pair (T^i, T^j), $i < j$).

Limits of this approach. While we can obtain queries for many pairs of tweaks, this does not change the overall number of involved tweaks. Thus, we gain on properties that involve pairs of tweaks, but not on properties on one half of the cipher (for example, a specific value of a round-key, or a correlation on one half).

4.2 Round-Key Collisions

Middle round-key collision. The first-party analysis [17] states that with probability 2^{-32}, the last round-key from the two tweaks are equal ($k_5 = \bar{k_5}$ or $k_7 = \bar{k_7}$), meaning that the first round of decryption cancels the last round of encryption. This allows to extend an attack by 1+1 rounds, at a cost of a factor 2^{32} in time and data, by trying 2^{32} distinct decryption tweaks.

Using the above principle, we can achieve better: decrypting the same value using $2^{16.5}$ distinct tweaks allows to obtain queries corresponding to 2^{32} distinct tweak pairs. The main constraint is that we no longer have any control on the queries; for an attack to work it needs to be in a known plaintext setting.

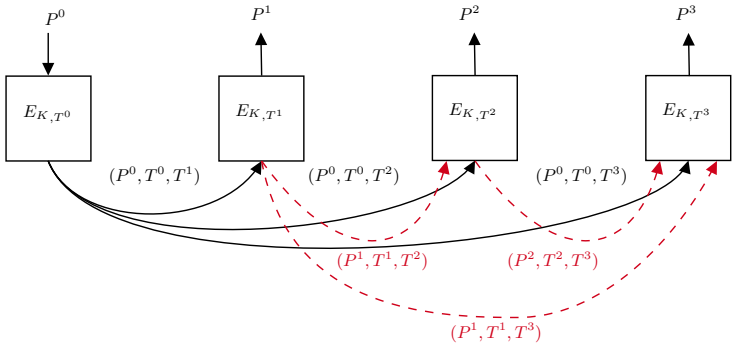


Fig. 3. From 3 actual queries (in black), we can deduce 3 additional non-trivial queries (in red) without actually requesting them.

Case of the 7+7-round variant. In the case of an analysis of the 7-round version, an attacker would look for a collision between k_7 and $\bar{k_7}$, k_6 and $\bar{k_6}$, and so on to cancel several middle rounds. While the first 4 round-keys are built from a process based on the PRINCE cipher [10] the next round-keys are deduced linearly by: $k_4 = k_0 \oplus k_1$, $k_5 = k_2 \oplus k_3$, $k_6 = k_0 \oplus k_2$, $k_7 = k_1 \oplus k_3$.

Remarkably, these last keys are related: $k_4 \oplus k_5 \oplus k_6 \oplus k_7 = 0$. Thus, if the last 3 round-keys collide ($k_5 = \bar{k_5}$, $k_6 = \bar{k_6}$ and $k_7 = \bar{k_7}$), we will have $k_4 = \bar{k_4}$. This observation induces that by checking around 2^{96} pairs of tweaks in the hope of a collision between the 3 last round-keys we actually observe a collision between the 4 last round-keys, reducing 7+7-round Beanie to 3+3-round Beanie.

As the security claim of 7+7-round Beanie is of 128 bits, expecting such a collision is possible, albeit leaving a limited margin for the 3+3-round attack.

8+8 and 9+9 rounds. While there is no claim associated with these versions, the key schedule defines k_8 and k_9. Unfortunately these keys are also in the linear span of (k_5, k_6, k_7), as $k_8 = k_0 \oplus k_3 = (k_0 \oplus k_2) \oplus (k_2 \oplus k_3) = k_6 \oplus k_5$ and $k_9 = k_1 \oplus k_2 = (k_1 \oplus k_3) \oplus (k_3 \oplus k_2) = k_7 \oplus k_5$. Hence, any attack against 7+7 rounds that uses a 4+4-round collision (such as the attacks of Section 6) can be applied *as-is* against 8+8 or 9+9 rounds if the original key schedule is used.

Partial Round-Key Collisions. The low diffusion in the two middle rounds of Beanie induces that even a partial round-key collision leads to parts of the internal state that are kept unmodified.

In 5-round Beanie for instance, if $k_5 \oplus \bar{k_5}$ is 0 in a nibble then the corresponding nibble goes through $SC^{-1} \circ SR^{-1} \circ ART(\bar{k_5}) \circ ART(k_5) \circ SR \circ SC$ which boils down to the identity. This observation can be extended by noting that for pairs of messages such a collision implies an equality of the differences when taking into account the addition of k_4 and $\bar{k_4}$, and a key collision on positions $0, 2, 5, 7$ or $1, 3, 4, 6$ allows to extend the difference equality over the `MC` operations.

We use this principle to build a 2+2-round truncated differential trail of Beanie in Section 6.1, shown in Figure 8. The designers of Beanie used this technique when evaluating the resistance of their cipher to impossible differential attacks [17]. Prior to that, Bariant *et al.* took advantage of a similar key collision to attack a weak-key version of the full version of ForkAES in [6].

4.3 A General Two-Step Approach to Attack 7+7-round Beanie

The above remark on the middle key collision for 7+7-round Beanie allows to perform a key recovery as follows:

- For many tweak pairs, apply one (or multiple) 3+3-round distinguishers,
- Apply a 3+3-round attack on the tweak pairs that passed the distinguishers.

Assuming running the first distinguisher costs less than 2^{32} (and that the following distinguishers are not prohibitively expensive), the first step will cost less than 2^{128}. Finally, if a random guess passes the distinguishers with probability 2^{-p}, we will have to repeat the attack 2^{96-p} times. We present such an example in Section E. It is based on differential-linear techniques.

5 Yoyo Analysis of Beanie

In this section we show how to use the SuperBoxes of Beanie to first mount a 4+2-round yoyo distinguisher and next extend it to 4+4 and 5+5-round attacks.

5.1 Reminder on the Yoyo Attack Vector

The yoyo game [9,26] allows to distinguish a section of an SPN cipher if it can be rewritten as $S_2 \circ L \circ S_1$ where S_1 and S_2 are SuperBoxes (that is, can be rewritten as the concatenation of at least 2 functions) and L is linear (or affine). It relies on pairs of states connected by an exchange operation as it gives states with equal Zero Difference Pattern. We recall all these central notions below:

Definition 1 (Exchanged pair [26]). *Let $x_1 = (x_1^0, x_1^1, \cdots, x_1^{w-1})$ and $x_2 = (x_2^0, x_2^1, \cdots, x_2^{w-1})$ be two states of w words and $v = (v^0, v^1, \cdots, v^{w-1})$ be a w-bit vector. Two states $\tilde{x}_1$ and $\tilde{x}_2$ can be derived by exchange from x_1 and x_2 according to the vector v as follows:*

$$\tilde{x}_1 = \rho^v(x_1, x_2) = (x_1^0 \cdot (v^0) \oplus x_2^0 \cdot (v^0 \oplus 1), \cdots, x_1^{w-1} \cdot (v^{w-1}) \oplus x_2^{w-1} \cdot (v^{w-1} \oplus 1)),$$

$$\tilde{x}_2 = \rho^v(x_2, x_1) = (x_2^0 \cdot (v^0) \oplus x_1^0 \cdot (v^0 \oplus 1), \cdots, x_2^{w-1} \cdot (v^{w-1}) \oplus x_1^{w-1} \cdot (v^{w-1} \oplus 1)).$$

Put differently, word i of $\tilde{x}_1 = \rho^v(x_1, x_2)$ is equal to the corresponding word of x_1 (i.e., x_1^i) if $v^i = 1$, and is equal to the corresponding word of x_2 (x_2^i) otherwise.

Definition 2 (Zero Difference Pattern [26]). *Let x be a state of w words. The zero difference pattern of x over its w words is given by a w-bit vector $\nu(x) = (z^0, z^1, \cdots, z^{w-1})$ where $z^i = 1$ if the i-th word of x is 0, and where $z^i = 0$ otherwise.*

Theorem 1 (from [26], proof in [26] and in Section B.1). *Let x_1 and x_2 be two states of w words and $\tilde{x}_1 = \rho^v(x_1, x_2)$ and $\tilde{x}_2 = \rho^v(x_2, x_1)$ two states deduced by exchanging words according to the vector v. Then for a $S_2 \circ L \circ S_1$ operation where S_1 is aligned on the w words we have:*

$$\nu(S_2 \circ L \circ S_1(x_1) \oplus S_2 \circ L \circ S_1(x_2)) = \nu(S_2 \circ L \circ S_1(\tilde{x}_1) \oplus S_2 \circ L \circ S_1(\tilde{x}_2)).$$

5.2 4+2-round Yoyo Distinguisher on Beanie

The definition of the SuperBoxes of Beanie made in Section 3.1 allows to easily set up Beanie to apply the yoyo game. We rearrange 4+2 rounds[2] as follows (we again omit the round-key additions for readability):

$$SR^{-1} \circ \underbrace{SC^{-1} \circ MC^{-1} \circ SC^{-1} \circ SR^{-1} \circ SR \circ SC \circ MC \circ SC}_{\hat{S}} \circ \underbrace{SR \circ MC \circ SR}_{L} \circ \underbrace{SC \circ MC \circ SC}_{S} \circ SR$$

where S and $\hat{S}$ are the two SuperBoxes of Beanie acting independently on each column of the internal state and L is a linear operation. This cutting is represented in Figure 4.

Theorem 1 implies[3]:

$$\nu(\hat{S} \circ L \circ S(x_1) \oplus \hat{S} \circ L \circ S(x_2)) = \nu(\hat{S} \circ L \circ S(\tilde{x}_1) \oplus \hat{S} \circ L \circ S(\tilde{x}_2)),$$

where (x_1, x_2) is a pair of 32-bit input states and $(\tilde{x}_1, \tilde{x}_2)$ is the corresponding exchanged pair over the columns (w.l.o.g. $\tilde{x}_1$ has the same first column as x_1 and the same second column as x_2, while $\tilde{x}_2$ has the same first column as x_2 and the same second column as x_1).

This property could be used as follow: consider a pair of states with a remarkable zero difference pattern (for instance with the first column being inactive), ask for its encryption through $\hat{S} \circ L \circ S$, build an exchanged pair of the obtained output pair by exchanging a column and ask for its decryption. With probability 1 the new pair has the same remarkable zero difference pattern with the first column being inactive. For a random permutation, this inactivity would occur with probability 2^{-16}.

By taking the `SR` operations into account we obtain the distinguisher of 4+2-round Beanie detailed in Algorithm 1. We experimentally confirmed this distinguisher using the reference code provided by the designers of Beanie [17].

We introduce the following notation to help describe our next attacks:

Definition 3 (exchanged quartet). *We call a quartet $(x_1, x_2, \tilde{x}_1, \tilde{x}_2)$ an* exchanged quartet *if it is made of 2 distinct pairs (x_1, x_2) and $(\tilde{x}_1, \tilde{x}_2)$ where one is an exchanged pair of the other (depending on the distinguisher, the exchange can be at the input side, at the output side, or even at another point in the cipher). If one pair has an inactive column, then:*

- *If the other pair has the same inactive column, this is a* valid quartet.
- *Otherwise, this is an* impossible quartet *(as this cannot happen according to Theorem 1).*

[2] Note that a similar reasoning allows to cover 2+4 rounds.

[3] a similar relation holds for $S \circ L \circ \hat{S}$.

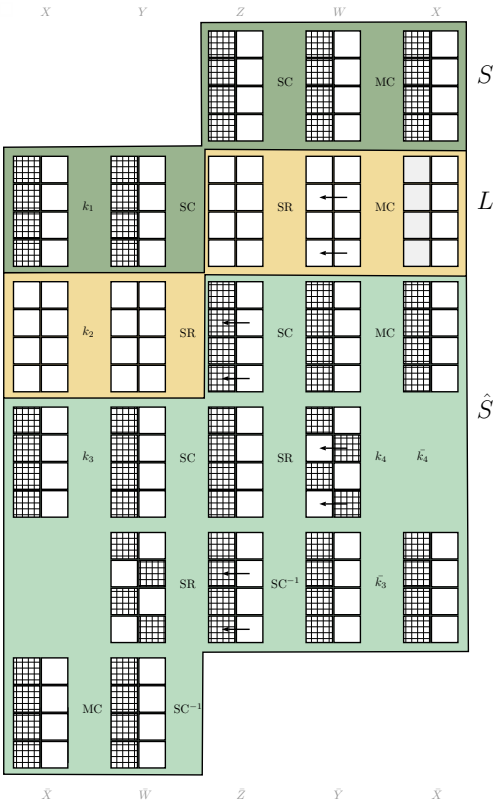


Fig. 4. Portion of Beanie seen as $\hat{S} \circ L \circ S$.

5.3 4+4-round Yoyo Attacks on Beanie

Naive variant. The previous distinguisher can easily be extended into a key-recovery attack by appending 2 rounds, in a procedure resembling the mixture attack from [18]. The idea is simply to make a guess on the last 2 round-keys to undo these rounds and use the distinguisher of Algorithm 1: if the pairs do not behave in line with Theorem 1, the key guess is incorrect.

This technique requires the full codebook and a time complexity of roughly 2^{64} encryptions. We thus leave the details to Section B.2 and focus on a more efficient variant which is the basis for our attack on 5+5 rounds.

Optimised variant. In this variant the yoyo distinguisher is positioned at the bottom (covering the last 2+4 rounds), and the key recovery step is at the top (recovering k_0 and k_1). The key recovery step is optimised by relying on the

Algorithm 1 4+2-round Yoyo Distinguisher of Beanie.

For a fixed key K and a fixed pair of tweaks (T^0, T^1), pick a random pair of plaintexts (P_1^0, P_2^0) inactive in nibbles $(0, 2, 5, 7)$.
Ask for the encryption of (P_1^0, P_2^0), denoted (P_1^1, P_2^1).
Build the exchanged pair $(\tilde{P}_1^1, \tilde{P}_2^1)$ by exchanging cells $(0, 2, 5, 7)$ between P_1^1 and P_2^1.
If (P_1^1, P_2^1) and $(\tilde{P}_1^1, \tilde{P}_2^1)$ are the same pair, pick a new pair (P_1^0, P_2^0).
Ask for the decryption of $(\tilde{P}_1^1, \tilde{P}_2^1)$, denoted $(\tilde{P}_1^0, \tilde{P}_2^0)$.
if $(\tilde{P}_1^0, \tilde{P}_2^0)$ is inactive in nibbles $(0, 2, 5, 7)$ **then**
    return "4+2-round Beanie "
end if
return "random permutation"

limited diffusion of 1.5 rounds. Also, instead of requiring the full codebook the attack relies on a limited set of known plaintexts[4].

The attack starts by analysing the random data to identify exchanged quartets (at the output side of $S \circ L \circ \hat{S}$) among it, as detailed in Algorithm 2. Algorithm 3 makes a guess on k_0 and k_1 and observe if it partially encrypts the exchanged quartets into quartets in line with Theorem 1, a property of the correct guess.

Algorithm 2 Identification of random exchanged quartets at the output of $S \circ L \circ \hat{S}$. We use fixed tweaks and known-plaintext queries.

Do 2^q random queries $P_i^1 = U(P_i^0, T^0, T^1)$ for a fixed key and a fixed pair of tweaks.
for all 2^{2q-1} pairs of outputs (P_i^1, P_j^1) **do**
    Check if the corresponding exchanged pair $(\tilde{P}_i^1, \tilde{P}_j^1)$ is among the data we have.
    Store plaintexts of the exchanged quartet if this is the case.
end for ▷ We get 2^{4q-66} exchanged quartets
return the stored list of exchanged quartets $(P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0)$.

Details of the Probabilities. Given an exchanged quartet and a guess for k_0 and k_1, we partially encrypt the two exchanged pairs of the quartet to obtain their difference at the input of the $S \circ L \circ \hat{S}$ function (an illustration in the 5+5-round case is given in Figure 5). The probability that both pairs are inactive over the first column is 2^{-16} when the key guess is correct (as in that case Theorem 1 applies) and is 2^{-32} for a wrong key guess. Moreover, it is impossible for the correct key guess to lead to a quartet in which one pair has an inactive column and not the other.

When also taking into account the case where the second column could be inactive at the input of $\hat{S}$ we obtain:

[4] A version of the attack relying on chosen plaintexts is possible, but the known plaintext scenario is required to extend the attack to 5+5 rounds.

– For the correct key guess, a quartet is valid with probability 2^{-15}.
– For a wrong key guess, a quartet is valid with probability 2^{-31}.
– For a wrong key guess, a quartet is impossible with probability 2^{-14}.

Under the correct key guess, the 2^{4q-66} exchanged quartets identified by Algorithm 2 generate approximately 2^{4q-81} valid quartets and no impossible quartets.

To eliminate enough wrong keys we want 4 valid quartets. With 2^{18} exchanged quartets the correct key has a 95% chance of having at least 4 valid quartets. Hence we choose $q = 21$.

In the case of a wrong key guess, having 4 valid quartets and no impossible quartet is of probability $\binom{2^{18}}{4} \times (2^{-31})^4 \times (1-2^{-14})^{2^{18}-4} \simeq 2^{-79.6}$. As we test a total of 2^{64} keys we do not expect any wrong key guess to survive.

Implementation of the key recovery. In the first step of the attack we make a partial key guess of k_0 and k_1 and use it to partially encrypt the previously identified quartets $(P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0)$. We keep the keys and the quartets such that both (P_i^0, P_j^0) and $(\tilde{P}_i^0, \tilde{P}_j^0)$ are encrypted after 2 rounds into states inactive in nibbles $5, 7$ or $4, 6$. As detailed in Algorithm 3, we organise this step by making two independent guesses of 24 key bits. This process can be visualised in Figure 6, where we make a guess on the key bits in set ① and ④ to compute the difference in the first column of X_2.

We next need to efficiently determine the value of the remaining 16 bits of k_1 (namely, $k_1[0, 2, 5, 7]$). We can make a guess on these bits to check the activity pattern in the first column of X_2 for (P_i^0, P_j^0) and $(\tilde{P}_i^0, \tilde{P}_j^0)$. If the key guess leads to an impossible quartet we remove the key from the possible set. The correct key is the one with no impossible quartets and 4 valid quartets.

We implement this step efficiently by relying on precomputed tables: the first tables ($\boldsymbol{B^1}$ and $\boldsymbol{B^2}$) simply contain the correspondence between a 16-bit (half) plaintext P, a 16-bit (half) key k_0 and the 16-bit encryption after one round. Next, Table $\boldsymbol{C}$ allows to compute from an input pair of half states at X_1 (positions $0, 2, 5, 7$) the possible values for $k_1[0, 2, 5, 7]$ that lead this pair to a difference inactive in nibbles $1, 3$ (or $0, 2$) of X_2. This table is created by listing all the possible pairs of values for the first column of X_2 with the correct inactivity pattern (inactivity in $1, 3$ or $0, 2$) and partially decrypting it up to Y_1.

Overall cost. The attack uses 2^{21} data. The time complexity is dominated by the search of valid k_1 for all the 2^{51} collisions. For each one this requires seeking collisions among 2^8 keys which is overall roughly 2^{62} 1-round operations. Hence, we estimate the attack to cost around 2^{59} time. The success probability is 0.95.

The memory complexity is dominated by the 2 precomputed tables $\boldsymbol{B^1}$ and $\boldsymbol{B^2}$ storing 2^{32} words of 16 bits each, equivalent to a total of 2^{32} words of 32 bits.

Recovering the remaining round-keys. First, we reuse the same initial data to recover $\bar{k_0}, \bar{k_1}$ by swapping input and output. Assuming we identified the correct $k_0, \bar{k_0}, k_1, \bar{k_1}$, we can recover the missing 128-bit key material $(k_2, k_3, k_4 = k_0 \oplus$

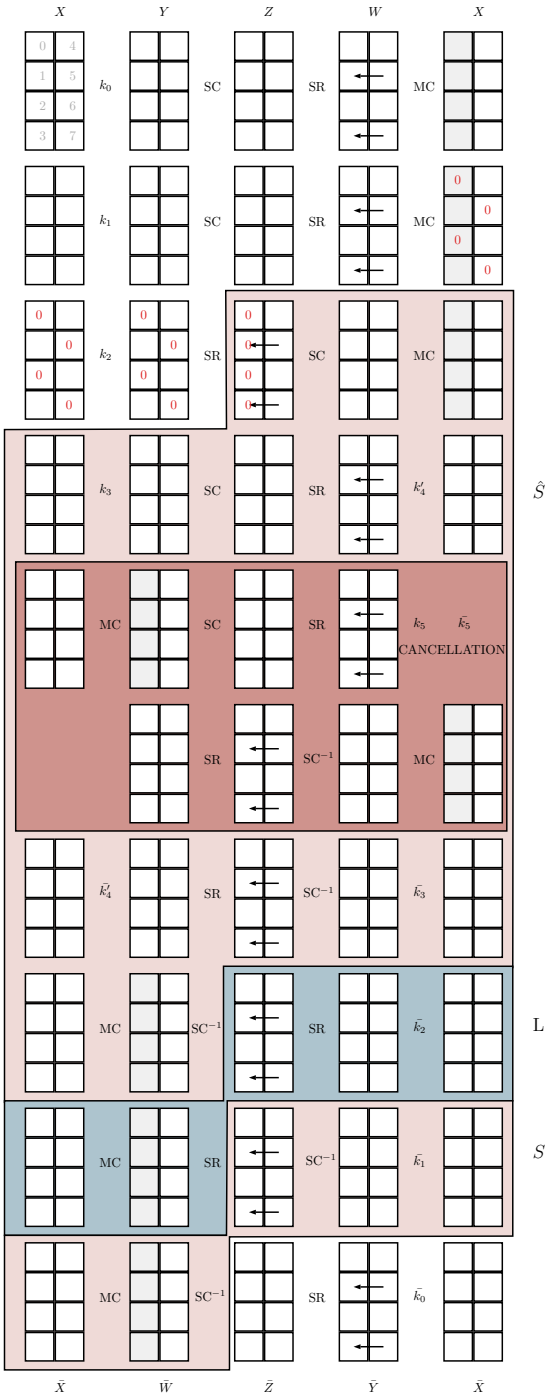


Fig. 5. Yoyo attack on 5+5-round Beanie. k'_4 and $\bar{k}'_4$ are equivalent values.

Algorithm 3 Key-recovery of k_0, k_1. The key guess sets are detailed in Figure 6.

Precompute Table $\boldsymbol{B^1}, \boldsymbol{B^2}$ (Algorithm 4) and Table $\boldsymbol{C}$ (Algorithm 5)
for all 2^{18} exchanged quartets $(P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0)$ identified by Algorithm 2 **do**
    for all 2^{24} values for the key bits of k_0, k_1 in set ① of Figure 6 **do**
        Partially encrypt P_i^0 and P_j^0 over the first 2 rounds to obtain $\Delta W_1[5, 7]$.
        Deduce their contribution to the difference in nibbles 5 and 7 of X_2 (i.e. $M \times (0, \Delta W_1[5], 0, \Delta W_1[7])^\intercal[1, 3]$) and to the difference in nibbles 4 and 6 of X_2 (i.e. $M \times (0, \Delta W_1[5], 0, \Delta W_1[7])^\intercal[0, 2]$).
        Do the same with $\tilde{P}_i^0$ and $\tilde{P}_j^0$.
    end for
    Store the concatenated values in Table $\boldsymbol{D^1}$ together with the key guesses.
    Follow the same idea with the set ④ of Figure 6 and store the results in a table $\boldsymbol{D^2}$.
    Sort both tables according to the differences in nibbles 5 and 7 of X_2 and look for collisions on these values (corresponding to a quartet made of two pairs inactive in nibbles 5 and 7 of X_2).
    Sort both tables according to the differences in nibbles 4 and 6 of X_2 and look for collisions on these values (corresponding to a quartet made of two pairs inactive in nibbles 4 and 6 of X_2).
end for
▷ We expect $(2^{18} \times 2^{24} \times 2^{24} \times 2^{-8-8}) = 2^{50}$ collisions for each inactivity pattern.
for all 2 inactivity patterns **do**
    for all 2^{50} collisions $(k_0, k_1[1, 3, 4, 6], (P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0))$ **do**
        Fetch the encryption of $P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0$ after one round in Table $\boldsymbol{B^1}$ and $\boldsymbol{B^2}$.
        Fetch the possible $k_1[0, 2, 5, 7]$ in Table $\boldsymbol{C}$ for (P_i^0, P_j^0) and for $(\tilde{P}_i^0, \tilde{P}_j^0)$ and compute the intersection.
    end for
end for
for all key guesses with 4 valid quartets **do** ▷ Approx. $2^{7.4}$ are expected
    Partially encrypt all the initial quartets.
    if it gives an impossible quartet **then**
        Mark the key guess as impossible.
    end if
end for
Return the only surviving value for k_0 and k_1.

Algorithm 4 Construction of Table $\boldsymbol{B^1}$ and $\boldsymbol{B^2}$.

Initialise a table $\boldsymbol{B^1}$
for all 16-bit half plaintext P (over nibbles $0, 2, 5, 7$) **do**
    for all 16-bit half key k_0 (over nibbles $0, 2, 5, 7$) **do**
        Compute the encryption of half-plaintext P with k_0.
        Store the obtained 16-bit half state in $\boldsymbol{B^1}$.
    end for
end for
Similarly, build Table $\boldsymbol{B^2}$ for positions $1, 3, 4, 6$.

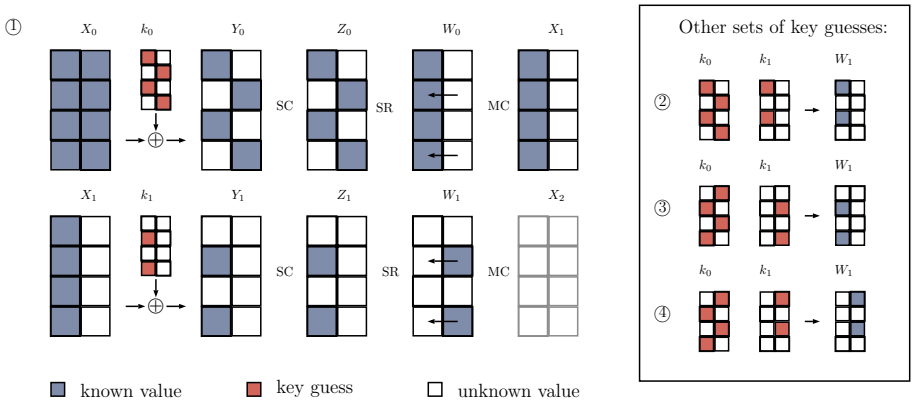


Fig. 6. The difference in X_2 can be computed by combining the partial encryption computed from 4 sets of key guesses of 24 bits each, denoted ①,②, ③ and ④.

Algorithm 5 Construction of Table $\boldsymbol{C}$.

1: Initialise a table $\boldsymbol{C}$ of 2^{16} empty lists.
2: **for all** (S, S') a pair of first column of X_2 such that $S \oplus S'$ is inactive in the required nibbles $(1, 3$ or $0, 2)$ **do**
3: Compute (Y, Y') its keyless one-round decryption over `MC`, `SR` and `SC`.
4: Add Y to the list $\boldsymbol{C}[Y \oplus Y']$.
5: **end for**

$k_1, \bar{k_2}, \bar{k_3}, \bar{k_4} = \bar{k_0} \oplus \bar{k_1}$) with the generic meet-in-the-middle against double-encryption, at a cost equivalent to roughly 2^{64} encryptions.

5.4 5+5-round Yoyo Attack on Beanie

We apply the previous attack to 2^{2t-1} unordered pairs (T^α, T^β) and hope to find a collision $k_5 = \bar{k_5}$. As noted in the first-party analysis, this collision generically allows to add 1+1 rounds to any distinguisher or attack, as in this case the 2 middle rounds cancel each other.

Queries. We choose a fixed encryption tweak T^0 and 2^t random decryption tweaks T^α. We query the same 2^q random plaintexts for these 2^t (T^0, T^α), so the data requirement of the attack is 2^{t+q}. As explained in Section 4, the initial queries give us access to 2^q input/output pairs for *each* of the 2^{2t-1} unordered pairs (T^α, T^β). We want 5 valid quartets to filter efficiently the wrong keys. We use $q = 20.9$ to obtain $2^{17.6}$ exchanged quartets. We take $t = 16.85$, so that the probability that one colliding pair of tweaks has at least 5 valid quartets is, from the analysis in Section D, 0.69.

Precomputation. Looking for the possible values of k_1 in Line 17 of Algorithm 3 requires to do the intersection of lists of size 2^8. To avoid this cost in the 5+5-round attack, we precompute a large table which for each quartet $(P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0)$ directly returns the values of $k_1[0, 2, 5, 7]$ that make *both* (P_i^0, P_j^0) and $(\tilde{P}_i^0, \tilde{P}_j^0)$ having the expected differences.

The construction of this table, that we call Table $\boldsymbol{C'}$, is detailed in Algorithm 6. Its dominant cost is the computation of intersections, which requires roughly 2^{70} operations, which is not the bottleneck of the attack.

Algorithm 6 Construction of Table $\boldsymbol{C'}$.

1: Initialise a table $\boldsymbol{C'}$ of 2^{61} empty lists.
2: **for all** 2^{61} possible ordered quartets $(P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0)$ of values in nibbles $0, 2, 5, 7$ **do**
3: Fetch the values of $k_1[0, 2, 5, 7]$ for (P_i^0, P_j^0) in Table $\boldsymbol{C}$
4: Fetch the values of $k_1[0, 2, 5, 7]$ for $(\tilde{P}_i^0, \tilde{P}_j^0)$ in Table $\boldsymbol{C}$
5: Store the intersection in the corresponding line of $\boldsymbol{C'}$
6: **end for**

Key recovery. The key recovery applies the 4+4-round attack in Algorithm 3 $2^{32.7}$ times, but this time with $2^{17.6}$ quartets. A wrong key is expected to have 5 valid quartets with probability $\binom{2^{17.6}}{5} \times (2^{-31})^5 \simeq 2^{-73.9}$. The total number of keys studied in Line 20 of Algorithm 3 for all the $2^{32.7}$ considered pairs of tweaks is expected to be less than $2^{-73.9} \times 2^{64} \times 2^{32.7} = 2^{22.8}$.

For all these we further look for impossible quartets. Since the probability for a wrong key to have no impossible quartet among the $2^{17.6} - 5$ remaining is $(1 - 2^{-14})^{2^{17.6}-5} \simeq 2^{-17.5}$ we expect around 40 wrong guesses to survive. We then attack the inverse cipher for each of them. The probability for a wrong tweak pair to have a surviving guess is $2^{64-73.9-17.5} = 2^{-27.4}$, and in the end only the correct key should remain.

Cost Estimate. The attack uses $2^{20.9} \times 2^{16.85} = 2^{38.75}$ data, and the dominant cost comes from the 5 lookups (4 to the $\boldsymbol{B}$ tables and 1 to Table $\boldsymbol{C'}$) made for each of the $2 \times 2^{32} \times 2^{17.6}$ collisions repeated for each of the $2^{32.7}$ pairs of tweaks, so a total of $2^{85.6}$ lookups. This number can be easily further reduced by storing in $\boldsymbol{D^1}$ and $\boldsymbol{D^2}$ the partial encryption after one round of $P_i^0, P_j^0, \tilde{P}_i^0, \tilde{P}_j^0$. In that way, the 5 lookups are reduced to 2, and we have a total of $2^{84.3}$ lookups. Assuming each costs around 1 Beanie round, we estimate the overall cost to be 2^{81} 5+5-round Beanie encryptions, for a success probability of 0.69.

While our estimate is slightly above the 2^{80} claim, this number depends both on how we compare memory accesses to encryptions (which in itself depends on what we condider: 5 rounds, 5+5 rounds, with or without the key schedule...) and the targeted success probability.

Recovering the Remainder. As previously, we can recover the other round keys with a meet-in-the-middle approach. The master key K can also be recovered with a negligible cost, as explained in the first-party analysis [17].

Overall Cost. The steps after k_0, k_1 have been recovered have a comparably negligible cost. Thus, overall, our attack costs $2^{38.75}$ data, $2^{84.3}$ lookups that we estimate cost 2^{81} overall, and uses a memory dominated by the size of Table $\boldsymbol{C'}$, which requires roughly 2^{60} blocks of 32 bits.

6 Truncated Differential Analysis of Beanie

In this section, we detail how to leverage probability-1 truncated differential distinguishers on 2+2 rounds of Beanie to build 5+5 and 7+7-round attacks.

Overall Idea. The goal of our attacks is to simultaneously identify a pair of tweaks which gives key collisions in the middle and the corresponding correct key guesses for k_0 and $\bar{k_0}$. Roughly, the pair of tweaks implies that we are left with a version of Beanie reduced to 3+3 rounds only. The correct key guesses then partially encrypt and decrypt pairs of messages to values in line with the probability-1 truncated differential trail over 2+2 rounds. The attack would fail if there are no round-key collisions, which allows to identify both when the reduction happened and what are the values of k_0 and $\bar{k_0}$.

To limit data complexity, we again take advantage of the query model (as explained in Section 4). The time complexity is maintained low by treating independently the guess on k_0 and on $\bar{k_0}$, and only merging them in a second step. The key guesses are used to compute pairs of plaintexts leading after one round of encryption to the input inactivity pattern of the 2+2-round truncated differential trail. The correct key is among the keys leading the same data pairs to valid inputs and outputs for the truncated differential distinguisher of probability 1.

6.1 Two 2+2-round Truncated Differential Distinguishers

As depicted in Figure 7, our *first truncated differential distinguisher* directly arises from the middle SuperBox of Beanie discussed in Section 3.1. Since the transformations made in 2+2 rounds can be seen as two independent functions acting on 16 bits of the internal state, if there is no difference in nibbles $1, 3, 4, 6$ of the input, the same nibble positions are inactive at the output (the same observation holds for positions $0, 2, 5, 7$). This distinguisher is of probability 1 for Beanie, while for a random permutation the expected output inactivity pattern appears with probability 2^{-16}.

Our *second truncated differential distinguisher* relies on a partial key collision between k_2 and $\bar{k_2}$, the two round-keys added in the middle. As shown in Figure 8, we require a collision on the 16 bits that are aligned with the 16 bits of the internal state that might be active. In that way, the operations

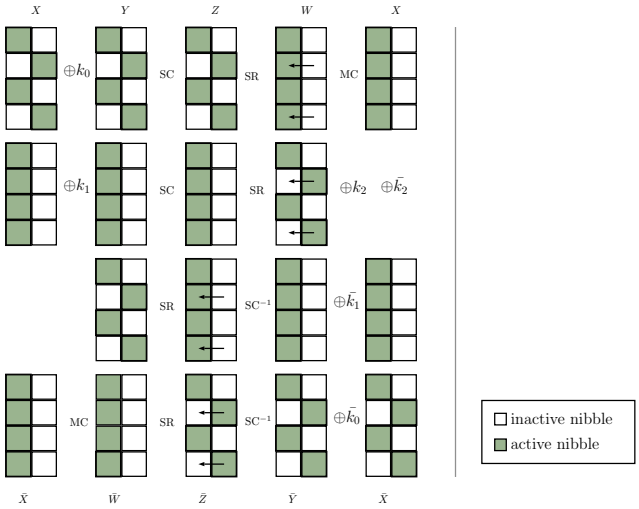


Fig. 7. Truncated differential of probability 1 deduced from the middle SuperBox.

$SC^{-1} \circ SR^{-1} \circ ART(\bar{k_2}) \circ ART(k_2) \circ SR \circ SC$ in the middle turn into identity for half of the internal state. Since this half also aligns with the `MC` operations of the first and last round, it further implies that the difference at the input of the first `MC` and at the output of the last `MC` is the same. Hence, the activity patterns of the 4 nibbles of the SuperBox with a partial collision are identical.

In our attack, we fix the desired position of the partial collision of the round-keys to nibbles $0, 2, 5, 7$ (the other option is to use $1, 3, 4, 6$, both are of probability 2^{-16}) and we consider 6 truncated differential trails with the following activity pattern at the input (equal to the one at the output): cell 0 active, cell 2 active, both cells 0 and 2 active, cell 5 active, cell 7 active (depicted in Figure 8) and both cells 5 and 7 active. Note these patterns are active in one column at most: this allows us to make guesses on halves of the first round round-key in the prepended and appended round used for key recovery (see Section 6.3).

Probabilities. We consider a random plaintext pair of 3+3-round Beanie. After partial encryption over one round, the obtained difference might be:

- For the first distinguisher:
  - Inactive in 0,2,5,7 or 1,3,4,6: probability $p_0 = \frac{2^{16}-1}{2^{32}-1}$ each,
  - Nothing happens: probability $(1 - 2p_0) = \left(1 - 2 \times \frac{2^{16}-1}{2^{32}-1}\right)$
- For the second distinguisher:
  - Active only in one fixed nibble: this event has probability $\frac{2^4-1}{2^{32}-1}$,
  - Active only in two fixed nibbles: this event has probability $\frac{\left(2^4-1\right)^2}{2^{32}-1}$,
  - Nothing happens: probability that is the complement of the above events.

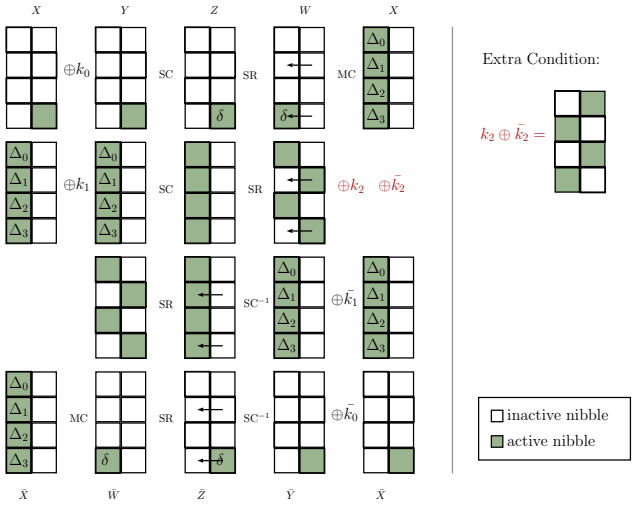


Fig. 8. Truncated differential of probability 1 relying on a partial collision of the middle round-keys k_2 and $\bar{k_2}$ (or if there are cancelled rounds in the middle, the partial collision is between $\texttt{MC}^{-1}(k_2)$ and $\texttt{MC}^{-1}(\bar{k_2})$). Any input activity pattern on cells 0,2,5,7 will be identical at the output.

Suppose we made a key guess of k_0 and $\bar{k_0}$:

- If the guess is correct and if an input pair has after 1 encryption round one of the activity patterns previously discussed then its corresponding output pair after 1 decryption round always has differences in the exact same status as the distinguishers are of probability 1.
- If the guess is incorrect, an input pair after 1 encryption round and its corresponding output pair after 1 decryption round have differences in the exact same status by chance. For instance, the probability to have N *pairs* that randomly have the same status in the case of distinguisher 1 is: $\left(2p_0^2 + (1-2p_0)^2\right)^N = \sum_{i=0}^{N} \binom{N}{i} \left(2p_0^2\right)^i (1-2p_0)^{2(N-i)}$ where $p_0 = \frac{2^{16}-1}{2^{32}-1}$.

Eliminating wrong keys. As $p_0 \ll 1$, the largest term of the previous sum is the one corresponding to $i = 0$, meaning the probability is close to the one corresponding to the situation where there are no pairs with an inactive column. Thus, in order to decrease the probability for a wrong key guess to look valid, we slightly change our test and additionally require that at least c pairs of states are inactive in 0,2,5,7 or 1,3,4,6. It gives:

$$\underbrace{\left(2p_0^2 + (1-2p_0)^2\right)^N}_{\text{Same events everywhere}} - \sum_{i=0}^{c-1} \underbrace{\binom{N}{i} \left(2p_0^2\right)^i (1-2p_0)^{2(N-i)}}_{\text{Same events and exactly } i \text{ eventful pairs}}$$

By choosing a proper value of $c > 0$ the attacker decreases the amount of data required to keep no false positives (in comparison to the case where $c = 0$), while on the other hand the correct keys is ensured to pass this new test (see below). If $N2p_0^2 \ll 1$, the formula can be approximated by the largest remaining term in the expansion of $\left(2p_0^2 + (1-2p_0)^2\right)^N$, which is the probability to have exactly c matching events: $\binom{N}{c}(2p_0^2)^c\,(1-2p_0)^{2(N-c)}$.

Keeping correct keys. The constraint of c events also applies for correct guesses. In this case the probability to have at least c events is

$$1 - \sum_{i=0}^{c-1} \binom{N}{i} (2p_0)^i (1-2p_0)^{N-i}.$$

6.2 First Truncated Differential Attack on Beanie

Data Generation. We start by querying the encryption of 2^p plaintexts P_i^0 for the fixed unknown key K and a set of 2^t pairs of tweaks (T^0, T^α) where T^0 is fixed and T^α takes 2^t different values. We denote by P_i^α the output corresponding to the encryption of the input P_i^0 with the key K and the tweak (T^0, T^α).

As discussed in Section 4 this initial collection of data can be seen as 2^{2t} sets of 2^p inputs with their associated outputs when encrypted under key K and tweak (T^α, T^β), denoted (P_i^α, P_i^β).

Key Guess. We guess the first round round-key for all the 2^t considered tweaks in order to compute which pairs in the data set have a difference matching the input of the truncated differential trail of Figure 7 after one round of encryption.

The complexity of this step is made reasonably low by using a precomputation table as explained in Algorithm 7. The idea of this algorithm is to start from all the possible pairs of internal states (X, X') positioned after the MixColumns of the first round and for which $X \oplus X'$ is inactive in the positions defined by the truncated differential trail (nibbles $1, 3, 4, 6$). We compute their decryption through `MC`, `SR` and `SC` and obtain the associated set of possible pairs of internal states (Y, Y') positioned after the addition of k_0. We store Y at the index $Y \oplus Y'$ in a table $\boldsymbol{A}$. There are overall 2^{47} pairs (X, X') to consider.

Once built, Table $\boldsymbol{A}$ allows to easily deduce for a given pair of plaintexts (P_1, P_2) what are the keys k_0 that lead after one round to the desired activity pattern. It is given by $k_0 \in \{P_1 \oplus Y | Y \in A[P_1 \oplus P_2]\} \cup \{P_2 \oplus Y | Y \in A[P_1 \oplus P_2]\}$.

Collision Step. Table $\boldsymbol{A}$ is used to build a larger table where each line contains the tweak T^α, the value of the 32-bit key guess for k_0, and the associated set of pairs of indices (i, j) indicating that the pair of inputs (P_i^α, P_j^α) has a difference inactive in the required nibbles after one round when the first-round round-key is k_0. This table is sorted according to the list of pairs of indices (i, j).

We conclude the attack by looking for a collision of the exact list of pairs of indices (i, j). (If any,) a pair of tweaks reducing the number of rounds to 3+3

Algorithm 7 Precomputation of valid keys

1: Initialise a table $\boldsymbol{A}$ of 2^{32} empty lists
2: **for all** unordered pairs (X, X') such that $X \oplus X'$ is inactive in nibbles $1, 3, 4, 6$ **do**
3: Compute Y, Y' the keyless one-round decryption of X and X'
4: Add Y to the list $\boldsymbol{A}[Y \oplus Y']$
5: **end for** ▷ For a pair P_1, P_2, the valid keys are $P_1 \oplus \boldsymbol{A}[P_1 \oplus P2]$ and $P_2 \oplus \boldsymbol{A}[P_1 \oplus P2]$

and the correct key guesses will be among the colliding elements as it shows a behaviour in line with the probability-1 truncated differential trail in the middle: the following analysis details how to set the different parameters such that the attack succeeds. The attack is summarised in Algorithm 8.

Identifying wrong guesses. To strengthen the attack, we consider both the distinguisher depicted in Figure 7 and its simple variant where the inactive positions are $0, 2, 5, 7$. We consider the events where these two inactivity positions happen, and the one where nothing happens. The probability to have by chance identical events for each of the N pairs becomes:

$$\left(\left(\frac{2^{16}-1}{2^{32}-1}\right)^2 + \left(\frac{2^{16}-1}{2^{32}-1}\right)^2 + \left(1 - 2 \times \frac{2^{16}-1}{2^{32}-1}\right)^2\right)^N = \left(\frac{2 + \left(2^{16}-1\right)^2}{\left(2^{16}+1\right)^2}\right)^N \simeq 2^{-N \times 2^{-13.47}}.$$

We further requires the number of collisions to be $\geq c$ so the probability is:

$$2^{-N \times 2^{-13.47}} - \sum_{i=0}^{c-1} \binom{N}{i} \left(2\left(\frac{2^{16}-1}{2^{32}-1}\right)^2\right)^i \left(1 - 2 \times \frac{2^{16}-1}{2^{32}-1}\right)^{2(N-i)}. \quad (1)$$

Success probability of a correct guess. With a tweak collision and correct key guesses, the probability to have at least c events will be

$$1 - \sum_{i=0}^{c-1} \binom{N}{i} \left(2\left(\frac{2^{16}-1}{2^{32}-1}\right)\right)^i \left(1 - 2 \times \frac{2^{16}-1}{2^{32}-1}\right)^{N-i}.$$

Complexity analysis. The attacker initially queries 2^{p+t} encryptions. Building the table requires for each tweak T^α to go over all the $N = 2^{2p-1}$ pairs, and for each one to write down the expected 2^{16} keys for which this pair is inactive in nibbles $1, 3, 4, 6$ after one round (respectively $0, 2, 5, 7$).

From this, we build a table of 2^{t+32} lines, where each line contains the tweak T^α, the 32-bit key guess for k_0, and the associated set of pairs of indices (i, j) indicating that (P_i^α, P_j^α) has a difference inactive in nibbles $1, 3, 4, 6$ after one round when the first-round round-key is k_0, followed by a second list of indices for the positions $0, 2, 5, 7$. We look for collisions in this table, which costs $18 \times 2^{t+32}$ Beanie rounds (see Section 3.2).

For a given tweak T^α and a given key guess of k_0, a proportion of 2^{-15} pairs has a difference inactive in 4 nibbles after one round for one of the two patterns, so in each line of the table we expect an average of $2^{2p-1-15}$ pairs of indices (i, j). For the correct pair of tweaks T^α, T^β that gives the required last round-keys colliding and for the correct key guess of k_0, $\bar{k_0}$ we have the exact same list of pairs of indices (i, j) in the table. We need p to be large enough for this list to be of size $\geq c$ with a good enough probability, which implies $p \geq 8 + 0.5 \log_2(c)$.

We fix c such that all wrong keys are expected to be eliminated. From this we choose t, p to minimize the data complexity while targeting a success probability, computed from the formula in Section D, of 90%.

Results. Against 5+5 rounds:

- Parameters: $t = 33.3$ to collide on 2 rounds, $p = 9.7$ to have $N = 2^{18.4}$ pairs. We want at least $c = 8$ eventful plaintext pairs. Data is $2^{t+p} = 2^{43}$.
- Time: The cost is dominated by the sorting of the first list. It contains $2^{33.3+32} = 2^{65.3}$ entries, thus we estimate the cost to be $2^{69.5}$ 1-round encryptions. Using Equation (1), the probability to collide and have at least $c = 8$ eventful plaintext pairs is $2^{-146.5}$, thus we do not expect any wrong key to survive. Overall this costs $2^{66.2}$ equivalent encryptions.
- Memory: This is dominated by the first table, which contains $2^{33.3+32}$ lines of roughly $128 + 11 \times 32$ bits. Overall this is $2^{69.2}$.
- Success probability: we have $2^{33.3}$ tweaks, and a colliding tweak has at least 8 collisions with probability 0.82. Overall, there is at least one good tweak with probability $1 - \exp(-2^{33.3 \times 2 - 65} \times 0.82) \simeq 0.91$.

Against 7+7 rounds, we use 2^{16} times more tweaks to ensure the extra round-key collision. A filter of $2^{-146.5}$ would leave some wrong guesses so we change the cutoff to $c = 9$ collisions to reach a filter of $2^{-162.3}$, and otherwise increase the parameters by 2^{16}. It costs $2^{43+16} = 2^{59}$ data, $2^{66.2+16} = 2^{82.2}$ time, $2^{69.2+16} = 2^{85.2}$ memory. A colliding tweak has enough collisions with probability 0.72. Overall the attack succeeds with probability 0.89.

Algorithm 8 Truncated differential attack on Beanie

1: Query 2^p random plaintexts for a fixed key K and 2^t pairs of tweaks (T^0, T^α).
2: **for all** 2^t tweaks (T^α) and every 2^{2p-1} pairs of plaintexts (P_i^α, P_j^α) **do**
3: Compute the set of first round round-keys k_0 that encrypt this pair into states inactive in positions 0,2,5,7 (resp. 1,3,4,6).
4: Store (i, j, b) at the line indexed by T^α, k_0 of a table. $b \in \{0, 1\}$ indicates which truncated trail is considered (positions 0,2,5,7 or 1,3,4,6).
5: **end for**
6: Remove all the lines that contain less than c entries.
7: Sort the table according to the list of indices (i, j, b).
8: A collision in the table gives a candidate for the keys k_0, $\bar{k_0}$ and for the pair of tweaks colliding on k_4, k_5 (and k_6 and k_7 in the 7+7-round attack).

Recovering the full key. Once we have identified a pair (T^α, T^β) of colliding tweaks and recovered $k_0, \bar{k_0}$, we can also recover $k_1, \bar{k_1}$ and $k_2, \bar{k_2}$. However, as the cipher reduces to 2+2 rounds, testing a key guess against a trial encryption does not allow to uniquely identify k_3 but only to recover $k_3 \oplus \bar{k_3}$. We do a meet-in-the middle on each SuperBox $\hat{S}$, where we test whether $\mathrm{Enc}(k_1, k_2, P_0^\alpha) \oplus \mathrm{Enc}(k_1, k_2, P_1^\alpha) = \mathrm{Enc}(\bar{k_1}, \bar{k_2}, P_0^\alpha) \oplus \mathrm{Enc}(\bar{k_1}, \bar{k_2}, P_1^\alpha)$. From this we can directly recover $k_3 \oplus \bar{k_3}$. As there are 32 bits of k_1, k_2 in each SuperBox, this step costs around 2^{33} time.

In the 5+5-round case, this is sufficient to continue: we can take a random T^γ and consider (T^α, T^γ). As we know most of the keys of T^α, this reduces to attacking 2+5 rounds, which is easy, for example using our 2+4-round yoyo distinguisher to identify the k_0 of T^γ, then our 2+2 truncated distinguisher to identify its k_1 and k_2, and end with a meet-in-the-middle.

In the 7+7-round case, the easiest approach is to look for a second pair (T^γ, T^δ) of colliding tweaks. It is possible to update the attack parameters so that 2 collisions are found with a good probability. We take the simple approach of estimating the cost will be less than doubled. Then we can attack (T^α, T^γ), which is a double encryption with only $k_3, \bar{k_3}$ unknown, at a cost of 2^{32}.

Finally, in line with the designer's claims, we assume we can recover the master key from (tweak, round-key) pairs.

6.3 Second Truncated Differential Attack

This section details how to use the second truncated differential distinguisher of Section 6.1. In a nutshell, its sparser activity pattern allows us to only guess the first round round-key in halves, while on the other hand the extra condition of partial key collision implies to query more tweaks.

Data. As we also collide on $\mathtt{MC}^{-1}(k_3)[0, 2, 5, 7]$, we query more tweaks. Otherwise this step is identical.

Pair preprocessing. For each 2^t tweaks, we compute the set of pairs inactive in 0,2,5,7 or 1,3,4,6. We expect $N \times 2^{-15}$ such pairs.

Distinguisher. We guess independently each half of k_0, which allows to compute half of X_1. Thus, the events we consider are:

- Associated with $k_0[0, 2, 5, 7]$: Activity on $X_1[0]$, on $X_1[2]$, or on $X_1[0, 2]$,
- Associated with $k_0[1, 3, 4, 6]$: Activity on $X_1[5]$, on $X_1[7]$, or on $X_1[5, 7]$.

Thus in each case the probability of matching events at random will be

$$\left(2 \underbrace{\frac{\left(2^4-1\right)^2}{\left(2^{32}-1\right)^2}}_{X_1[0] \text{ or } X_1[2]} + \underbrace{\frac{\left(2^4-1\right)^4}{\left(2^{32}-1\right)^2}}_{X_1[0,2]} + \underbrace{\left(1 - \frac{2\left(2^4-1\right) + \left(2^4-1\right)^2}{2^{32}-1}\right)^2}_{\text{Nothing happens}} \right)^N \simeq 2^{-N \times 2^{-22.47}}$$

As with Equation (1), we can require to have a minimum number to collision $\geq c$, giving the probability:

$$2^{-N\times 2^{-22.47}} - \sum_{i=0}^{c-1} \binom{N}{i} \left(\frac{2\left(2^4-1\right)^2 + \left(2^4-1\right)^4}{\left(2^{32}-1\right)^2} \right)^i \left(1 - \frac{2\left(2^4-1\right) + \left(2^4-1\right)^2}{2^{32}-1} \right)^{2(N-i)} . \tag{2}$$

Table precomputation. In a similar way to Algorithm 7, we build a table allowing to easily get for a plaintext pair (active on half a state) the possible values for (the same) half of k_0 that encrypt it after 1 round into a pair with the aimed pattern. In this variant it is sufficient to iterate over 2^{24} half-states pairs (X,X').

Collision Step. We next build a large table where each line corresponds to a tweak value and a half-key guess of k_0 and contains a list of indices indicating the plaintext pairs with the desired pattern after 1 round. We do this independently for $k_0[0,2,5,7]$ and $k_0[1,3,4,6]$.

We obtain independent candidates for $T^\alpha, T^\beta, k_0[0,2,5,7], \bar{k_0}[0,2,5,7]$ and $T^\alpha, T^\beta, k_0[1,3,4,6], \bar{k_0}[1,3,4,6]$. Merging them gives candidates T^α, T^β, k_0, $\bar{k_0}$.

If we obtain more than one candidate at this stage we can test them by partially encrypting/decrypting the pairs that have not been exploited so far to check if we obtain the same activity pattern at the input and at the output side. The attack is summarised in Algorithm 9.

Partially correct guesses. The standard assumption for wrong guesses is that their behaviour is uniformly random. This is not always true here, as some tweak pairs might collide on 3 rounds but not on k_3, and the correct guess of $k_0, \bar{k_0}$ will satisfy our first truncated distinguisher, meaning there is some correlation between the activity patterns on each side. This has, however, a negligible impact, as the probability for a pair to pass in this case is

$$\underbrace{\frac{2\left(2^4-1\right)^2 + \left(2^4-1\right)^4}{\left(2^{32}-1\right)\left(2^{16}-1\right)}}_{X_1[0],\ X_1[2] \text{ or } X_1[0,2]} + \underbrace{\frac{\left(2^{16}-1-2\left(2^4-1\right)-\left(2^4-1\right)^2\right)^2}{\left(2^{32}-1\right)\left(2^{16}-1\right)}}_{X_1[1,3,4,6] \text{ inactive but nothing is observed}} + \underbrace{\left(1-\frac{2^{16}-1}{2^{32}-1}\right)}_{X_1[1,3,4,6] \text{ active}}$$

which is approximately $2^{-2^{-22.48}}$, almost the probability of the random case.

Complexity analysis. For the 5+5-round attack:

- Parameters: $t = 41.4$ to collide on 2.5 rounds, $p = 13.6$ to have $N = 2^{26.2}$ pairs. We want at least 4 eventful plaintexts. Data is $2^{t+p} = 2^{55}$.

Algorithm 9 Second Truncated differential attack on BEANIE

1: Query 2^p random plaintexts for a fixed key K and 2^t pairs of tweaks (T^0, T^α).
2: **for all** 2^t tweaks T^α **do**
3: Identify the $2^{2p-1-15}$ plaintext pairs inactive in nibbles $0, 2, 5, 7$ or $1, 3, 4, 6$.
4: **end for**
5: **for all** 2^t tweaks T^α and 2^{2p-17} pairs of plaintexts inactive in $1, 3, 4, 6$ **do**
6: Compute the set of $k_0[0, 2, 5, 7]$ that lead to activity in $X_1[0, 2]$ only.
7: Store these results in a table, where each line is indexed by T^α, k_0 and contains a list of (i, j, x), with x telling whether $X_1[0]$ is active, $X_1[2]$ is active, or both.
8: **end for**
9: Collide on the list to obtain candidates for $T^\alpha, T^\beta, k_0[0, 2, 5, 7], \bar{k_0}[0, 2, 5, 7]$.
10: **for all** tweaks T^α part of a candidate and 2^{2p-17} pairs of plaintexts inactive in $0, 2, 5, 7$ **do**
11: Compute the set of $k_0[1, 3, 4, 6]$ that lead to activity in $X_1[5, 7]$ only.
12: Store these results in a table, where each line is indexed by T^α, k_0 and contains a list of (i, j, x), with x telling whether $X_1[5]$ is active, $X_1[7]$ is active, or both.
13: **end for**
14: Collide on the list to obtain candidates for $T^\alpha, T^\beta, k_0[1, 3, 4, 6], \bar{k_0}[1, 3, 4, 6]$.
15: Merge both lists to obtain candidates for $T^\alpha, T^\beta, k_0, \bar{k_0}$.
16: If more than one candidate survives, partially decrypt the pairs that have not been considered so far and check for matching activity patterns in X_1.

- Time: The cost is dominated by the sorting of the first list. It contains $2^{41.4+16} = 2^{57.4}$ entries, thus we estimate the cost to be $2^{61.6}$ 1-round encryptions. Applying Equation (2), the collision probability is $2^{-106.4}$, thus around $2^{8.4}$ candidates survive. The probability for a wrong key to survive the second collision is also $2^{-106.4}$, hence even without the last filtering no wrong key is expected. Overall this costs $2^{58.3}$ equivalent encryptions.
- Memory: This is dominated by the first table, which contains $2^{41.4+16}$ lines of roughly $64 + 5 \times 32$ bits. Overall this is $2^{60.2}$ memory.
- Success probability: we have $2^{41.4}$ tweaks, and a colliding tweak has at least 4 collisions with probability 0.67. Overall, there is at least one good tweak with probability $1 - \exp(-2^{25.4 \times 2 - 49} \times 0.67) \simeq 0.90$. Note that if it fails we can try again, with the same data, and seek a collision on $k_1[1, 3, 4, 6]$ instead.

For the 7+7-round attack, the only difference is that we use 2^{16} times more tweaks. Thus, this costs $2^{55+16} = 2^{71}$ data, $2^{58.3+16} = 2^{74.3}$ time. Memory is slightly larger, $2^{57.4+16}$ lines of roughly $128 + 5 \times 32$ bits, which is $2^{76.6}$. Success probability is still 0.90.

Recovering the full key. We use the same approach as in the first version. The only difference is that due to the partial collision on k_3, we only recover half of k_2. For the 5+5 version, we are missing 48 bits of keys to apply the 2+4-round yoyo distinguisher with a random tweak, thus we can do it in time 2^{48}. For the 7+7 version, the meet-in-the middle recovers k_3 and half of k_2, also in 2^{48}.

6.4 Lower-data 5+5 round variant

In this variant of the second distinguisher we consider one round of collision only ($k_5 = \bar{k_5}$) and a partial key collision over the round-keys (relation between k_4 and $\bar{k_4}$). The main difference is that we have one more round of key-recovery, so instead of computing the matching $k_0[0, 2, 5, 7]$ we compute k_0 and $k_1[0, 2, 5, 7]$.

Precomputation. We compute a table to easily access the keys that lead a given pair to the correct pattern, this time after two rounds (Algorithm 10). As there are 2^{39} pairs after two rounds and 16 relevant bits of k_1, this costs 2^{55} operations.

Algorithm 10 Precomputation of valid keys over 2 rounds

1: Initialise a table $\boldsymbol{A}$ of 2^{32} empty lists
2: **for all** $k_1[0, 2, 5, 7]$ **do**
3: **for all** unordered (X, X') s.t. $X \oplus X'$ is inactive in nibbles other than $5, 7$ **do**
4: Compute Y, Y' the two-round decryption of X and X' with $k_1[1, 3, 4, 6]$ (and the remaining key bits at 0)
5: Add $(Y, k_1[1, 3, 4, 6])$ to the list $\boldsymbol{A}[Y \oplus Y']$
6: **end for**
7: **end for** ▷ For a pair (P, P'), the valid keys are $(k_0, k_1[1, 3, 4, 6])$ given by $(Y \oplus P, k_1[1, 3, 4, 6])$ or $(Y \oplus P', k_1[1, 3, 4, 6])$ for $(Y, k_1[1, 3, 4, 6])$ in $\boldsymbol{A}[P \oplus P']$

Complexity analysis.

- Parameters: $t = 25.4$ to collide on 1.5 rounds, $p = 13.6$ to have $N = 2^{26.2}$ pairs. We want at least 4 eventful plaintexts. Data is $2^{t+p} = 2^{39}$.
- Time: The first list contains $2^{25.4+48} = 2^{73.4}$ entries, which we estimate to cost $2^{77.6}$ 1-round encryptions. The collision probability is $2^{-106.4}$, and only $2^{39.4}$ candidates are expected to survive. The next phase involves processing $2^{13.6+39.4+16} = 2^{69}$ plaintexts overall, which will be negligible. Wrong keys also pass this step with probability $2^{-106.4}$, and we expect no wrong candidate to remain. Overall this costs less than $2^{74.3}$ encryptions.
- Memory: This is dominated by the first table, which contains $2^{25.4+48}$ lines of roughly $32 + 48 + 5 \times 32$ bits. Overall this is $2^{76.4}$ 32-bits words.
- Success probability: we have $2^{25.4}$ tweaks, and a colliding tweak has at least 4 collisions with probability 0.67. Overall, there is at least one good tweak with probability $1 - \exp(-2^{25.4 \times 2 - 49} \times 0.67) \simeq 0.9$. Note that if it fails we can try again, with the same data, and seek a collision on $k_1[1, 3, 4, 6]$ instead.

Recovering the full key. We use the same approach as before: a first meet-in-the middle recovers everything but half of k_3, and then we take a random tweak, recover its k_0, k_1, k_2 with our distinguishers and end with a meet-in-the-middle.

Algorithm 11 Lower-data attack on 5+5-round BEANIE

Query 2^p random plaintexts for a fixed key K and 2^t pairs of tweaks (T^0, T^α).
for all 2^t tweaks T^α and 2^{2p-1} associated pairs of plaintexts **do**
    Get the $k_0, k_1[1,3,4,6]$ that lead to inactivity in nibbles other than $X_2[5,7]$
    Store these results in a table, where each line is indexed by T^α, k_0, k_1 and contains a list of (i,j,x), x telling whether $X_2[5]$ is active, $X_2[7]$ is active, or both.
end for
Remove from the list all candidates without enough pairs.
Collide on the list to obtain candidates for $T^\alpha, T^\beta, k_0, \bar{k_0}, k_1[1,3,4,6], \bar{k_1}[1,3,4,6]$.
for all remaining candidates **do**
    for all possible values of $k_1[0,2,5,7]$ **do**
        Find $\bar{k_1}[0,2,5,7]$ by solving $\texttt{MC}^{-1}(\bar{k_1})[0,2,5,7] = \texttt{MC}^{-1}(k_1 \oplus k_0 \oplus \bar{k_0})[0,2,5,7]$
    ▷ We collide on $\texttt{MC}^{-1}(k_4)[0,2,5,7]$, the solution is unique as MC is MDS.
        for all 2^p plaintexts of T^α and T^β **do**
            Partially decrypt to get X_2, $\bar{X_2}$.
            Sort the plaintexts and look for pairs only active in $X_2[0]$ and/or $X_2[2]$.
        end for
        If the pairs of T^α and T^β do not match, discard the guess of $k_1[0,2,5,7]$.
    end for
    If all guesses are discarded, discard the candidate.
end for
return the surviving candidates.

6.5 Attacking 7+7-round BEANIE with a Better Key Schedule

In case no directly exploitable round-key relations exist, a collision of the last 3 round-keys no longer implies $k_4 = \bar{k_4}$ so our attacks cover 1+1 rounds less. This is however not enough to thwart our attacks, as they cost less than 2^{96}: we can fully guess k_0, and then apply one of our two attacks with very slightly more data to improve the filtering: instead of a collision to find (T^α, k_0) we will collide to find (T^α, k_0, k_1). This would overall multiply the data cost by less than $2^{0.5}$, and the time cost by less than $2^{32.5}$.

7 Conclusion and Discussion

On the Design of Small-State TBC. As identified in both design [7,8,11,17] and cryptanalysis works [16,22,24,28], the main difficulty faced when designing a small-state TBC is to manage the absorbtion of a large tweak and/or key in a small state. This size discrepancy generally leads to distinct tweaks that give identical tweak-key material over a few rounds, making related-tweak attacks generically threatening. As we demonstrated in the case of BEANIE, we can still expect that by chance two distinct tweaks behave identically for a few rounds, even in cases where we do not have any control of the differences.

The small state size also makes possible to query the full codebook for a given tweak, which means as long as the tweak is controlled, there is not a large

gap between known and chosen plaintext queries. Moreover, one can precompute whole chunks of the cipher, which helps to mitigate the cost of a key-recovery.

On the U-shape model. The U-shape model forces to connect r encryption rounds with r decryption rounds.[5] If at first glance, U-shaped queries seem to double the number of rounds of a cipher from a security standpoint, it is much weaker for two reasons: each half is independent, and what were identical rounds for a standard query become cancelled rounds in a U-shaped query. Thus, in terms of security, a U-shaped query ends up closer to a restricted r-round query than a $2r$-round query.

Weaknesses of Beanie. The main issue of Beanie was its limited number of rounds; the only property necessary for our attacks is the decomposition in aligned boxes. The only clear design flaw we identified is the generation of the round-keys $k_i, i \geq 4$ in Beanie. This flaw only matters for the 7+7-round attack and is easy to fix (it suffices to change one of the derived round-keys, for example with $k_7 = k_0 \oplus k_1 \oplus k_2 (= k_4 \oplus k_2)$). Moreover, it is not required to break 7+7 rounds.

If anything, the general idea of the tweakey-schedule to have the round-keys be a permutation of the tweak is sound: if this was not the case, we could have looked for collisions on more than 4 rounds. In the worst case it would even have been possible to find pairs of tweaks with 7 rounds of cancellation (that is, the 7+7-round U-shaped query would have been $P = P \oplus k_0 \oplus \bar{k_0}$) in time around $2^{7\times 16} = 2^{112}$.

Patching Beanie. During the presentation of Beanie at FSE 2026 and as a consequence of this work, the authors updated their claims: they claim 2^{80} time and 2^{40} data for 6+6 round Beanie, and no longer make a full security claim. All our 7+7 attacks apply as-is on the 6+6 version, but do not threaten the new claim. While we achieved 2^{80} time, this is with a vastly higher amount of data.

Acknowledgments

We thank the designers of Beanie for their feedback on the early version of this document. We thank Pierre-Jean Spaenlehauer for a fruitful discussion on combinatorics. We also thank the anonymous reviewers for their helpful comments. This work has been partially supported by the French Agence Nationale de la Recherche through the OREO project under Contract ANR-22-CE39-0015, and through the France 2030 program under grant agreement No. ANR-22-PECY-0010 CRYPTANALYSE.

[5] Which in a sense is reminiscent of the reconstruction queries in Forkciphers [4].

References

1. Interoperability and performance standards for medium and high frequency radio systems. United States Department of Defense Interface Standard MIL- STD-188-141B (1999)
2. Interoperability and performance standards for medium and high frequency radio systems. United States Department of Defense Interface Standard MIL- STD-188-141D (2017)
3. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
4. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: A new primitive for authenticated encryption of very short messages. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 153–182. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34621-8_6
5. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: DLCT: A new tool for differential-linear cryptanalysis. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 313–342. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17653-2_11
6. Bariant, A., David, N., Leurent, G.: Cryptanalysis of forkciphers. IACR Trans. Symm. Cryptol. **2020**(1), 233–265 (2020). https://doi.org/10.13154/tosc.v2020.i1.233-265
7. Belkheyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: BipBip: A low-latency tweakable block cipher with small dimensions. IACR TCHES **2023**(1), 326–368 (2023). https://doi.org/10.46586/tches.v2023.i1.326-368
8. Belkheyar, Y., Derbez, P., Ghosh, S., Leander, G., Mella, S., Perrin, L., Rasoolzadeh, S., Stennes, L., Sun, S., Assche, G.V., Vizár, D.: ChiLow and ChiChi: New constructions for code encryption. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part I. LNCS, vol. 15601, pp. 212–243. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91107-1_8
9. Biham, E., Biryukov, A., Dunkelman, O., Richardson, E., Shamir, A.: Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR (invited talk). In: Tavares, S.E., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 362–376. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48892-8_27
10. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Berlin, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_14
11. Canale, F., Güneysu, T., Leander, G., Thoma, J.P., Todo, Y., Ueno, R.: SCARF - A low-latency block cipher for secure cache-randomization. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 1937–1954. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/canale
12. Che, C., Tian, T.: Differential-linear cryptanalysis and cube attacks on chilow. IACR Trans. Symmetric Cryptol. **2026**(1), 293–317 (2026). https://doi.org/10.46586/TOSC.V2026.I1.293-317, https://doi.org/10.46586/tosc.v2026.i1.293-317

13. Daemen, J., Rijmen, V.: Understanding two-round differentials in AES. In: Prisco, R.D., Yung, M. (eds.) Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4116, pp. 78–94. Springer (2006). https://doi.org/10.1007/11832072_6, https://doi.org/10.1007/11832072_6
14. Dansarie, M.: Cryptanalysis of the SoDark cipher for hf radio automatic link establishment. IACR Trans. Symm. Cryptol. **2021**(3), 36–53 (2021). https://doi.org/10.46586/tosc.v2021.i3.36-53
15. Dansarie, M., Derbez, P., Leander, G., Stennes, L.: Breaking HALFLOOP-24. IACR Trans. Symm. Cryptol. **2022**(3), 217–238 (2022). https://doi.org/10.46586/tosc.v2022.i3.217-238
16. Flórez-Gutiérrez, A., Lambooij, E., Leurent, G., Raddum, H., Tiessen, T., Verbauwhede, M.: Cryptanalysis of full SCARF. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part I. LNCS, vol. 15601, pp. 397–426. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91107-1_14
17. Gerhalter, S., Hodzic, S., Medwed, M., Nageler, M., Folwarczny, A., Nikov, V., Hoogerbrugge, J., Schneider, T., McConville, G., Eichlseder, M.: BEANIE - A 32-bit cipher for cryptographic mitigations against software attacks. IACR Trans. Symmetric Cryptol. **2025**(4), 31–69 (2025). https://doi.org/10.46586/TOSC.V2025.I4.31-69, https://doi.org/10.46586/tosc.v2025.i4.31-69
18. Grassi, L.: Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. IACR Trans. Symm. Cryptol. **2018**(2), 133–160 (2018). https://doi.org/10.13154/tosc.v2018.i2.133-160
19. Hadipour, H., Derbez, P., Eichlseder, M.: Revisiting differential-linear attacks via a boomerang perspective with application to AES, ascon, CLEFIA, SKINNY, PRESENT, KNOT, TWINE, WARP, LBlock, simeck, and SERPENT. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part IV. LNCS, vol. 14923, pp. 38–72. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68385-5_2
20. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE'94. LNCS, vol. 1008, pp. 196–211. Springer, Berlin, Heidelberg (Dec 1995). https://doi.org/10.1007/3-540-60590-8_16
21. Knuth, D.E.: The Art of Computer Programming: Sorting and Searching, volume 3. Addison-Wesley Professional (1998)
22. Lambooij, E., Neumann, P., Verbauwhede, M., Wang, S., Zhang, T.: Meet-in-the-middle attacks on full chilow. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part VI. Lecture Notes in Computer Science, vol. 16805, pp. 353–373. Springer (2026). https://doi.org/10.1007/978-3-032-35412-9_12, https://doi.org/10.1007/978-3-032-35412-9_12
23. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 17–25. Springer, Berlin, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5_3
24. Leander, G., Rasoolzadeh, S., Stennes, L.: Cryptanalysis of HALFLOOP block ciphers destroying HALFLOOP-24. IACR Trans. Symm. Cryptol. **2023**(4), 58–82 (2023). https://doi.org/10.46586/tosc.v2023.i4.58-82
25. Peng, S., Khalesi, A., Ahmadian, Z., Hadipour, H., He, J., Hu, K., Niu, Z., Rasoolzadeh, S., Wang, M.: Cube and integral attacks on chilow-32. IACR Trans. Symmetric Cryptol. **2026**(1), 253–292 (2026). https://doi.org/10.46586/TOSC.V2026.I1.253-292, https://doi.org/10.46586/tosc.v2026.i1.253-292

26. Rønjom, S., Bardeh, N.G., Helleseth, T.: Yoyo tricks with AES. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 217–243. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_8
27. Selçuk, A.A.: On probability of success in linear and differential cryptanalysis. Journal of Cryptology **21**(1), 131–147 (Jan 2008). https://doi.org/10.1007/s00145-007-9013-7
28. Wang, J., Boura, C., Derbez, P., Hu, K., Li, M., Wang, M.: Cryptanalysis of full-round BipBip. IACR Trans. Symm. Cryptol. **2024**(2), 68–84 (2024). https://doi.org/10.46586/tosc.v2024.i2.68-84

A Details of the Tweak-key Schedule of Beanie

As can be seen in Figure 9, the internal state of the tweak-key schedule is represented as two 4×4 matrices, with each cell containing a nibble. Multiple operations are applied each round on these two matrices:

AddKey: the master key is XORed with the state, along with round-dependent constants.

SubCellsT (SCT'): the same 4-bit SBox used in the data path is applied to each cell.

MixColumnsT: each of the two blocks of the state is multiplied by the matrix M' defined in the specification of the low-latency cipher PRINCE [10].

ShiftRowsT: the first row is left untouched, the second row is rotated one position to the left, the third row by two positions, and the last row by three positions.

FeistelMixT: the two matrices are decomposed into two pairs of two columns each, (X_0, X_1) and (X_2, X_3). The result is obtained by performing the following operations: $(X_1 \oplus X_0, X_2)$ and $(X_3 \oplus X_2, X_0)$.

ShiftRowsMixT (SRMT): viewing the tweak-key as a 4×8 matrix, each row is rotated to the left by 0, 2, 4, and 6 positions, respectively.

After r rounds (where r matches the number of rounds in the data path), the output is read as (k_0, k_1, k_2, k_3), providing the round tweak-keys to XOR with the data path for the first three rounds. The values for the remaining rounds are computed as follows: $k_4 = k_0 \oplus k_1$, $k_5 = k_2 \oplus k_3$, $k_6 = k_0 \oplus k_2$, $k_7 = k_1 \oplus k_3$, $k_8 = k_0 \oplus k_3$, $k_9 = k_1 \oplus k_2$.

B Details on the Yoyo Technique

B.1 Proof of Theorem 1

Proof. The definition of the exchanged pairs implies $x_1 \oplus x_2 = \tilde{x}_1 \oplus \tilde{x}_2$. By applying the first SuperBox layer (which is aligned with the words of the states and in particular these that have been exchanged), we deduce that

$$S_1(x_1) \oplus S_1(x_2) = S_1(\tilde{x}_1) \oplus S_1(\tilde{x}_2).$$

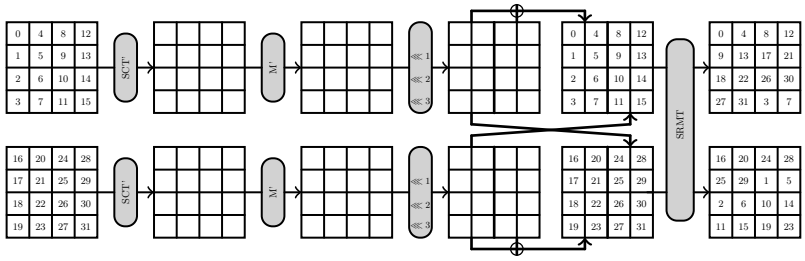


Fig. 9. One round of the tweak-key schedule of Beanie (**AddKey** is omitted for readability).

The linearity of L implies:

$$L \circ S_1(x_1) \oplus L \circ S_1(x_2) = L \circ S_1(\tilde{x}_1) \oplus L \circ S_1(\tilde{x}_2).$$

Finally, if some words aligned with the final SuperBox application S_2 are inactive in the difference $L \circ S_1(x_1) \oplus L \circ S_1(x_2)$, the equality implies the same inactivity for $L \circ S_1(\tilde{x}_1) \oplus L \circ S_1(\tilde{x}_2)$. This inactivity is preserved by the bijective S_2:

$$\nu(S_2 \circ L \circ S_1(x_1) \oplus S_2 \circ L \circ S_1(x_2)) = \nu(S_2 \circ L \circ S_1(\tilde{x}_1) \oplus S_2 \circ L \circ S_1(\tilde{x}_2)).$$

□

B.2 Naive Attack on 4+4 rounds of Beanie

As detailed in Algorithm 12, an attacker picks pairs with a specific inactivity pattern, guesses the last 2 round-keys to build exchanged pairs at the output of the $\hat{S} \circ L \circ S$ part and checks if their corresponding plaintexts have the correct Zero Difference Pattern.

Algorithm 12 Direct 4+4-round recovery of $\bar{k_0}, \bar{k_1}$ with the yoyo technique.

1: Query the full codebook for a fixed key K and a fixed pair of tweaks (T^0, T^1).
2: **for all** $\bar{k_0}, \bar{k_1}$ **do**
3: Pick a random pair of plaintexts inactive in nibbles $(0, 2, 5, 7)$.
4: Partially decrypt them using $\bar{k_0}, \bar{k_1}$, and build an exchanged pair.
5: **if** the exchanged pair is equal to the (unordered) initial pair **then**
6: Try again with another pair.
7: **end if**
8: Fetch the pair of plaintexts corresponding to the exchanged pair in the codebook.
9: Keep $\bar{k_0}, \bar{k_1}$ only if this pair of plaintexts is inactive in nibbles $(0, 2, 5, 7)$.
10: **end for**
11: Repeat 6 times to identify the correct 64-bit key.

Using 6 pairs allows to uniquely identify the correct key out of the 2^{64} possibilities, as a wrong key passes the test with probability 2^{-16}. This attack uses the full codebook over one tweak pair (2^{32} initial queries), a number of operations bounded by 2^{64} encryptions and a memory of 2^{32} blocks of 32 bits.

The same initial queries can be reused to recover k_0, k_1 by swapping input and output. Assuming we have effectively identified $k_0, \bar{k_0}, k_1, \bar{k_1}$, we can recover the missing 128-bit key material $(k_2, k_3, k_4 = k_0 \oplus k_1, \bar{k_2}, \bar{k_3}, \bar{k_4} = \bar{k_0} \oplus \bar{k_1})$ with the generic meet-in-the-middle against double-encryption, at a cost equivalent to roughly 2^{64} encryptions.

C Collision Extraction Algorithms

We identify collisions using counting sort [21]. If the sorting key space is too large, we do a partial counting sort on a truncated sorting key, do a collision detection algorithm on the subsets with matching partial sorting keys, and only if a collision is detected we sort the subset with a standard merge sort.

We will make our analysis assuming the following operations are atomic and cost 1:

- Increment/Decrement a value in a memory cell and get the outcome,
- Write a value and get the previous value in a memory cell.

Moreover, contrary to the classical analysis of sorting algorithm, we are interested in concrete average-case cost estimates.

Complexity analysis of Algorithm 13. Initialising counter costs S accesses (1 write each). The first loop does $2|L|$ accesses (1 read, 1 increment each), the second S accesses (1 increment-and-read each), and the third $3|L|$ (1 read, one decrement-and-read, one write each). Finally, the while loop performs L memory accesses (one read each), plus the number of collisions. Assuming the number of collisions is small enough, the overall cost is $2S + 6|L|$ memory accesses, and a memory cost of $|L| + S$.

Case of larger sorting key space. The above approach cannot beat generic comparison-based algorithms if the key space is too large. The approach is to use Algorithm 13 on partial sorting keys, then Algorithm 14 to detect further partial collisions among the previous partial collisions, and finally apply a standard merge sort to identify actual collisions. In the end this can be seen as a variant of bucket sort.

Complexity analysis of Algorithm 15. This algorithm is parameterisable, as we can choose S_1 and S_2. We take $S_1 \simeq L/2^8$ and $S_2 = 2^{32}$. This way, we expect to split the space in subsets of size roughly 2^8. We expect that there exists around $L/2^{24}$ collisions on s_1 and s_2, meaning that we will expect to make a merge sort for only a fraction 2^{-16} of the subsets.

Algorithm 13 Counting sort-based collision extraction

Input: An unsorted list L, a function $s : L \to \{0, \cdots, S-1\}$
Output: A list of tuples of elements of L $(x_0^j, \ldots x_i^j)$ such that for all i, i', j, $x_i^j \neq x_{i'}^j$ and $s(x_i^j) = s(x_{i'}^j)$

Collisions $\leftarrow \emptyset$
Counter $\leftarrow \{0\}^S$ ▷ S counters associated to the sorting key initialised at 0
Sorted $\leftarrow \{0\}^{|L|}$ ▷ Initialisation is not mandatory
for all $x \in L$ **do**
    Counter[$s(x)$]+=1 ▷ Detect collisions
end for ▷ There are Counter[i] values with sorting key i
for i in $\{1, \cdots, S-1\}$ **do**
    Counter[i]+= Counter[$i-1$] ▷ Position of element with key i in the sorted table
end for
    ▷ Interval Counter[$i-1$] to Counter[i]-1 can fit all inputs with sorting key i
for all $x \in L$ **do**
    Counter[$s(x)$]-=1
    Sorted[Counter[$s(x)$]] $= x$ ▷ Put x at one of the remaining free places for sorting key $s(x)$
end for
$i = 0$
while $i < |L|$ **do**
    $j = i + 1$
    while s(Sorted[i]) $= s$(Sorted[j]) **do** ▷ Identify and process batches of collisions
        Increment j
    end while
    if $j > i + 1$ **then**
        Add Sorted[$i \ldots j-1$] to Collisions
    end if
    $i \leftarrow j$
end while
return Collisions

Algorithm 14 Basic collision-detection algorithm

Input: An unsorted list L, a function $s : L \to \{0, \cdots, S-1\}$
Output: Whether L contains a collision

Counter $\leftarrow \{\text{False}\}^S$
for all x in L **do**
    if Counter[$s(x)$] is True **then**
        return True
    end if
    Counter[$s(x)$] $\leftarrow$ True
end for
return False

Algorithm 15 Counting sort-based collision extraction, large key space

Input: An unsorted list L, a function $s : L \to \{0, \cdots, S-1\}$, and two truncations of s, $s_1 : L \to \{0, \cdots, S_1 - 1\}$, $s_2 : L \to \{0, \cdots, S_2 - 1\}$
Output: A list of tuples of elements of L $(x_0^j, \ldots x_i^j)$ such that for all i, i', j, $x_i^j \neq x_{i'}^j$ and $s(x_i^j) = s(x_{i'}^j)$

Collisions $\leftarrow \emptyset$
Counter $\leftarrow \{0\}^{S_1}$ ▷ S_1 counters initialised at 0
Sorted $\leftarrow \{0\}^{|L|}$ ▷ Initialisation not mandatory
CollDetect $\leftarrow \{-1\}^{S_2}$ ▷ Initialisation not mandatory
for all $x \in L$ **do**
    Counter[$s_1(x)$]+=1 ▷ Detect s_1 collisions
end for
for i from 1 to S_1 **do**
    Counter[i]+= Counter[$i-1$] ▷ Position of element with s_1 key i in the sorted table
end for
for all $x \in L$ **do**
    Counter[$s_1(x)$]-=1
    Sorted[Counter[$s_1(x)$]] $= x$ ▷ Put x at one of the remaining free places for sorting key $s_1(x)$
end for
$i = 0$
while $i < |L|$ **do**
    $j = i + 1$
    while s_1(Sorted[i]) = s_1(Sorted[j]) **do**
        Increment j
    end while
    if $j > i + 1$ **then**
        for all $k \in \{i, \cdots, j-1\}$ **do**
            $h \leftarrow s_2$(Sorted[k])
            if CollDetect[h] $= i$ **then** ▷ s_2 Collision found
                Do a merge sort of Sorted[$i \ldots j-1$] using s
                Extract collisions to Collisions
                Break for loop
            end if
            CollDetect[h] $\leftarrow i$
        end for
    end if
    $i \leftarrow j$
end while
return Collisions

Overall the first parts cost $6L+2S_1$, as the previous algorithm. The new part costs a negligible S_2, an additional $2L$ to check the collisions on s_2, and finally an actual comparison sort on a small fraction of subsets. If we note p_i the size of the i-th bucket, this part uses around $\sum_{i:\text{Bucket } i \text{ contains a collision}} p_i \log_2(p)$ comparisons and a similar number of memory accesses. Given that only a fraction 2^{-16} of the buckets contain collisions, we expect this step to be negligible.

In the end we can approximate the cost of finding collisions to 8 memory accesses, assuming the number of collisions is negligible to the initial list size.

Implementing s for sets. We see sets as sorted lists of integer. Thus, we can implement s as a partial lexicographic order on the first elements, with some truncation if needed. This does not produce a uniform distribution, as the elements are sorted, but we can still define buckets such that the probability to lie in a bucket will be close to the expected distribution. In this case the cost is $(4 + 3C(s_1) + C(s_2))L + 2S_1$. In our cases s_1 can always be computed from the first 4 elements of the list and s_2 from 2 elements. Thus for simplicity we estimate this cost to be 18 memory accesses.

D Probability for a Tweak Pair to Collide and Pass a Distinguisher

In some attacks we need both to have an ℓ-bit collision between 2^t tweaks, which occurs with probability $\simeq 1 - \exp(-2^{2t-1-\ell}))$, and for the tweaks to satisfy a given property, which occurs with probability q. A simple analysis can estimate the overall probability to be $q(1-\exp(-2^{2t-1-\ell}))$, but this neglects the fact that, especially if one ensures a high probability of collision, we might have more than one tweak pair that collides, in which case we succeed if at least one colliding tweak pair passes. The probability to succeed and have k colliding tweaks is then

$$\binom{2^{2t-1}}{k} \left(\frac{1}{2^\ell}\right)^k \left(1 - \frac{1}{2^\ell}\right)^{2^{2t-1}-k} (1-q)^k$$

In the regimes we consider the k with non-negligible probability are small constants, meaning the overall failure probability can be approximated as

$$\sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{2^{2t-1}}{2^\ell}\right)^k \left(1 - \frac{1}{2^\ell}\right)^{2^{2t-1}} (1-q)^k \simeq \exp(2^{2t-1-\ell}(1-q)) \exp(-2^{2t-1-\ell})$$

This can be simplified to $\exp(-2^{2t-1-\ell}q)$, for a success probability in $1 - \exp(-2^{2t-1-\ell}q)$.

Remark 1 (Valid regime). In our case, this estimation no longer holds if we are in a regime in which multicollisions occur, as the events of probability q would no longer be independent.

Remark 2 (Application in our attack). With this estimation, for an overall fixed amount of data, there is a tradeoff: we can either have more tweaks (which increases t), or a higher amount of data per tweak (which increases q). Moreover, the failure probability is exponential in the number of queried tweaks, which means that a slight increase in data from an attack with constant success probability drastically decreases its failure probability.

D.1 Application to our Truncated Differentials

In our truncated differential attacks, we have an additional parameter c, that tells us how many matches we want for a tweak pair to be kept. For a fixed c we can compute the success probability with the above formula. Then, we do a numeric optimization for a fixed overall amount of data, that we can then tune to reach the desired success probability.

Then if the probability to have an event is p_e, we use

$$q = 1 - \sum_{i=0}^{c-1} \binom{N}{i} p_e^i (1-p_e)^{N-i}.$$

Remark 3. Technically, c depends on the amount of data, as it needs to be such that the probability for a wrong tweak to survive is low enough. However, in our cases, the probability for a wrong pair to match is small enough that c is not affected by small changes in data cost.

E Attack on 7+7-round Beanie based on Differential-Linear Properties

In this section, we describe a concrete use case of the generic approach of Section 4.3, consisting in using a 3+3-round distinguisher of Beanie to aim at the 7+7-round version. First, we search for a 3+3-round differential-linear distinguisher with high correlation. Next, we evaluate the cost of extending it to 7+7-round, which in turn leads to a 7+7-round key-recovery attack.

E.1 Differential-Linear Distinguishers of Beanie

Differential-linear distinguishers were introduced by Langford and Hellman [23] and consist of splitting the cipher E into two subciphers such that $E = E_1 \circ E_0$, where we have a differential characteristic $\Delta_i \xrightarrow{p} \Delta_o$ on E_0 and a linear characteristic $\lambda_i \xrightarrow{q} \lambda_o$ on E_1. In a differential-linear distinguisher, the attacker queries pairs of plaintexts $(P, P \oplus \Delta_i)$ and evaluates the correlation of the below linear combination of their corresponding ciphertexts (C, C'):

$$\lambda_i \cdot (C \oplus C') = 0.$$

The differential-linear characteristic is $\Delta_i \xrightarrow{c} \lambda_o$ and its correlation was first estimated to be $c = pq^2$. Bar-on *et al.* [5] introduced the Differential-Linear

Connectivity Table (DLCT), leading to the sandwich framework, which consists of decomposing the cipher E into three parts: $E = E_1 \circ E_m \circ E_0$. The middle part E_m aims to handle the dependencies between E_0 and E_1. In [19], Hadipour *et al.* generalised the DLCT and gave a more accurate estimate of the differential-linear correlation to $c = prq^2$, with r the correlation of the middle part.

We analyze the cipher using the sandwich framework [5], i.e., if $\mathcal{B}$ represents the Beanie cipher, we decompose it as $\mathcal{B} = E_1 \circ E_m \circ E_0$, as depicted in Figure 10. The 1+1-round of the U-shape is part of the E_m subcipher. For the subciphers E_0 and E_1, we reuse the bit-based SAT models provided by the designers of Beanie [17] to obtain, respectively, a differential trail with probability p and a linear trail with correlation q. For the middle part E_m, we construct a cell-wise Constraint Programming (CP) model that extends the input difference and the output mask of E_m probabilistically in the forward and backward directions, respectively. The input difference and the output mask of the middle part are obtained from the SAT models of E_0 and E_1, respectively. To ensure a high correlation in the middle part, we add two constraints to the CP model:

- *Minimization of the common active SBoxes:* we minimize the number of SBoxes activated by both the input difference and the output mask propagation, as in [19].
- *Minimization of the interaction between input-state SBoxes and the common output active SBoxes:* in other words, for each round and each input SBox, we count the number of common active SBoxes it affects at the end of the round. We then minimize the sum of these counters over all middle-part rounds. This new constraint yields a more hollow middle part; however, it increases the number of active SBoxes in the input and the output of the middle and thereby decreases p and q. This ad hoc heuristic is efficient on Beanie as the middle part covers almost the whole distinguisher, and we have short differential/linear trails.

The middle-part correlation r is computed experimentally. We ran the model for various configurations of r_0+r_1-round distinguishers, as depicted in Table 2.

r_0+r_1-round	Estimates		Experimental		
	[17] pq^2	prq^2	Exp. corr.	pairs	trials
2 + 2	2^{-20}	1	1	2^{20}	100
2 + 3 = ((2+2)+1)	2^{-26}	2^{-6}	$2^{-5.81}$	2^{24}	1000
3 + 3 = (1 + (2+2) + 1)	$< 2^{-32}$	2^{-13}	$2^{-11.38}$	2^{30}	10000

Table 2. r_0+r_1-round differential-linear distinguishers of Beanie. We deduced the naive estimation pq^2 from [17] where authors gave the probability (resp. squared correlation) of the best differential (resp. linear) characteristic on r_0 (resp. r_1) rounds.

E.2 A 7+7-round Differential-Linear Attack

As outlined in Section 4.3, we proceed in two steps:

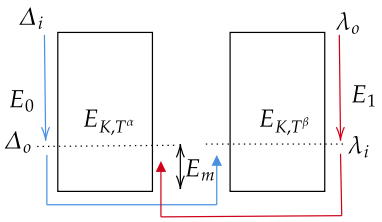


Fig. 10. Overview of a differential-linear distinguisher on Beanie.

- *Collision search:* We first use the characteristic ($\Delta = 0x30100c00 \xrightarrow{c=2^{-11.38}} \lambda = 0x90f0000d$) of our 3+3-round differential-linear distinguisher to identify potential colliding pairs of tweaks.
- *Key recovery attack:* If r candidates pass the distinguisher and given a 3+3 round attack with time complexity below $2^{128-log_2 r}$, we can try all the r candidates to run a 7+7-round attack below the security claim.

Analysis of the collision search: A query to a colliding pair of tweaks will degenerate into a 3+3-round query, and as such we expect the bias to be centred around $\varepsilon = c/2$. For a wrong pair of tweaks, we assume it will be centred around 0.

To estimate the efficiency of our distinguisher we rely on the analysis made by Selçuk in [27]. The following formula gives the number of plaintext pairs N required to reach a given success probability P_s to have the right pair of tweaks appear among the r highest correlations. We have:

$$N = \Big(\frac{\Phi^{-1}(P_s) + \Phi^{-1}(1 - 2^{-a-1})}{2\varepsilon}\Big)^2$$

where a is the advantage of the attack as defined by Selçuk [27], i.e, if we are attacking m bits (2^m candidates) then $a = m - \log_2 r$.

In our case, $m = 97, \varepsilon = c/2 = 2^{-12.38}$, for $P_s = 0.99, a = 52$, we need $N = 2^{29.55}$ pairs to have the right pair of tweaks among the 2^{45} best candidates. The collision search is described in Algorithm 16, and its time complexity t_1 is given by the cost of the encryption of the whole codebook which is $2^{49+32} = 2^{81}$ and $5 \cdot 2^{97} \cdot N$ lookups. Assuming a lookup to cost 1 round of Beanie, we have $t_1 \approx 2^{81} + 2^{97} \times 2^{29.55} \cdot \frac{5}{14} \approx 2^{125.06}$ encryptions.

Key recovery attack: Given any attack on 3+3 rounds of Beanie with a time complexity $< 2^{83}$, we can try all our 2^{45} candidates to run a 7+7-round attack below the security claim. Such attacks can be deduced from the ideas discussed in the previous sections, or we can rely on our 2+3-round differential-linear

distinguisher ($\Delta = 0xf0f00000, \lambda = 0x90f0000d, c = 2^{-5.81}$) to reach a 3+3-round attack with a success probability of 0.99, an advantage of 77 and a number of pairs of $2^{18.89}$.

Note that the 3+3-round differential-linear distinguisher used in the collision search is strongly related to the 2+3-round one used for the key-recovery (it can be seen as an extension by one round at the differential side). Thus, to avoid any unexpected behaviour from the tweak pairs that have shown biases in the first search we position the 2+3-round distinguisher the other way around from the one used in the collision search. The recovery of k_0 is detailed in Algorithm 17. Its time complexity is negligible in front of t_1.

The overall differential-linear attack complexity: The time complexity of the whole attack is $t_1 \approx 2^{125.06}$ encryptions. For the tweak collision search, we can take advantage of the query model and query the encryption of the full codebook with a fixed unknown key K and 2^{49} pairs of tweaks (T^0, T^α). Therefore, we end up with 2^{97} sets of the full codebook encrypted with the unknown key K and pair of tweaks (T^α, T^β). For the 3+3-round attack, we reuse part of this data, so the overall data complexity remains $d = 2^{49+32} = 2^{81}$. For the memory complexity, we will need about 2^{81} registers of 32 bits for the data.

Algorithm 16 Searching a colliding pair of tweaks

```
Query the full codebook for a fixed key K and 2^49 pairs of tweaks (T^0, T^α)
for all 2^97 pairs of tweaks (T^α, T^β) do
    Initiate a counter c_{α,β} to 0
    for N pairs (P_i^α, P_j^α = P_i^α ⊕ Δ) do
        Get the corresponding outputs P_i^β and P_j^β
        if λ · (P_i^β ⊕ P_j^β) = 0 then
            Increment c_{α,β}
        end if
    end for
    if c_{α,β} deviates enough from N/2 then
        Save (T^α, T^β)
    end if
end for
```

F 5+5-round attack with 4 round-keys collision

For multiple attacks in this paper, we used to our advantage collisions on round-key for 7+7 rounds. Interestingly, a similar relation occurs for 5+5 rounds: $k_5 = k_3 \oplus k_2$, which means that a collision on k_2, k_3, k_4 implies a collision on k_4. This can only be leveraged with more than 2^{40} differents tweaks, and so doesn't break security claims for 5+5-round Beanie, even if it needs less time and memory.

Algorithm 17 Key recovery attack

Input: The list of candidate pairs of tweaks $\mathcal{T}$ saved in Algorithm 16 and the associated data.
Output: colliding pair of tweaks (T^α, T^β) and its associated first-round round-key k_0.

Pick a set of $2^{18.89}$ random pairs $(X_i, X_j = X_i \oplus \Delta)$.
for all 2^{32} possible values for k_0 **do**
  Partially decrypt over one round the set of pairs (X_i, X_j) into (P_i, P_j) with k_0.
  for all candidates $(T^\alpha, T^\beta) \in \mathcal{T}$ **do**
    Initiate a counter $c_{k_0,\alpha,\beta}$ to 0
    for all (P_i, P_j) **do**
      Fetch the corresponding outputs P_i^β and P_j^β in the stored codebooks.
      if $\lambda \cdot (P_i^\beta \oplus P_j^\beta) = 0$ **then**
        Increment $c_{k_0,\alpha,\beta}$
      end if
    end for
    $c_{k_0,\alpha,\beta} = \frac{|c_{k_0,\alpha,\beta} - N/2|}{N}$ ▷ Gives the bias for each candidate
  end for
end for
return (k_0, T^α, T^β) with the highest associated bias.

With such a 4-round collision, we are left with 1+1 rounds of encryption, that has no diffusion, and is composed only of the following operations:

$$ARK(\bar{k_0}) \circ SC^{-1} \circ SR^{-1} \circ MC^{-1} \circ ARK(\bar{k_1}) \circ ARK(k_1) \circ MC \circ SR \circ SC \circ ARK(k_0)$$

Since MC is linear, the middle operations $MC^{-1} \circ ARK(\bar{k_1}) \circ ARK(k_1) \circ MC$ which is the same as xoring $M \times (k_1 \oplus \bar{k_1})$. 1+1 rounds of Beanie reduces to

$$ARK(\bar{k_0}) \circ SC^{-1} \circ SR^{-1} \circ ARK(M \times (\bar{k_1} \oplus k_1)) \circ SR \circ SC \circ ARK(k_0)$$

We deduce that for 1+1-round Beanie, an Sbox is active in output, if and only if the same Sbox is active in input. For a random permutation, this property holds with a probability of $\frac{(2^4-1)^s}{2^{32}-1}$, where s is the number of active Sboxes.

We can mount an attack on 1+1-round Beanie using this property. We divide tweaks into different sets such that each tweak in a set can have a collision only with another tweak in the same set. We compute the different collision sets recursively. For each step, for each tweak, we consider two plaintexts, we compute their truncated difference, among the $2^8 - 1$ possible, and we store each tweak in the corresponding collision set. We should then have $2^8 - 1$ collision sets, and for each of these new sets, we take another pair of plaintext, compute its truncated difference, and put it in the new collision set. After two steps, we have up to $(2^8 - 1)(2^8 - 1)$ possible collision set, and so on, until enought steps have been made to find a collision.

The global idea for Algorithm 18 is instead of computing the truncated differences for each plaintexts pair for each tweak once and search for two or more

tweaks having the exact same truncated differences, we do it in a recursive manner, one pair of ciphertexts at a time. For each ciphertext pair tried, we will divide our tweaks into new possible collision sets, where two tweaks are in the same collision set if and only if for each pair of ciphertext tested before, they have the exact same truncated differences. The algorithm is illustrated in Figure 11.

Algorithm 18 Identification of a 4-round collision of 5+5 BEANIE

Input: A : a list of tweaks with corresponding n plaintexts $(T_\alpha, P_1^\alpha, \ldots, P_n^\alpha)$
Initialise $\mathbf{P} = \{A\}$, a table containing a list of possible collision sets, starting with all the tweaks in the same collision set.
for all i, j such that $0 < i < j \leqslant n$ **do**
    Initialize an empty temporary table $\mathbf{B} = \{\}$ containing list of possible collision sets after the next for loop
    for all $Old_{Coll_{Set}} \in \mathbf{P}$ **do**
        Initialize a new array of lists of tweaks candidates $New_{Coll_{Sets}}$ with a size of $2^8 - 1$
        for all $T \in Old_{Coll_{Set}}$ **do**
            Store T_α in $New_{Coll_{Sets}}$ at the line corresponding to the truncated difference of P_i^α and P_j^α
        end for
        Append each list of $New_{Coll_{Sets}}$ that has at least two tweaks to $\mathbf{B}$
    end for
    $\mathbf{P} \leftarrow \mathbf{B}$
end for
return P

For an efficient implementation of Algorithm 18, we propose to store each tweak into a single table, and to form sets of collision set, we sort the table by the truncated difference of P_i^α and P_j^α, and we use a counting sort when the size of the current set size is $\gg 2^8$, In this case, the time complexity is $2S + 6|L|$, where $S = 2^8 - 1$, and $|L|$ the number of elements in input. The memory complexity is $S + |L|$. As $|L| \gg S$, the time complexity is roughly $6|L|$ When we don't have $|L| \gg 2^8$, we propose to use Algorithm 19. As we only need to group tweaks according to their truncated difference, we only need that the colliding tweaks are stored in the same set.

Complexity analysis for Algorithm 19:

We analyze it in the context of Algorithm 18. Notably, we can allocate the table T only once, and to "re-allocate" the table, it suffices to set the size of each stack to 0 (one write each, in total, S writes).

For the first for loop, we have to read each element of L one time, the size of the associated stack one time, and push up to two elements in two different stacks, so up to 4 additional write. We have $2|L|$ reads and $4|L|$ writes for the first loop.

For the second for loop, we read each element of P, which has in the worst case $\min(|L|, S)$ elements. For the inner for loop, the size of all $T[i]$ is $|L|$ so we add

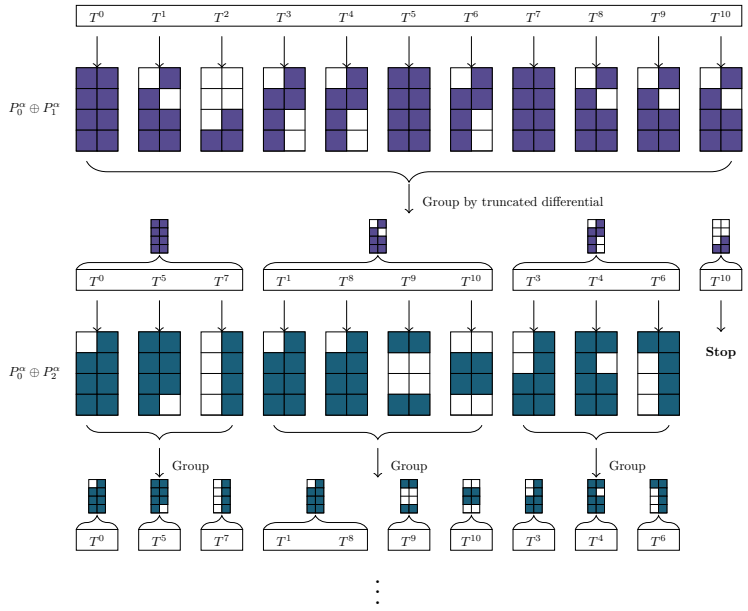


Fig. 11. Exemple of two iterations of Algorithm 18, for 11 tweaks. After the two iterations, a collision is possible only on tweaks T^1 and T^8

Algorithm 19 Partial Sort

Input: An unsorted array L, a function $s : L \to \{0, \ldots, S-1\}$
Output: A partially sorted array L, and a list of indexes I such that all elements x in L stored between the indexes $(I[i], I[i+1])$ are equals, $I[i+1]$ excluded

Allocate $|L|S$ memory for an array T composed of S stacks of dimension $|L|$
Allocate S memory for a stack P
for all $x \in L$ **do**
    $s = size(T[s(x)])$
    if $s == 0$ **then**
        $push(P, s(x))$
    end if
    $push(T[s(x)], x)$
end for
$n_t \leftarrow 0$
Allocate $size(P)$ memory for a stack I, and push 0 to it
for all $i \in P$ **do**
    for all $x \in T[i]$ **do**
        $L[n_t] \leftarrow x$
        increment n_t
    end for
    $push(I, size(T[i]))$
end for
return L, I

L reads followed by $|L|$ writes. We finish by pushing up to $\min(|L|, S)$ to I, for a cost of $2 \times \min(|L|, S)$. In our use case we reset our table T, adding $\min(|L|, S)$ writes (one for each element of P). $\min(|L|, S) + L$ reads, and $L + 4 \times \min(|L|, S)$ writes. In total, we have $\min(|L|, S) + 3L$ reads, and $5L + 4 \times \min(|L|, S)$ writes. In our use case, we can simplify the formula by replacing $\min(|L|, S)$ by $|L|$, giving us $4|L|$ reads and $9|L|$ writes, totalizing a complexity of $13|L|$.

For the memory, we have $2^8 \times |L|$ required for the table T. However, as stated before, each table is already allocated beforehand. As this algorithm is meant to be used when we don't have$|L| \gg 2^8$, its size is negligible compared to the number of tweaks we need.

Complexity analysis. In order to analyse the complexity of our attack, we assume that for each tweak, the plaintexts are independent. Moreover, we assume that all nibbles of each plaintexts are independent.

The principles are similar to the attacks from Section 6.3, but at the nibble level. Each nibble has to be an independent permutation, which means that equalities at the input nibbles have to be reflected in the output nibbles. For each tweak we can define a set of relations (for example, $\{P_1^T = P_4^T = P_8^T, P_5^T = P_9^T\}$), and this set must match for correct tweak pairs. For a fixed set of relations with k equalities and p queries, there are $\frac{16!}{(16-p+k)!}$ possible functions. Moreover, the number of relations with k equalities correspond to the Stirling numbers of the second kind, $\left\{ {p \atop n-k} \right\}$. Overall, the probability for to random tweaks to collide on a single nibble for p plaintexts is

$$\sum_{k=0}^{p-1} \underbrace{\left(\frac{16!}{(16-p+k)!16^p} \right)^2}_{\text{proba to have a fixed relation set with } k \text{ equalities}} \times \left\{ {p \atop n-k} \right\}.$$

Two tweaks collide if they match for all 8 nibbles. With $p = 12$, the probability for a random tweak pair to collide on all 8 nibbles is $2^{-112.5}$. If we take 2^{49} tweaks, they will contain a tweak collision 86% of the time. Then, with 2^{97} pairs, we do not expect any wrong pair to collide.

Now that we have showed that we don't need more than 11 plaintexts per tweak, we compute its complexity.

We need $12 \times 2^{49} \simeq 2^{52.6}$ plaintexts. For the memory, we need to store 12 32-bit plaintexts per tweak, to which we had a pointer to these plaintexts, and a temporary value used to store the truncated differential.

Assuming 64-bit pointers, we need $(12 \times 32 + 64 + 8)2^{49} \simeq 2^{57.8}$ bits, (i.e. $2^{52.8}$ 32-bit words). Additional memory usage required for the algorithm is negligible.

Concerning time complexity, we need to compute for each $\frac{p(p-1)}{2} = 66$ pairs of plaintexts their xor, needing 3 memory access (one for the pointer giving the index of the tweak T and its associated plaintext, two for the plaintexts), as well as sorting each collision set. As there is 12×2^{49} tweaks, for each given pair of tweaks, we have 12×2^{49} elements divided into multiple collisions set. Using a counting sort and the Algorithm 19 to sort the sets when their size is small

enough, sorting by $(P_i^T \oplus P_j^T)$ cost at most $13|L|$. The time complexity is at most $(13+3) \times 66 \times 2^{49} \simeq 2^{59}$.

To conclude the attack, once we have found a pair of colliding tweaks, we do 8 exhaustive search of 12-bits, each time extracting a nibble on k_0, $k_1 \oplus \bar{k_1}$ and $\bar{k_0}$.