Tuple Cryptanalysis: Slicing & Fusing Multisets*

Marine Minier [‡], Raphaël C.-W. Phan [†]

[‡] Université de Lorraine, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, F-54506, *France*
Inria, Villers-lès-Nancy, F-54600, *France*
CNRS, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, F-54506, *France*
[†] Faculty of Engineering, Multimedia University (MMU),
Cyberjaya, *Malaysia*

Abstract. In this paper, we revisit the notions of Square, saturation, integrals, multisets, bit patterns and tuples, and propose a new Slice & Fuse paradigm to better exploit multiset type properties of block ciphers, as well as relations between multisets and constituent bitslice tuples. With this refined analysis, we are able to improve the best bounds proposed in such contexts against the following block ciphers: Threefish, PRINCE, PRESENT and RECTANGLE.

Keywords: Block ciphers, Square, saturation, integrals, multisets, bit patterns, tuples, bitslice, Slice & Fuse paradigm, division property.

Received 1 May 2016, Revised 25 October 2016, Accepted 2 November 2016.

1 Introduction

Cryptanalysis based on a *multiset* of related texts aims for some property of the multiset to pass through cipher components with a probability that is furthest away as possible from the uniform distribution. Given a bijective function $S : \{0,1\}^w \rightarrow \{0,1\}^w$, then with a multiset $\mathcal{M}$ of 2^w elements such that $\forall x_j, x_{j'} \in \mathcal{M}$, $x_j \neq x_{j'}$ (so-called a multiset with the permutation property P, or called an active set), we have that $\forall x_j, x_{j'} \in \mathcal{M}$, $S(x_j) \neq S(x_{j'})$, i.e. such a P multiset passes through S without having its property changed. Given such a P, its presence can be detected by checking if the exclusive-OR ($\bigoplus$) sum of all elements of this multiset equals zero. This was first observed by Knudsen in 1997 [3] as part of the analysis of the SQUARE cipher, hence subsequently it was de facto known in the cryptographic community as the *Square attack*.

The term multiset was in fact first used for this cryptanalysis context by Lucks [9], within a so-called *saturation attack* framework. This naming convention makes sense, because essentially, only the active set (only later known as the permutation P set) is a proper set, other types e.g. the even (E) set, the passive (also called constant C) set, are actually multisets because any distinct value could appear more than once in such sets. The saturation term is named after the so-called saturated property, which is used to denote a dth-order P multiset. Furthermore, the semi-saturated property was used to refer to a multiset with 2^{w-1} unique values instead of 2^w. Lucks noted that when viewed in terms of separate bit positions, one such bit position is fixed to a constant value while the other $w-1$ are allowed to vary. To our knowledge, this was the first mention of viewing multisets in terms of their bitwise channels.

* *Paper C2, pre-proceedings of Mycrypt 2016.*

Knudsen & Wagner later proposed a formalisation of this type of cryptanalysis within the group theoretic setting as the *Integral cryptanalysis* [11], focussing on how the $\bigoplus$ sum of a P multiset could propagate through different cipher components; presenting ways to determine such an integral sum for multisets of different types i.e. C, P and those whose integral sum equals some fixed value.

Meanwhile, Biryukov & Shamir [10] initiated a formal study of *multiset calculus* by considering how multiple words of multisets propagate through the substitution and affine layers of a cipher, i.e. the notion of multiple multiset words $\mathcal{M}_1 \dots \mathcal{M}_\ell$ *composing* to a multi-word multiset. This type of analysis is crucial as a multiset input in any word of a block eventually spreads to other words through the diffusing properties of the affine layers. Nakahara Jr et al. [13] at Mycrypt 2005 built on this notion by focussing on how an n-bit P word multiset $\mathcal{M}$ can be *decomposed* into its constituent subword multisets $\mathcal{V}_i$ each of length $w < n$. Z'aba et al. [17] took this decomposition notion further by focussing on constituent *bitslice* multisets i.e. $w = 1$. Indeed, when the cardinality of a bitslice multiset $\mathcal{V}$ is more than two, i.e. $|\mathcal{V}| > 2$, the elements of the bitslice multiset are then seen to form a bit pattern. Using such bit patterns that are either constant c, regularly alternating a_i or non-alternating b_i, they were able to trace how such patterns behave through the exclusive-OR (XOR) operation and through the Sbox layer.

Aumasson et al. [22] emphasized on the internal ordering of the elements within a multiset, therefore such ordered multisets are better known as *tuples*. Keeping track of ordered tuple elements is useful in order to better trace the effects of operations among elements of different multisets, as well as be exploited to cancel out differences among elements of different tuples.

More recently, Todo [34, 35] generalized the integral cryptanalysis approach by taking the integral sum on the low-order polynomial subsets of all the elements in the output multisets, such that due to the higher-order differential type property linked to the algebraic normal form (ANF) representation of the cipher component, a zero sum is obtained; so-called the *division property*. The approach was then formalised by Boura and Canteaut [46] in relation to Reed-Muller codes, based on parity computation across different multiple bits of each multiset element, which is related to the ANF representation of functions and the algebraic degree. In contrast, our proposed approach considers each bitslice channel independently and the focus is on the internal ordering of such bitslice elements rather than on their integral sum.

The Slice-&-Fuse Paradigm. In this paper, we put forth a refined multiset calculus, wherein we propose new types of bit*slice* tuples to better represent the rich internal structures of multiset constituents, analyse the behaviour of such tuples through different cipher components including exclusive-OR, addition, AND and Sboxes, and discuss how these constituent bitslice tuples can be recomposed (*fused*) to form structured multisets.

We demonstrate such tuple formulations on the ciphers Threefish, PRINCE, PRESENT and RECTANGLE. For Threefish, more structured multiset properties are detected analytically in contrast to best-known results in [22], while for PRINCE we are able to extend the multiset tracking by one more round compared to previous work [44]. For PRESENT and RECTANGLE, we need much smaller multisets (thus less data complexity) and/or are able to detect a sum property in more number of output bits compared to the literature. Moreover, as side results, in Appendix A we also improve the previous

integral attacks against CRYPTON and mCrypton.

2 Multiset Calculus

2.1 Multiset & Bitslice Channels

A w-bit *multiset*

$$\mathcal{M} = \{x_j\}_{j=0}^{2^w-1} = \beta_{w-1} \dots \beta_0$$

is a collection of 2^w w-bit word elements. The multiset can also collectively be viewed as a concatenation of w bitslice channels β_i, $i = 0, \dots, w-1$, one for each bit position within the word; and where each channel comprises 2^w bit elements. The bit elements within these bitslice channels will follow some internal ordering, and thus for the rest of this paper, we will use the term *bit tuples* (these have also been called bit patterns [17]) to refer to such ordered bitslice channels.

By definition, the elements $\{x_j\}$ within a multiset can take the same w-bit values, therefore it is at times useful to explicitly denote, using the multiset notation, the multiplicity/frequency $m(x_j)$ of element values. Thus a multiset can alternatively be expressed as a set of unordered pairs of the form $\{x_j, m(x_j)\}$, where each distinct value of x_j is taken from a so-called root set $\mathcal{R}$. Obviously, $|\mathcal{M}| \geq |\mathcal{R}|$.

Definition 1 (Multiset Properties): Depending on the values of its elements, a multiset is said to have (one or more of) any of the following properties:

- P: $\forall x_j, x_{j'} \in \mathcal{M}$, $x_j \neq x_{j'}$; thus $m(x_j) = m(x_{j'}) = 1$.
  This is known as a permutation multiset, where all elements are distinct.
- C: $\forall x_j \in \mathcal{M}$, $x_j = c$ for some constant value c; thus $m(c) = 2^w$.
  This is called a constant multiset, where all elements equal some constant c.
- E^k: $\forall x_j \in \mathcal{M}$, $m(x_j) \mod k = 0$, where k is of the form 2^n.
  For simplicity reasons, E^2 will be denoted by E. This is an even multiset, where each element value from $\mathcal{R}$ appears an even number of times in $\mathcal{M}$.
- B: $\bigoplus_{j=0}^{2^w-1} x_j = 0$.
  Multisets with this property are said to be balanced. Essentially, this property is detected by the existence of a zero integral $\bigoplus$ sum.
- A: $\sum_{j=0}^{2^w-1} x_j = 0$.
  This is the additive variant of the $\bigoplus$ zero sum.
- F: $\sum_{j=0}^{2^w-1} x_j = 2^{w-1}$.
  This is similar to A except that the sum is non-zero.
- Q: $\forall x_j \in \mathcal{M}, \forall x'_j \in \mathcal{M}'$, $\bigoplus_{j=0}^{2^w-1} x_j = \bigoplus_{j=0}^{2^w-1} x'_j$.
  This equality property exists for two multisets when their respective sums are equal.

It has been observed [22] that these multiset properties are essentially of two categories:

- those characterized by the multiplicity of the elements: P,C,E
- those characterized by relations among the elements: B,A,F,Q

Definition 2 (Bit Tuples): A bitslice channel β can be (one or more of) any of the following bit tuples:

- c: a contiguous sequence of all '0' or all '1' bits, i.e. $c \in \{00\ldots00, 11\ldots11\}$.
- a_i: alternating segments of length 2^i.
  e.g. $a_1 = \langle 00\ 11\ 00\ 11\ldots00\ 11\rangle$.
- $\hat{a}_i$: the segment dual of a_i such that their segment boundaries are out of sync by half of the segment length.
  e.g. $\hat{a}_1 = \langle 01\ 10\ 01\ 10\ 01\ 10\ldots10\rangle$ or $\langle 10\ 01\ 10\ 01\ 10\ 01\ldots01\rangle$.
  Note. For any a_i where $i \neq 0$, its segment dual $\hat{a}_i$ is palindromic.
- $\tilde{a}_i$: the cyclic variant(s) of a_i such that their segment boundaries are out of sync. Note that $\hat{a}_i$ is a special case of $\tilde{a}_i$.
  e.g. Given $a_2 = \langle 0000\ 1111\ 0000\ 1111\ldots0000\ 1111\rangle$, then we could have $\tilde{a}_2 = \langle 000\ 1111\ 0000\ 1111\ldots0000\ 1111\ 0\rangle$ or $\tilde{a}_2 = \langle 0\ 1111\ 0000\ 1111\ldots0000\ 1111\ 000\rangle$
- p_{ij}: palindromic dual to a_i with alternating flipped pattern segments of length 2^j. See Example 1(a).
- $\hat{p}_{ij}$: the segment dual of p_{ij} such that their segment boundaries are out of sync by half of the segment length. See Example 1(a).
- c_{ij}: complement dual to $\hat{a}_i$ with alternating flipped pattern segments of length 2^j. *Note.* Observe that c_{ij} is to $\hat{a}_i$ what p_{ij} is to a_i. See Example 1(b).
- m_{ij} (resp. s_{ij}): masked dual of a_i where every other contiguous sequence of 2^j bits is masked to '0' (resp. set to '1').
  e.g. $m_{02} = \langle 0000\ 0101\ 0000\ 0101\ldots\rangle$ and $s_{02} = \langle 1111\ 0101\ 1111\ 0101\ldots\rangle$.
- z_j: a bit tuple of pattern cycle length 2^{j+1}, that begins with contiguous '0' bits followed by 2^{j-1} '1' bit(s).
  e.g. $z_2 = \langle 00000011\ 00000011\ \ldots00000011\rangle$.
- f_j: a bit tuple with alternating flipped pattern segments each of length 2^j.
  e.g. $f_3 = \langle 00101011\ 11010100\ \ldots00101011\ 11010100\rangle$.
- l_j: a bit tuple with alternating pattern segments each of length 2^j comprising contiguous runlengths of '0's and '1's.
  e.g. $l_2 = \langle 0111\ 0111\ \ldots0111\rangle$ or $l_2 = \langle 0001\ 0001\ \ldots0001\rangle$.
- e: unlike the above tuples where bit elements within the tuple conform to some defined ordering, the e is just used to denote the property of a bit tuple such that its bit elements appear an even number of times.

Note that a_i and c were defined in [17]. The other bit tuple definitions are new and will be useful later when we trace how different bit tuples are changed by the various cipher operations e.g. ARX (addition, rotation, exclusive-OR) and AND.

Example 1(a): For bit tuple $a_0 = \langle 01010101\ldots01\rangle$, its palindromic duals p_{0j} include:

- $p_{01} = \langle 01\ 10\ 01\ 10\ \ldots01\ 10\rangle$
- $p_{02} = \langle 0101\ 1010\ 0101\ 1010\ldots0101\ 1010\rangle$
- $p_{03} = \langle 01010101\ 10101010\ 01010101\ 10101010\ldots01010101\ 10101010\rangle$
- $p_{04} = \langle 0101010101010101\ 1010101010101010\ \ldots0101010101010101\ 1010101010101010\rangle$.
- ...

and the corresponding segment duals $\hat{p}_{0j}$ of these p_{0j}, $j \neq 1$ are:

- $\hat{p}_{02} = \langle 01\ 1010\ 0101\ 1010\ldots0101\ 1010\ 01\rangle$
- $\hat{p}_{03} = \langle 0101\ 10101010\ 01010101\ 10101010\ldots01010101\ 10101010\ 0101\rangle$.
- $\hat{p}_{04} = \langle 01010101\ 1010101010101010\ \ldots0101010101010101\ 1010101010101010\ 01010101\rangle$.
- ...

Example 1(b): For the bit tuple $\hat{a}_1 = \langle 0\ 11\ 00\ 11\ 00\ \ldots\ 00\ 11\ 00\ 11\ 0\rangle$, then its complement duals can be:

- $c_{12} = \langle 0110\ 1001\ 0110\ 1001 \ldots\ 0110\ 1001\ 0110\ 1001\rangle$
- $c_{13} = \langle 01100110\ 10011001\ 01100110\ 10011001 \ldots 01100110\ 10011001\rangle$
- $c_{14} = \langle 0110011001100110\ 1001100110011001\ 0110011001100110\ 1001100110011001 \ldots$ $0110011001100110\ 1001100110011001\rangle$
- ...

Note. An a_i and any of its palindromic duals p_{ij} differ in half of their bits. Similarly, an $\hat{a}_i$ and any of its complement duals c_{ij} also differ in half of their bits. Since a_i and $\hat{a}_i$ are e (even), therefore their respective duals p_{ij} and c_{ij} are also e.

Definition 3 (Segment Length): The length ℓ_s of a segment within a multiset or bit tuple is defined as the number of times that a unique element value is repeated contiguously within the defined ordering of the multiset/tuple. Note that for a multiset to have property P, its ℓ_s needs to be 1.

Definition 4 (Pattern Cycle): The cycle of a multiset or bit tuple is defined as the point when a pattern of elements repeats. The number of elements traversed before this occurs is known as the cycle length ℓ_c. Note that for a multiset $\mathcal{M}$ to have property P, its ℓ_c needs to equal $|\mathcal{M}|$.

Note. A w-bit multiset $\mathcal{M}$ is said to have property P if its segment length $\ell_s = 1$ and its cycle length $\ell_c = |\mathcal{M}|$. □

Example 2: Consider a 3-bit multiset, thus comprising 8 elements:

$$\mathcal{M} = \langle 000, 001, 010, 011, 100, 101, 110, 111\rangle$$

The above multiset is said to have the permutation property P as all elements are distinct, each of multiplicity 1. $\mathcal{M}$ has segment length $\ell_s = 1$ and cycle length $\ell_s = |\mathcal{M}| = 8$. Observe that the multiset also has property B and F.

Denote $\mathcal{M} = \beta_2\beta_1\beta_0$. The bitslice channel β_0 of $\mathcal{M}$ corresponding to the least significant bit (LSB) position of any element of $\mathcal{M}$ is an a_0 bit tuple, while β_1 and β_2 are respectively of the form a_1 and a_2. The segment and cycle lengths of these bitslice channels are:

- $\ell_s(a_0) = \ell_s(\langle 01010101\rangle) = 1$, $\ell_c(a_0) = \ell_c(\langle 01\ 01\ 01\ 01\rangle) = 2$,
- $\ell_s(a_1) = \ell_s(\langle 00\ 11\ 00\ 11\rangle) = 2$, $\ell_c(a_1) = \ell_c(\langle 0011\ 0011\rangle) = 4$,
- $\ell_s(a_2) = \ell_s(\langle 0000\ 1111\rangle) = 4$, $\ell_c(a_2) = \ell_c(\langle 00001111\rangle) = 8$. □

Multiset Equivalence. Note that our definition of the bit tuple is cyclic. For instance, if the elements in the multiset of Example 2 were permuted to be:

$$\mathcal{M}' = \langle 001, 010, 011, 100, 101, 110, 111, 000\rangle$$

the bit tuples are still considered to be of the form $a_2a_1a_0$. We say that two multisets are equivalent if they have the same property, up to cyclic shift as in the previous

example. For two multisets that are not equivalent in this sense, there is a need to distinguish between them. For this purpose, we can define more precisely the P property as it could be mapped into two different bit tuples that are not equivalent.

Definition 5 (Refined P Property): We refine the P property as follows:

- P_{ord}: the property P_{ord} corresponds to a multiset that could be written as the composition of bit tuples of the form $\ldots a_2 a_1 a_0$.
- P_{nord}: the property P_{nord} corresponds to a multiset that could not be written as the composition of bit tuples of the form $\ldots a_2 a_1 a_0$.

When required, we precise if P is of the form P_{ord} or P_{nord}.

Slicing a Multiset. To our knowledge, Nakahara Jr. et al. [13] at Mycrypt 2005 were the first to consider viewing a w-bit multiset in terms of the properties of its constituent v-bit ($v < w$) channels; i.e. *slicing* a multiset. This may allow to better trace how multiset properties propagate through cipher components.

For instance, removing any bitslice channel from the 3-bit P multiset $\mathcal{M}$ of Example 2 results in a 2-bit multiset that is no longer P but rather is E. Thus, we could directly state that any v-bit ($v < w$) subset $\mathcal{V}$ of a w-bit P multiset $\mathcal{M}$ cannot be P but must be E.

2.2 Fusing the Slices of a Multiset

Decomposing a w-bit multiset $\mathcal{M}$ into its constituent bitslice channels or some v-bit ($v < w$) subset $\mathcal{V}$ could allow to better track the changes in its properties through cipher round operations. After going through round operations, it is helpful to evaluate if the multiset retains its original properties e.g. P is preserved. For this, we need to recompose the subset $\mathcal{V}$, i.e. *fuse* some concatenation of bitslice channels back into the w-bit multiset $\mathcal{M}$ and to infer if a multiset property at the w-bit level is preserved or if a new multiset property is obtained.

The most structured property that has the ability to go through round operations is the property P, thus it is interesting to study its constituent bitslice channel tuples.

From Example 2, we see that bit tuples of the form $a_{i_1} a_{i_2} \ldots a_{i_w}$, for any pairwise indices $i_j \neq i_k$ $(j, k \in \{1, \ldots, w\})$, compose to a word multiset of property P. We are interested in other constituent bit tuples that compose to a P.

Example 3. To motivate the idea, consider the multiset $\mathcal{M}$ as below:

$$\mathcal{M} = \langle 011, 111, 101, 010, 001, 100, 000, 110 \rangle$$

At first glance, such a multiset has no obvious bit tuple form. To facilitate extracting its bit tuples, some prior reordering is helpful before bitslicing. We introduce some notation to keep track of the elementwise permutations ρ, to enable subsequent unwinding to the element ordering of the original, i.e. $\rho(\mathcal{M})$. For instance, permute $\mathcal{M}$ according to (1 3 4) and we get a multiset $\rho(\mathcal{M})$ with property P, of the form $a_0 a_2 a_2$:

$$\rho(\mathcal{M}) = \langle 010, 111, 011, 101, 001, 100, 000, 110 \rangle$$

This example highlights that a P can be formed by a_i's that may have the same index, e.g. in this case, there are two a_2s. However, from closer inspection, these two a_2s are not identical in form, in fact their segment boundaries are out of sync.

We define a pair of a_is whose segment boundaries are out of sync by half of the segment length, as *segment duals* of each other, denoted as $\langle a_i, \hat{a_i} \rangle$. Note that such pairwise duals are equal in half of their elements. In fact, when concatenated, these two bitslice channels form 2-bit elements that occur an even number of times.

With this in place, the above permuted multiset $\rho(\mathcal{M})$ is actually of the form $a_0 a_2 \hat{a_2}$.

It is worth to investigate what property is obtained when fusing (recomposing) two dual bit tuples to form a 2-bit multiset $\mathcal{V}$. Continuing from the above example, $\mathcal{V} = a_2 \hat{a_2}$ gives:

$$\langle 10, 11, 11, 01, 01, 00, 00, 10 \rangle$$

which is a 2-bit multiset with elements of segment length $\ell_s = 2$, and cycle length $\ell_c = 8$.

Since its cycle length is already 8, what remains is to cause its segment length to reduce to 1, in order for this multiset $\mathcal{V}$ to be used to form a multiset $\mathcal{M}$ of property P. To do this, we append another bitslice channel such that the resultant segment length ℓ_s reduces to 1.

Consider appending a bit tuple a_0 (note that this has segment length $\ell_s = 1$) to $\mathcal{V}$, thus we can get either $a_0 a_2 \hat{a_2}$, $a_2 a_0 \hat{a_2}$ or $a_2 \hat{a_2} a_0$, as follows:

$$\begin{array}{lll} 010 & 100 & 100 \\ 111 & 111 & 111 \\ 011 & 101 & 110 \\ 101 & 011 & 011 \\ 001 & 001 & 010 \\ 100 & 010 & 001 \\ 000 & 000 & 000 \\ 110 & 110 & 101, \end{array}$$

all of which are multisets of property P. Notice that appending such a bit tuple a_0 has caused the resultant segment length ℓ_s to become 1.

Consider instead, to append the bit tuple $\hat{a_1}$ (note that this has segment length $\ell_s = 2$) to $\mathcal{V}$, but such that its segment boundary is out of sync with the segment boundary of $\mathcal{V}$. Thus we get either $\hat{a_1} a_2 \hat{a_2}$, $a_2 \hat{a_1} \hat{a_2}$ or $a_2 \hat{a_2} \hat{a_1}$:

$$\begin{array}{lll} 110 & 110 & 101 \\ 111 & 111 & 111 \\ 011 & 101 & 110 \\ 001 & 001 & 010 \\ 101 & 011 & 011 \\ 100 & 010 & 001 \\ 000 & 000 & 000 \\ 010 & 100 & 100, \end{array}$$

all of which are multisets of property P. Notice that appending such a bit tuple, though of segment length $\ell_s = 2$, yet due to the out of sync in segment boundaries, causes the resultant segment length ℓ_s to be halved, i.e. reduces to 1. $\square$

Example 4. Consider another example 3-bit P multiset:

$$\mathcal{M}' = \langle 011, 111, 110, 010, 001, 101, 100, 000 \rangle$$

which has the form $\hat{a_1} a_2 a_1$. Focussing on the concatenation of the two dual bit tuples $\hat{a_1} a_1$ gives:

$$\langle 01, 11, 10, 00, 01, 11, 10, 00 \rangle$$

which is a 2-bit multiset with elements of segment length $\ell_s = 1$, and cycle length $\ell_c = 4$. As the segment length ℓ_s is already 1, what remains is to cause the cycle length ℓ_c to increase to $|\mathcal{M}'| = 8$, in order for the resultant multiset to have the P property. Therefore, the only appropriate bit tuple to append to this would be of the form a_2 which has cycle length $\ell_c = 8$. This is why the example $\mathcal{M}'$, which is of the form $\hat{a_1} a_2 a_1$, is a P. □

We now have the notations, definitions and criteria for representing, slicing (decomposing) and fusing (recomposing) multisets. These are crucial in order to facilitate the tracking of multiset & bit tuple properties through cipher rounds.

2.3 Tuples through Cipher Operations

In this section, we analyze the propagation of bitslice tuples through some main primitive operations commonly used in block ciphers and hash functions; notably the exclusive-OR (XOR), the AND operation, and substitution boxes (Sboxes).

Tuples through XOR. The emphasis we place here is on constituent tuples obtained from slicing the multisets that have property P, since this has the richest structure that can survive through round operations better than multisets of other properties. Therefore it is vital that we understand what happens to P inputs after going through XOR; the most complex being P ⊕ P.

As a P multiset comprises bitslice tuples of the form a_j, then any P ⊕ P will cause the following types of XOR between its a_j (or cyclic variants $\hat{a}_j, \tilde{a}_j$) bitslice tuples:

- $a_j \oplus a_j = c$
- $a_j \oplus \hat{a}_{j;\, j \neq 0} = a_{j-1}$
- $a_j \oplus a_{j-1} = \hat{a}_j$
- $a_i \oplus a_{j;\, j > i+1} = p_{ij}$
- $a_i \oplus \hat{a}_{j;\, j > i+1} = \hat{p}_{ij}$
- $\hat{a}_i \oplus a_{j;\, j > i+1} = c_{ij}$
- $a_j \oplus \hat{a}_{j-1} = \hat{p}_{(j-2)\ j}$

- $a_j \oplus \tilde{a}_{j;\, j \neq 0,1} = \ell_j$
- $a_i \oplus \tilde{a}_{j;\, j > i} = f_j$
- $\tilde{a}_i \oplus a_{j;\, j > i} = f_j$
- $\tilde{a}_i \oplus \tilde{a}_{j;\, j > i} = f_j$
- $\hat{a}_i \oplus \tilde{a}_{j;\, j > i} = f_j$
- $\tilde{a}_i \oplus \hat{a}_{j;\, j > i} = f_j$

Note that these properties are more refined than the ones observed in [17], and better allow to retain the internal rich structures within a P.

Furthermore, the following XOR relations can also be observed between other types of bitslice tuples:

- $a_j \oplus p_{(j-2)(j-1)} = \hat{p}_{(j-2)\ j}$
- $a_j \oplus p_{ik;\ i,k<j}$: this leads to the complement dual of p_{ik}, i.e. p_{ik} but with alternating flipped pattern segments of length 2^j

Note. All the above bit tuples have the bit tuple property e.

Tuples through AND. As the AND operation serves as one of the primitives in recent ciphers such as PRINCE, SIMON, and SIMECK, as well as when Sbox output bits are expressed in ANF representation, we consider bit tuple propagations through this AND operation.

In more detail, we analyze the bit tuple interactions involving one or two a_j:

- $c \wedge a_j = a_j$ or $0 \ldots 0$ (i.e. the all '0' sequence)
- $a_j \wedge a_j = a_j$
- $a_j \wedge a_{j-1} = z_j$
- $a_i \wedge a_{j;j>i+1} = m_{ij}$
  Intuitively, this leads to a bit tuple that equals a_i except that every other contiguous sequence of length 2^j corresponding to '0' bits in a_j contains bits masked to '0'

Note. All the above output bit tuples have even parity. Furthermore, their segment and/or pattern cycles are based on a_i, therefore their pattern boundaries are aligned with the segment boundaries (if any) of the a_i's.

Tuples through Sboxes. Any output bit of an Sbox can be viewed as the output of a coordinate Boolean function constituting the Sbox, and can be expressed in ANF representation, thereby relating each output bit as a function of input bits.

While the ANF representation of Sboxes has been exploited in the literature to track the development of the algebraic degrees of Sbox outputs, in this paper we propose the novel approach of using the ANF to enable the tracking of how bit tuples propagate through an Sbox. Essentially, from the ANF, it can be observed that any Sbox output bit is the result of ANDing and then XORing the input bits. Therefore, the problem of tracking how bit tuples behave as they pass through an Sbox can be reduced to the problem of tracking how bit tuples propagate through AND and XOR operations.

Details of this approach of tracking bit tuples through Sboxes appear in the later subsections §3.3 and §3.4 on the PRESENT and RECTANGLE ciphers.

3 Multiset Properties through Ciphers

3.1 Threefish & ARX

To concisely exemplify our approaches, notably in terms of the slicing of multisets and analysing their constituent bitslice tuples, we will first consider here the MIX operation within the round function of the Threefish block cipher [21], before moving on the other ciphers. MIX is defined as follows:

$$\mathsf{MIX}(x, y) = \langle x + y, (x + y) \oplus (y \ggg r) \rangle.$$

Note that this MIX function is essentially of the ARX form, i.e. addition (ADD), rotation (ROT), and exclusive-OR (XOR).

The multisets' behaviours as they propagate through the Addition, XOR and Rotation operations are shown in Table 1 as reported in [22]. Our aim here is to derive more internal structures and corresponding properties within these output multisets, based on our formulations of multisets and their bitslice channels.

Table 1. Truth tables of Addition (ADD), XOR, and Rotation (ROT) on multisets as reported in [22].

$+$	A	B	C	E	F	P
A	A	X	A	X	F	F
B	X	X	X	X	X	X
C	A	X	C	E	F	P
E	X	X	E	X	X	X
F	F	X	F	X	A	A
P	F	X	P	X	A	A

$\oplus$	A	B	C	E	F	P
A	X	X	X	X	X	X
B	X	B	B	B	X	B
C	X	B	C	E	X	P
E	X	B	B	B	X	B
F	X	X	X	X	X	X
P	X	B	P	B	X	B

$\ggg$	A	B	C	E	F	P
n	X	B	C	E	X	P

Multisets through Rotation (ROT). We commence by looking at the simplest operation in terms of influence on multiset properties, i.e. the rotation (ROT), also denoted by $\ggg$.

To see why ($\ggg$P) still gives P, consider its bitslice channels, e.g. for a $\mathsf{P} = a_2a_1a_0$, then we have: $(\ggg a_2a_1a_0) \rightarrow a_0a_2a_1 = \mathsf{P}$.

Multisets through Addition (ADD). In [22] it was shown that $\mathsf{P} + \mathsf{P} \rightarrow \mathsf{A}$ based on detecting a zero sum with respect to modulo addition. Alternatively, using bitslice tuples enables to show why we get the property: $\mathsf{P}+\mathsf{P} = a_2a_1a_0 + a_2a_1a_0 \rightarrow a_1a_0c = \mathsf{E}$.

More importantly, crucial to the analysis of multisets through MIX is the behaviour of $\mathsf{C} + \mathsf{P}$. W.l.o.g. consider a multiset P_1 comprising the bit tuples $a_2a_1a_0$, e.g. whose elements are ordered ascendingly. It is known that when added to a C, this results in a P_2 where the same ordering is preserved. However, if we focus on the constituent bitslice tuples, we see that these tuples may no longer have segment boundaries aligned with those of a_2, a_1, a_0, i.e. its bitslice tuples would be from the set $\in \{a_j, \hat{a}_{j;j\neq 0}, \tilde{a}_{j;j\neq 0,1}\}$.

Multisets through Exclusive-OR (XOR). We consider the XOR of different types of P, or with A, as typically encountered when propagating multisets through ARX constructions, e.g. the MIX function of Threefish; in terms of the constituent bitslice channels of such multiset types. Without loss of generality, we describe the analysis with respect to 3-bit multisets for better clarity.

- $\mathsf{P} \oplus \mathsf{P}$: $a_2a_1a_0 \oplus a_2a_1a_0 \rightarrow ccc = \mathsf{C}$.
  This case considers the XOR of P multisets comprising the same bitslice tuples. The next case considers the XOR of P multisets comprising different orderings of bitslice tuples caused by rotation of P.
- $\mathsf{P} \oplus (\ggg\mathsf{P})$: $a_2a_1a_0 \oplus a_0a_2a_1 = \langle a_2 \oplus a_0, a_1 \oplus a_2, a_0 \oplus a_1\rangle \rightarrow p_{02}\hat{a}_2\hat{a}_1 = \mathsf{E}$. It could be generalized to E^k for larger P sets.

- $\mathsf{P}_1 \oplus \mathsf{P}_2$: here we utilize the tuples formulation of this paper to show why in some cases, where different types of P structures can be precise in their bitslice tuples, that the XOR of two P can still produce a P. It remains an open problem whether all types of P can be similarly defined, including P_{nord} which is not captured in our tuples formulation. In our earlier work i.e. [22] §2.2, it was stated with an example that this was possible for non-trivial cases of P, though no detailed analysis could be provided as to why this behaviour exists because we lacked the formulations to precisely detail different types of P. It was also mentioned in §3.2 that properties observed from empirical results were stronger than analytical predictions because tracking analytically was difficult. By using the tuples formulation, one can answer the question as to why a P could be empirically detected after $\mathsf{P}_1 \oplus \mathsf{P}_2$ although state-of-the-art integral and multiset analysis techniques were to date not able to explain why it is a P. To show that we can get a P, consider $\mathsf{P}_{ord,1} = a_2a_1a_0$ and $\mathsf{P}_2 = a_0\hat{a}_2a_1$, then $\mathsf{P}_{ord,1} \oplus \mathsf{P}_2 = a_2a_1a_0 \oplus a_0\hat{a}_2a_1 = \langle a_2 \oplus a_0, a_1 \oplus \hat{a}_2, a_0 \oplus a_1\rangle = p_{02}a_2\hat{a}_1 = \mathsf{P}_3$. This results in a permutation P multiset because the segment length ℓ_s of the multiset is 1 due to the tuple p_{02} while the cycle pattern length $\ell_c = 8$ due to the tuples a_2 and p_{02}.
- $\mathsf{A} \oplus \mathsf{P}$: $a_1a_0c \oplus a_2a_1a_0 = \langle a_1 \oplus a_2, a_0 \oplus a_1, c \oplus a_0\rangle \rightarrow \hat{a}_2\hat{a}_1a_0 = \mathsf{P}$, for the specific type of A that can be formulated as a_1a_0c e.g. when it is produced from $\mathsf{P} + \mathsf{P}$. Note that such an A is also an E. To see this why this XOR results in a P, we can recall our discussions on fusing in subsection 2.2, notably the segment length ℓ_s of this multiset is 1, while its cycle length ℓ_c is 8 equaling the size of this multiset.

Table 2 summarizes the revised truth table for XOR based on our analysis in this subsection.

Table 2. Revised truth table of XOR on multisets based on tuple formulations.

$\oplus$	A	B	C	E	F	P
A	X	X	X	X	X	P/X
B	X	B	B	B	X	B
C	X	B	C	E	X	P
E	X	B	B	B	X	P/B
F	X	X	X	X	X	X
P	P/X	B	P	P/B	X	P/E/C

Multisets through MIX. With the enriched multiset properties based on bitslice tuples as discussed above, we can more precisely trace the following multiset propagations through the MIX function of Threefish, w.l.o.g. consider 3-bit multisets for simplicity of description:

$$\mathsf{MIX}(\mathsf{C}, \mathsf{P}) = \langle \mathsf{C} + \mathsf{P}, (\mathsf{C} + \mathsf{P}) \oplus (\ggg \mathsf{P})\rangle = \langle \mathsf{P}, \beta_2\beta_1\beta_0\rangle$$

Consider a P multiset of the form $a_2a_1a_0$, thus $(\ggg \mathsf{P}) = a_0a_2a_1$. From our above discussion in the subsection for the case of multisets through ADD, we see that $\mathsf{C}+\mathsf{P} \rightarrow \mathsf{P} = \alpha_2\alpha_1a_0$ where $\alpha_2 \in \{a_2, \hat{a}_2, \tilde{a}_2\}$, $\alpha_1 \in \{a_1, \hat{a}_1\}$. Therefore, we can see that the output bit tuples $\beta_2\beta_1\beta_0$ would form the multiset $(\mathsf{C} + \mathsf{P}) \oplus (\ggg \mathsf{P})$ as follows:

- $\beta_2 = \alpha_2 \oplus a_0 =$
  - $a_2 \oplus a_0 = p_{02}$, or
  - $\hat{a}_2 \oplus a_0 = \hat{p}_{02}$, or
  - $\tilde{a}_2 \oplus a_0 = f_2$.
- $\beta_1 = \alpha_1 \oplus a_2 =$
  - $a_1 \oplus a_2 = \hat{a}_2$, or
  - $\hat{a}_1 \oplus a_2 = \hat{p}_{02}$.
- $\beta_0 = a_0 \oplus a_1 = \hat{a}_1$.

These results have been corroborated by experiments on up to 8-bit multisets. Examples of an output multiset $(\mathsf{C} + \mathsf{P}) \oplus (\ggg \mathsf{P}) = \beta_2\beta_1\beta_0$ with such properties are as follows:

- $\beta_2 = p_{02} = 0101\ 1010$ or $\hat{p}_{02} = 01\ 1010\ 01$ or $f_2 = 1011\ 0100$
- $\beta_1 = \hat{a}_2 = 00\ 1111\ 00$ or $\hat{p}_{02} = 01\ 1010\ 01$
- $\beta_0 = \hat{a}_1 = 01100110$

Thus, we can deduce more structure through MIX at its output, answering the question left partially open in previous work i.e. [22] where it was stated that dependencies between tuples of words were difficult to track analytically, after having remarked that there was a difference in precision between what could be empirically observed and what could be analytically predicted.

In addition, for the cases where the input multiset going into MIX is $\langle \mathsf{P}, \mathsf{P} \rangle$, e.g. when via chosen-ciphertext attacks such structures are input to MIX, we can obtain:

$$\mathsf{MIX}(\mathsf{P}, \mathsf{P}) = \langle \mathsf{P} + \mathsf{P}, (\mathsf{P} + \mathsf{P}) \oplus (\ggg \mathsf{P}) \rangle = \langle \mathsf{E}, \mathsf{P}/\mathsf{E} \rangle$$

rather than the $\langle \mathsf{A}, \mathsf{B} \rangle$ previously observed. To see this, first note that for the left half of the output multiset, as per the subsection discussion in the case of multisets through Addition (ADD), we have $(\mathsf{P} + \mathsf{P}) \rightarrow \mathsf{E} = a_1 a_0 c$. For the right half $(\mathsf{P} + \mathsf{P}) \oplus (\ggg \mathsf{P})$ of the multiset, recall that $(\ggg \mathsf{P}) = a_0 a_2 a_1$. Thus we have the right output multiset as:

$$\langle a_1 \oplus a_0, a_0 \oplus a_2, c \oplus a_1 \rangle = \langle \hat{a}_1, p_{02}, a_1 \rangle = \mathsf{P}.$$

An example of when the right half of the output multiset becomes E is where $\mathsf{P} = a_3 a_2 a_1 a_0$, such that $(\mathsf{P} + \mathsf{P}) \oplus (\ggg \mathsf{P}) = (a_2 a_1 a_0 c) \oplus (a_0 a_3 a_2 a_1) = \langle a_2 \oplus a_0, a_1 \oplus a_3, a_0 \oplus a_2, c \oplus a_1 \rangle = \langle p_{02} p_{13} p_{02} \hat{a}_1 \rangle = \mathsf{E}$, since the same bit tuple p_{02} is present twice in the multiset, with the same boundary alignment. Alternatively, one could obtain a P multiset for other parameters, e.g. for different rotation amounts, for instance $(\mathsf{P} + \mathsf{P}) \oplus (\ggg_2 \mathsf{P}) = (a_2 a_1 a_0 c) \oplus (a_1 a_0 a_3 a_2) = \langle a_2 \oplus a_1, a_1 \oplus a_0, a_0 \oplus a_3, c \oplus a_2 \rangle = \langle \hat{a}_2 \hat{a}_1 p_{03} a_2 \rangle = \mathsf{P}$, where $\ggg_r$ denotes rotation to the right by r amounts.

These examples show that one could analytically deduce better properties for MIX(P,P), filling the gap between what could be observed via analysis and what was actually observed via experiments.

3.2 PRINCE

PRINCE is an SPN block cipher with a particular involutive structure for low latency [25]. Its 64-bit block can be represented as a 4×4 state of nibbles, which goes through initial keying, 5 forward SPN rounds, an unkeyed middle layer, and then 5 more backward SPN rounds before final keying.

A forward SPN round mainly consists of an Sbox layer S acting on nibbles and a diffusion layer $M = SR \circ M'$, before a typical keyed XOR operation; where SR is the AES ShiftRows operation and M' is an involutive operation acting on independent columns at a time. [44] observed that M' can be expressed in terms of bitwise equations as follows. For the leftmost and rightmost columns, the output nibbles (one in each row below, and each row comprising its constituent four bits) are of the form:

$$\begin{array}{llll}
y_0^0 = x_1^0 \oplus x_2^0 \oplus x_3^0 & y_0^1 = x_0^1 \oplus x_2^1 \oplus x_3^1 & y_0^2 = x_0^2 \oplus x_1^2 \oplus x_3^2 & y_0^3 = x_0^3 \oplus x_1^3 \oplus x_2^3 \\
y_1^0 = x_0^0 \oplus x_1^0 \oplus x_2^0 & y_1^1 = x_1^1 \oplus x_2^1 \oplus x_3^1 & y_1^2 = x_0^2 \oplus x_2^2 \oplus x_3^2 & y_1^3 = x_0^3 \oplus x_1^3 \oplus x_3^3 \\
y_2^0 = x_0^0 \oplus x_1^0 \oplus x_3^0 & y_2^1 = x_0^1 \oplus x_1^1 \oplus x_2^1 & y_2^2 = x_1^2 \oplus x_2^2 \oplus x_3^2 & y_2^3 = x_0^3 \oplus x_2^3 \oplus x_3^3 \\
y_3^0 = x_0^0 \oplus x_2^0 \oplus x_3^0 & y_3^1 = x_0^1 \oplus x_1^1 \oplus x_3^1 & y_3^2 = x_0^2 \oplus x_1^2 \oplus x_2^2 & y_3^3 = x_1^3 \oplus x_2^3 \oplus x_3^3
\end{array}$$

On the other hand, the output nibbles of the inner two columns are:

$$\begin{array}{llll}
y_0^0 = x_0^0 \oplus x_1^0 \oplus x_2^0 & y_0^1 = x_1^1 \oplus x_2^1 \oplus x_3^1 & y_0^2 = x_0^2 \oplus x_2^2 \oplus x_3^2 & y_0^3 = x_0^3 \oplus x_1^3 \oplus x_3^3 \\
y_1^0 = x_0^0 \oplus x_1^0 \oplus x_3^0 & y_1^1 = x_0^1 \oplus x_1^1 \oplus x_2^1 & y_1^2 = x_1^2 \oplus x_2^2 \oplus x_3^2 & y_1^3 = x_0^3 \oplus x_2^3 \oplus x_3^3 \\
y_2^0 = x_0^0 \oplus x_2^0 \oplus x_3^0 & y_2^1 = x_0^1 \oplus x_1^1 \oplus x_3^1 & y_2^3 = x_0^2 \oplus x_1^2 \oplus x_2^2 & y_2^3 = x_1^3 \oplus x_2^3 \oplus x_3^3 \\
y_3^0 = x_1^0 \oplus x_2^0 \oplus x_3^0 & y_3^1 = x_0^1 \oplus x_2^1 \oplus x_3^1 & y_3^2 = x_0^2 \oplus x_1^2 \oplus x_3^2 & y_3^3 = x_0^3 \oplus x_1^3 \oplus x_2^3
\end{array}$$

The middle layer is of the form: $S^{-1} \circ M' \circ S$, while a backward SPN round comprises the functions M^{-1} and S^{-1}.

Since PRINCE rounds essentially comprise the Sbox layer and the diffusion layer M, the entire cipher consists of only AND and exclusive-OR (XOR) operations, thus it is an AND-XOR structured cipher.

Some previous results on integral and bitslice tuple tracking through PRINCE have been proposed in [28, 44]. More precisely, in [28, 44] a classical integral property was presented as follows: with one active nibble (i.e. 2^4 chosen plaintexts) is transformed into a balanced property on each nibble after rounds of the form SM, SM, $SR^{-1}M'SR$, $M^{-1}S^{-1} \simeq SM$, SM, SR^{-1}, S^{-1}, which is seen effectively as 2.5 rounds since SR^{-1} offers little diffusion. This first-order integral property could be extended by one additional round SM at the beginning using a 4th-order integral property requiring 2^{16} chosen plaintexts. Another 1st-order integral was presented in [44] needing 2^4 texts, covering the rounds SM,SM',$S^{-1}M^{-1}$, which can be considered effectively as 2.5 rounds since M' does not provide full diffusion compared to M. Moreover, a bitslice tuple property with three particular active bits (i.e. requiring 2^3 chosen plaintexts) was also proposed for the rounds of the form SM, SM' in [44], which is effectively at most seen to be 2 rounds of PRINCE.

In fact, we found the following bitslice tuple property on 3 rounds of the form SM, SM and SM:

$$\begin{array}{cccc}
cccc & cccc & cccc & ccca_2 \\
cccc & cccc & cccc & ccca_1 \\
cccc & cccc & cccc & cccc \\
cccc & cccc & cccc & ccca_0
\end{array}$$

gives after three rounds of SM, SM and SM:

$$\begin{array}{cccc} eeee & eeee & eeee & eeee \\ eeee & eeee & eeee & eeee \\ eeee & eeee & eeee & eeee \\ eeee & eeee & eeee & eeee \end{array}$$

This property stays true for triplets of bits placed on the 3 least significant bits of the Sbox output.

In more detail, this property starts with an input multiset as follows:

$$\begin{array}{cccc} c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,a_2 \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,a_1 \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\;\;c \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,a_0 \end{array}$$

- S in R1: $\rightarrow$

$$\begin{array}{cccc} c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_2\;\alpha_2\;\alpha_2\;\alpha_2 \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_1\;\alpha_1\;\alpha_1\;\alpha_1 \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & c\;\;c\;\;c\;\;c \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_0\;\alpha_0\;\alpha_0\;\alpha_0 \end{array}$$

  After the S in round 1 (R1), all output bits of the nibble for which one of the inputs received an a_j tuple become the α_j tuple, where $\alpha_j \in \{a_j, c\}$.

- M' in R1: $\rightarrow$

$$\begin{array}{cccc} c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_{01}\;\;\alpha_{02}\;\;\alpha_{012}\;\;\alpha_{12} \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_{12}\;\;\alpha_{01}\;\;\alpha_{02}\;\;\alpha_{012} \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_{012}\;\;\alpha_{12}\;\;\alpha_{01}\;\;\alpha_{02} \\ c\,c\,c\,c & c\,c\,c\,c & c\,c\,c\,c & \alpha_{02}\;\;\alpha_{012}\;\;\alpha_{12}\;\;\alpha_{01} \end{array}$$

  where α_{ijk} is shorthand for $\alpha_i \oplus \alpha_j \oplus \alpha_k$.

- SR in R1: $\rightarrow$

$$\begin{array}{cccc|cccc|cccc|cccc} c & c & c & c & c & c & c & c & c & c & c & c & \alpha_{01} & \alpha_{02} & \alpha_{012} & \alpha_{12} \\ c & c & c & c & c & c & c & c & \alpha_{12} & \alpha_{01} & \alpha_{02} & \alpha_{012} & c & c & c & c \\ c & c & c & c & \alpha_{012} & \alpha_{12} & \alpha_{01} & \alpha_{02} & c & c & c & c & c & c & c & c \\ \alpha_{02} & \alpha_{012} & \alpha_{12} & \alpha_{01} & c & c & c & c & c & c & c & c & c & c & c & c \end{array}$$

- S in R2: $\rightarrow$

$$\begin{array}{cccc} c\,c & c & c & c\,c & c & c & c\,c & c & c & v_{42} \\ c\,c & c & c & c\,c & c & c & v_{42} & c\,c & c & c \\ c\,c & c & c & v_{42} & c\,c & c & c & c\,c & c & c \\ v_{42} & c\,c & c & c & c\,c & c & c & c\,c & c & c \end{array}$$

where v_{ij} denotes a nibble of i bits taking only j values.

- M' in R2: $\rightarrow$

$$\begin{array}{cccc}
c\ \alpha_0\ \alpha_0\ \alpha_0 & \alpha_1\ c\ \alpha_1\ \alpha_1 & \alpha_2\ c\ \alpha_2\ \alpha_2 & c\ \alpha_3\ \alpha_3\ \alpha_3 \\
\alpha_0\ c\ \alpha_0\ \alpha_0 & \alpha_1\ \alpha_1\ c\ \alpha_1 & \alpha_2\ \alpha_2\ c\ \alpha_2 & \alpha_3\ c\ \alpha_3\ \alpha_3 \\
\alpha_0\ \alpha_0\ c\ \alpha_0 & \alpha_1\ \alpha_1\ \alpha_1\ c & \alpha_2\ \alpha_2\ \alpha_2\ c & \alpha_3\ \alpha_3\ c\ \alpha_3 \\
\alpha_0\ \alpha_0\ \alpha_0\ c & c\ \alpha_1\ \alpha_1\ \alpha_1 & c\ \alpha_2\ \alpha_2\ \alpha_2 & \alpha_3\ \alpha_3\ \alpha_3\ c
\end{array}$$

- SR in R2: $\rightarrow$

$$\begin{array}{cccc}
c\ \alpha_0\ \alpha_0\ \alpha_0 & \alpha_1\ c\ \alpha_1\ \alpha_1 & \alpha_2\ c\ \alpha_2\ \alpha_2 & c\ \alpha_3\ \alpha_3\ \alpha_3 \\
\alpha_1\ \alpha_1\ c\ \alpha_1 & \alpha_2\ \alpha_2\ c\ \alpha_2 & \alpha_3\ c\ \alpha_3\ \alpha_3 & \alpha_0\ c\ \alpha_0\ \alpha_0 \\
\alpha_2\ \alpha_2\ \alpha_2\ c & \alpha_3\ \alpha_3\ c\ \alpha_3 & \alpha_0\ \alpha_0\ c\ \alpha_0 & \alpha_1\ \alpha_1\ \alpha_1\ c \\
c\ \alpha_1\ \alpha_1\ \alpha_1 & c\ \alpha_2\ \alpha_2\ \alpha_2 & \alpha_3\ \alpha_3\ \alpha_3\ c & \alpha_0\ \alpha_0\ \alpha_0\ c
\end{array}$$

- S in R3: $\rightarrow$

$$\begin{array}{cccc}
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e \\
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e \\
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e \\
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e
\end{array}$$

  To see this, note from our previous discussion of tuple propagations through the XOR operation, that the above bit tuples input to the Sbox S are of the form:
  - $\alpha_{01} = \alpha_0 \oplus \alpha_1 \in \{c \oplus c, a_0 \oplus c, c \oplus a_1, a_0 \oplus a_1\} = \{c, a_0, a_1, \hat{a}_1\}$
  - $\alpha_{12} = \alpha_1 \oplus \alpha_2 \in \{c \oplus c, a_1 \oplus c, c \oplus a_2, a_1 \oplus a_2\} = \{c, a_1, a_2, \hat{a}_2\}$
  - $\alpha_{02} = \alpha_0 \oplus \alpha_2 \in \{c \oplus c, a_0 \oplus c, c \oplus a_2, a_0 \oplus a_2\} = \{c, a_0, a_2, p_{02}\}$
  - $\alpha_{012} = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \in \{c, a_0, a_1, a_2, \hat{a}_1, \hat{a}_2, p_{02}, \hat{p}_{02}\}$.

  All these possible bit tuples are even (e) tuples. Therefore, as S is bijective, the output bit tuples will also be e. These will remain as e after the subsequent M layer.

- M in R3: $\rightarrow$

$$\begin{array}{cccc}
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e \\
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e \\
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e \\
e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e & e\ e\ e\ e
\end{array}$$

  Therefore, after 3 rounds of PRINCE, we have that all the output bits give a zero integral $\bigoplus$ sum.

Note that there exist many such properties positioned at different places of the block with the three bits aligned in a column and for different combinations of three rounds. Note that this property leads to be able to build a distinguisher with only 2^3 plaintexts on 3 rounds SM, SM and SM.

Moreover, and due to the structure of the PRINCE matrix, we could transform the 4-th order integral property on 4 rounds described in [28] requiring 2^{16} chosen plaintexts

into a 3-th order integral property on 4 rounds requiring only 2^{12} chosen plaintexts as noticed in [45]. Indeed:

$$\begin{array}{cccc} \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^4 \end{array}$$

is computed, inverting one round, from:

$$\begin{array}{cccc} \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^{12} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^{12} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^{12} \end{array}$$

Thus, this Integral property on 4 rounds could be extended by one more round at the beginning leading to a 5 rounds distinguisher with 2^{48} chosen plaintexts remarking that

$$\begin{array}{cccc} \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^{12} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^{12} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{A}^{12} \end{array} \text{ is obtained from } \begin{array}{cccc} \mathsf{A}^{48} & \mathsf{C} & \mathsf{A}^{48} & \mathsf{A}^{48} \\ \mathsf{A}^{48} & \mathsf{C} & \mathsf{A}^{48} & \mathsf{A}^{48} \\ \mathsf{A}^{48} & \mathsf{C} & \mathsf{A}^{48} & \mathsf{A}^{48} \\ \mathsf{A}^{48} & \mathsf{C} & \mathsf{A}^{48} & \mathsf{A}^{48} \end{array}$$

and do not required the whole codebook but only 2^{48} chosen plaintexts. Thus, we are able to construct an Integral attack on 7 rounds with a time complexity of about 2^{48} encryptions to recover half of the key improving the state-of-the-art concerning the best Integral attack on PRINCE.

3.3 PRESENT

PRESENT [15] is a popularly analysed lightweight cipher, published in 2007 and having been cited for over 1000 times by the end of 2015. PRESENT has a block size of 64 bits, and consists of 31 rounds, where each round is simply involving an XOR with a round key, a layer S comprising 16 parallel applications of a 4×4 Sbox and a bit permutation layer L such that the output bits of each Sbox spread uniformly to bit locations which are 16 bits apart. We can thus view PRESENT as being an NX structured cipher, similar to PRINCE.

For our purpose, we will exploit the ANF of the Sbox outputs of PRESENT, which is listed as follows [30]:

$$y_3 = 1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \quad (1)$$

$$y_2 = 1 \oplus x_2 \oplus x_3 \oplus x_0x_1 \oplus x_0x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \quad (2)$$

$$y_1 = x_1 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \quad (3)$$

$$y_0 = x_0 \oplus x_2 \oplus x_3 \oplus x_1x_2 \quad (4)$$

Note that from the above, the algebraic degree of the coordinate Boolean functions of the PRESENT Sbox is 2 for the LSB y_0 and 3 for the rest.

We begin with a multiset of 2^4 elements such that the 4 rightmost bitslices collectively form a 4-bit P word, which the other bitslices are c. For compactness of notation, we denote by c^s the contiguous sequence of s bitslices each of which is a bit tuple of the form c.

- Input: c^{16}, c^{16}, c^{16}, $c^{12}a_3a_2a_1a_0$

- S in R1: $\rightarrow c^{16}$, c^{16}, c^{16}, $c^{12}a_3a_2a_1a_0$
  The bit tuples propagate unchanged through the Sbox layer in Round 1.

- L in R1: $\rightarrow c^{15}a_3$, $c^{15}a_2$, $c^{15}a_1$, $c^{15}a_0$
  The linear layer L in Round 1 moves the a_i bit tuples ($i \in \{0, \ldots, 3\}$) to other bitslice positions, such that there is one a_i bit tuple in each 16-bit word state of PRESENT.

- S in R2: $\rightarrow c^{12}\alpha_3^3a_3$, $c^{12}\alpha_2^3a_2$, $c^{12}\alpha_1^3a_1$, $c^{12}\alpha_0^3a_0$
  where $\alpha_i \in \{c, a_i\}$
  Remark. On input c^3a_i, the output from the Sbox is $\alpha_i^3a_i$, where each α_i is either a c tuple (all constant '0's or constant '1's), or a_i including its bitwise complement. This can be seen by analyzing the ANF of the Sbox, noting that in this case only the input bit x_0 varies as per the bit tuple sequence, while the other bits $x_1x_2x_3$ are constants. Therefore y_0 will be x_0 or its complement $\overline{x_0}$, thus will be an a_i tuple.

- L in R2: $\rightarrow (c^3\alpha_3\ c^3\alpha_2\ c^3\alpha_1\ c^3\alpha_0)^3\ c^3a_3\ c^3a_2\ c^3a_1\ c^3a_0$
  The linear layer causes each 4-bit input to the Sbox in the next layer to be of the form $c^i\alpha_i$ or c^3a_i.

- S in R3: $\rightarrow (\alpha_3^4\ \alpha_2^4\ \alpha_1^4\ \alpha_0^4\ )^3\ \alpha_3^3a_3\ \alpha_2^3a_2\ \alpha_1^3a_1\ \alpha_0^3a_0$
  As per the above analysis for S in $R2$, we have discussed how an input c^3a_i propagates through the Sbox. What remains is to analyze the propagation of the input $c^3\alpha_i$ through the Sbox. Recall that $\alpha_i \in \{c, a_i\}$, thus an input $c^3\alpha_i$ is in fact the union of two types of inputs, i.e. $c^3a_i \cup c^3c$. The output will therefore be $\alpha_i^3a_i$ or c^4.

- L in R3: $\rightarrow (\alpha_3\alpha_2\alpha_1\alpha_0)^{15}\ a_3a_2a_1a_0$
  The linear layer permutes the bit tuples such that every Sbox input in the next layer is of the form $\alpha_3\alpha_2\alpha_1\alpha_0$ except for the rightmost Sbox whose input is $a_3a_2a_1a_0$.

- S in R4: $\rightarrow (e^4)^{15}\ a_3a_2a_1a_0$
  An $\alpha_3\alpha_2\alpha_1\alpha_0$ input to an Sbox comprises four bitslices α_i, each of which could be a c or a_i tuple. This means that each bitslice is either (in the case that it is c) a bit multiset with one unique element of multiplicity 2^4, or (in the case that it is a_i) a bit multiset with two unique elements ('0' and '1') each of multiplicty 2^3. In either case, the bit multiset exhibits even parity, i.e. its elements have even multiplicity. Composing the four bitslices back into the 4-bit word, the 4-bit input $\alpha_3\alpha_2\alpha_1\alpha_0$ to the Sbox is therefore also a multiset of property E. Since the Sbox is bijective, therefore this E property is preserved through to the output. Denote its output by e^4 with each e to represent a bit tuple of even parity.

- L in R4: $\rightarrow e^{15}a_3$, $e^{15}a_2$, $e^{15}a_1$, $e^{15}a_0$ The linear layer causes the 4-bit inputs to the Sbox in the next layer to be of the form e^4.

- S in R5: $\rightarrow (?^3e)^{16}$
  All four input bit tuples going into any Sbox have even parity (note that the same is also true for a_i).
  We focus on the LSB output y_0 of any Sbox, whose expression is given in equation (4). Notably, the only nonlinear term is the AND $(x_1 \wedge x_2)$ term between x_1 and x_2, which is in this case $(e \wedge e)$, leading to an output bit tuple that is also e, preserving the segment/pattern boundaries.
  As each of the other terms in y_0 i.e. x_0, x_2, x_3 to be XORed to x_1x_2 also has even parity, therefore the output bit y_0 of the Sbox would have even parity.

- L in R5: $\rightarrow ?^{48}(e)^{16}$
  The linear layer moves all the LSB output bits y_0 of all 16 Sboxes to the rightmost 16 bit positions, therefore at the end of Round 5, all the 16 rightmost bit tuples have even parity, and thus are balanced, i.e. $\bigoplus = 0$.

We have empirically tested this for several runs, each time with 2^{16} random structures of plaintext multisets and keys: it is verified that the rightmost 16 bits after 5 rounds of PRESENT have zero integral $\bigoplus$ sum with probability 1.

This result contrasts with the best-known integrals for PRESENT reported in [30], where a 5-round and a 7-round integral were demonstrated such that a zero integral sum is detected only in the single rightmost bit after five (resp. seven) rounds.

On the contrary, our tuple integral based on a multiset of size 2^4, has 16 times more checkable bits that provide more bit filtering conditions, thus better filtration power during the key recovery stage.

Furthermore, the single-bit integrals in [30] were tracked based on the maximal algebraic degree d of the coordinate Boolean function producing the rightmost LSB at the final output. This is done by collecting enough different texts within a multiset such that there are more than 2^d texts in order to exploit the higher-order derivative property [2], which basically tests for a zero sum.

In contrast, our integral is tracked by bitslice tuples through the round operations, notably through the Sbox's output bits, and within the Sboxes through the AND and XOR operations that make up the ANF expressions of an Sbox's coordinate Boolean functions. Such a bit tuple approach thus enables to observe richer internal structures within the multisets and bitslice tuples.

3.4 RECTANGLE

RECTANGLE [42] is a bit-sliced lightweight cipher, such that in each round it comprises an XOR based key addition, and then just an Sbox layer S followed by bitwise rotation layer R, thus RECTANGLE can be viewed as an NRX cipher, i.e. involving the primitive operations AND, rotation (ROT) and exclusive-OR (XOR). It has a 64-bit block size and 25 number of rounds.

Zhang et al. [38] reported a 7-round integral distinguisher needing a multiset of 2^{36} elements. Kosuge et al. [39] reported an 8-round integral distinguisher requiring a huge multiset of 2^{60} elements. Both these integrals track that the integral $\bigoplus$ sum equals zero in some output bits.

In contrast, we demonstrate how tuples can apply to RECTANGLE by using much smaller multisets, and track more structures in the output bits in contrast to zero sums, i.e. we track the bitslice tuples.

The Sbox S of RECTANGLE can be represented in terms of its ANF as:

$$y_3 = x_1 \oplus x_3 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2 \oplus x_1x_2x_3 \tag{5}$$
$$y_2 = 1 \oplus x_2 \oplus x_3 \oplus x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_2x_3 \oplus x_0x_1x_2 \tag{6}$$
$$y_1 = 1 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_1x_3 \tag{7}$$
$$y_0 = x_0 \oplus x_2 \oplus x_3 \oplus x_0x_1 \tag{8}$$

As an aside, note that the algebraic degree of the coordinate Boolean functions producing two of its output bits i.e. y_0 and y_1 is 2, versus degree of 3 for the other two output bits.

RECTANGLE's state can be represented as a rectangle of 4×16 bits as follows:

$x_{15}x_{14}x_{13}x_{12}$	$x_{11}x_{10}x_9x_8$	$x_7x_6x_5x_4$	$x_3x_2x_1x_0$
$x_{31}x_{30}x_{29}x_{28}$	$x_{27}x_{26}x_{25}x_{24}$	$x_{23}x_{22}x_{21}x_{20}$	$x_{19}x_{18}x_{17}x_{16}$
$x_{47}x_{46}x_{45}x_{44}$	$x_{43}x_{42}x_{41}x_{40}$	$x_{39}x_{38}x_{37}x_{36}$	$x_{35}x_{34}x_{33}x_{32}$
$x_{63}x_{62}x_{61}x_{60}$	$x_{59}x_{58}x_{57}x_{56}$	$x_{55}x_{54}x_{53}x_{52}$	$x_{51}x_{50}x_{49}x_{48}$

Consider if we had an input multiset of 2^4 elements of the following form:

a_0ccc	$cccc$	$cccc$	$cccc$
a_1ccc	$cccc$	$cccc$	$cccc$
a_2ccc	$cccc$	$cccc$	$cccc$
a_3ccc	$cccc$	$cccc$	$cccc$

We then observe its progagation as detailed below:

- S in R1: This propagates essentially unchanged through the Sbox layer in round 1, because we have four bit tuples $a_3a_2a_1a_0$ forming the leftmost column, entering one Sbox, and constant bit tuples entering the other Sboxes. Alternatively, viewed as 4-bit column-words, we have: PCCC CCCC CCCC CCCC

- R in R1: After bitwise rotation in round 1, we have:

$a_0\ c\ c\ c$	$c\ c\ c\ c$	$c\ c\ c\ c$	$c\ c\ c\ c$
$c\ c\ c\ c$	$c\ c\ c\ c$	$c\ c\ c\ c$	$c\ c\ c\ a_1$
$c\ c\ c\ c$	$a_2\ c\ c\ c$	$c\ c\ c\ c$	$c\ c\ c\ c$
$c\ c\ c\ a_3$	$c\ c\ c\ c$	$c\ c\ c\ c$	$c\ c\ c\ c$

- S in R2: At the leftmost column, we have $ccca_0$ entering the Sbox. Analyzing the Sbox's ANF, we see that the following bit tuples will be obtained at the output:

$$y_0 : \alpha_0 \in \{a_0, c\}; y_1 : a_0; y_2 : \alpha_0 \in \{a_0, c\}; y_3 : \alpha_0 \in \{a_0, c\}$$

Similarly, for the other input tuples with one a_j entering the Sbox, we have:

$$x_3x_2x_1x_0 = cca_1c \rightarrow \begin{cases} y_0 : \alpha_1 \in \{a_1, c\} \\ y_1 : \alpha_1 \in \{a_1, c\} \\ y_2 : \alpha_1 \in \{a_1, c\} \\ y_3 : \alpha_1 \in \{a_1, c\} \end{cases}$$

$$x_3x_2x_1x_0 = ca_2cc \rightarrow \begin{cases} y_0 : a_2 \\ y_1 : a_2 \\ y_2 : \alpha_2 \in \{a_2, c\} \\ y_3 : \alpha_2 \in \{a_2, c\} \end{cases}$$

$$x_3x_2x_1x_0 = a_3ccc \rightarrow \begin{cases} y_0 : a_3 \\ y_1 : \alpha_3 \in \{a_3, c\} \\ y_2 : \alpha_3 \in \{a_3, c\} \\ y_3 : \alpha_3 \in \{a_3, c\} \end{cases}$$

- R in R2: Therefore, after going through rotation, we have:

$$\begin{array}{cccc} \alpha_0\ c\ \ c\ \ a_3 & a_2\ c\ \ c\ \ c & c\ \ c\,c\,c & c\,c\ \ c\ \ \alpha_1 \\ c\ \ c\,\alpha_3\ a_2 & c\ \ c\ \ c\ \ c & c\ \ c\,c\,c & c\,c\,\alpha_1\ a_0 \\ c\ \ c\ \ c\ \ \alpha_1 & \alpha_0\ c\ \ c\ \ \alpha_3 & \alpha_2\ c\,c\,c & c\,c\ \ c\ \ c \\ c\ \ c\,\alpha_1\ \alpha_0 & c\ \ c\,\alpha_3\ \alpha_2 & c\ \ c\,c\,c & c\,c\ \ c\ \ c \end{array}$$

- S in R3: Passing through a subsequent Sbox layer gives:

$$\begin{array}{cccc} \alpha_0\ c\ \alpha_1 \oplus \alpha_3\ e & \alpha_0 \oplus \alpha_2\ c\ \alpha_3\ \alpha_2 \oplus \alpha_3 & \alpha_2\ c\,c\,c & c\,c\,\alpha_1\ \ a_0\alpha_1 \\ \alpha_0\ c\ \ \alpha_1\alpha_3\ \ e & \alpha_0 \oplus a_2\ c\ \alpha_3\ \alpha_2 \oplus \alpha_3 & \alpha_2\ c\,c\,c & c\,c\,\alpha_1\ \alpha_0 \oplus \alpha_1 \\ \alpha_0\ c\ \alpha_1 \oplus \alpha_3\ e & \alpha_0\alpha_2\ \ c\ \alpha_3\ \ \alpha_2\alpha_3 & \alpha_2\ c\,c\,c & c\,c\,\alpha_1\ \ \alpha_0\alpha_1 \\ \alpha_0\ c\ \ \alpha_1\alpha_3\ \ e & \alpha_0\alpha_2\ \ c\ \alpha_3\ \ \alpha_2\alpha_3 & \alpha_2\ c\,c\,c & c\,c\,\alpha_1\ \alpha_0 \oplus \alpha_1 \end{array}$$

- R in R3: After rotation, we have:

$$\begin{array}{cccccccccccccccc} \alpha_0 & c & \alpha_1 \oplus \alpha_3 & e & \alpha_0 \oplus \alpha_2 & c & \alpha_3 & \alpha_2 \oplus \alpha_3 & \alpha_2 & c & c & c & c & c & \alpha_1 & a_0\alpha_1 \\ c & \alpha_1\alpha_3 & e & \alpha_0 \oplus a_2 & c & \alpha_3 & \alpha_2 \oplus \alpha_3 & \alpha_2 & c & c & c & c & c & \alpha_1\ \alpha_0 \oplus \alpha_1 & \alpha_0 \\ c & c & \alpha_1 & \alpha_0\alpha_1 & \alpha_0 & c & \alpha_1 \oplus \alpha_3 & e & \alpha_0\alpha_2 & c & \alpha_3 & \alpha_2\alpha_3 & \alpha_2 & c & c & c \\ c & \alpha_1 & \alpha_0 \oplus \alpha_1 & \alpha_0 & c & \alpha_1\alpha_3 & e & \alpha_0\alpha_2 & c & \alpha_3\ \alpha_2\alpha_3 & \alpha_2 & c & c & c & c \end{array}$$

- S in R4: Going through another Sbox layer results in the following:

$$\begin{array}{cccc} \alpha_0\ \alpha_1\alpha_3\ *\ * & \alpha_0 \oplus \alpha_2\ \alpha_1\alpha_3\ *\ * & \alpha_0\alpha_2\ \alpha_3\ \alpha_2\alpha_3\ \alpha_2\alpha_3 & \alpha_2\ \alpha_1\ \ \alpha_0\alpha_1\ \ \alpha_0\alpha_1 \\ \alpha_0\ \alpha_1\alpha_3\ *\ * & \alpha_0 \oplus \alpha_2\ \alpha_1\alpha_3\ *\ * & \alpha_0\alpha_2\ \alpha_3\ \alpha_2\alpha_3\ \alpha_2\alpha_3 & \alpha_2\ \alpha_1\ \alpha_0 \oplus \alpha_1\ \alpha_0 \oplus a_0\alpha_1 \\ \alpha_0\ \alpha_1\alpha_3\ *\ * & \alpha_0\alpha_2\ \ \alpha_1\alpha_3\ *\ * & \alpha_0\alpha_2\ \alpha_3\ \alpha_2\alpha_3\ \alpha_2\alpha_3 & \alpha_2\ \alpha_1\ \ \alpha_0\alpha_1\ \ \alpha_0\alpha_1 \\ \alpha_0\ \alpha_1\alpha_3\ *\ * & \alpha_0\alpha_2\ \ \alpha_1\alpha_3\ *\ * & \alpha_0\alpha_2\ \alpha_3\ \alpha_2\alpha_3\ \alpha_2\alpha_3 & \alpha_2\ \alpha_1\ \alpha_0 \oplus \alpha_1\ \alpha_0 \oplus a_0\alpha_1 \end{array}$$

- R in R4: After rotation, we have:

$$\begin{array}{cccccccccccccccc} \alpha_0 & \alpha_1\alpha_3 & * & * & \alpha_0 \oplus \alpha_2 & \alpha_1\alpha_3 & * & * & \alpha_0\alpha_2 & \alpha_3 & \alpha_2\alpha_3 & \alpha_2\alpha_3 & \alpha_2 & \alpha_1 & \alpha_0\alpha_1 & \alpha_0\alpha_1 \\ \alpha_1\alpha_3 & * & * & \alpha_0 \oplus \alpha_2 & \alpha_1\alpha_3 & * & * & \alpha_0\alpha_2 & \alpha_3 & \alpha_2\alpha_3 & \alpha_2\alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \oplus \alpha_1 & \alpha_0 \oplus a_0\alpha_1 & \alpha_0 \\ \alpha_2 & \alpha_1 & \alpha_0\alpha_1 & \alpha_0\alpha_1 & \alpha_0 & \alpha_1\alpha_3 & * & * & \alpha_0\alpha_2 & \alpha_1\alpha_3 & * & * & \alpha_0\alpha_2 & \alpha_3 & \alpha_2\alpha_3 & \alpha_2\alpha_3 \\ \alpha_1 & \alpha_0 \oplus \alpha_1 & * & \alpha_0 & \alpha_1\alpha_3 & * & * & \alpha_0\alpha_2 & \alpha_1\alpha_3 & * & * & \alpha_0\alpha_2 & \alpha_3 & \alpha_2\alpha_3 & \alpha_2\alpha_3 & \alpha_2 \end{array}$$

- S in R5: In Round 5, passing through the Sbox layer obtains:

$$\begin{array}{cccc} e\ *\ *\ * & \alpha_0 \oplus \alpha_2\ *\ *\ * & \alpha_0\alpha_2\ *\ *\ * & e\ \ \alpha_1\ \ \alpha_0\alpha_1\ \ \alpha_0\alpha_1 \\ e\ *\ *\ * & \alpha_1\alpha_3\ \ *\ *\ * & \alpha_3\ \ *\ *\ * & e\ \alpha_0 \oplus \alpha_1\ \alpha_0 \oplus a_0\alpha_1\ \ \alpha_0 \\ *\ *\ *\ * & \alpha_0\ \ *\ *\ * & \alpha_0\alpha_2\ *\ *\ * & e\ \ \alpha_3\ \ \alpha_2\alpha_3\ \ \alpha_2\alpha_3 \\ e\ *\ *\ * & \alpha_1\alpha_3\ \ *\ *\ * & \alpha_1\alpha_3\ *\ *\ * & *\ \ \alpha_2\alpha_3\ \ \alpha_2\alpha_3\ \ \alpha_2 \end{array}$$

To see why e tuples are obtained at the output of the Sbox in the leftmost column, note that the input tuples are of the form $(\alpha_1, \alpha_2, \alpha_1\alpha_3, \alpha_0)$, and recall that $\alpha_i \in \{a_i, c\}$. Therefore, the possible input tuples to the Sbox are:

- $(c^s a^{4-s})$, $s \in \{0, \ldots, 3\}$ where the composition of c and a tuples are in any order. This forms an E word into the Sbox, therefore the output will also be an E.

- (a_1, a_2, a_3, a_0): This is a P word into the Sbox, therefore the output will also be a P.

- (a_1, a_2, a_1, a_0): This is actually an E word, thus the output will be an E.

- $(a_1, a_2, a_1 a_3, a_0) = (a_1, a_2, m_{13}, a_0)$: Note from our previous discussion that $a_1 a_3$ gives m_{13} i.e. a tuple as like a_1 but where every other contiguous sequence of 2^3 bits is 0. Given such an input, then the output tuple from the Sbox is of the form: $(e * ee)$.

Similar arguments apply to the fourth rightmost column whose input tuple is $(\alpha_2, \alpha_1, \alpha_0\alpha_1, \alpha_3)$. The crucial analysis is for the case where the input is $(a_3, a_0 a_2, a_1, a_2) = (a_3, m_{02}, a_1, a_2)$, which gives an output of $(*eee)$.

- R in R5: After another rotation, we obtain:

$$\begin{array}{cccc} e*** & **** & **** & e*** \\ **** & **** & ***e & ***e \\ e*** & **** & **** & **** \\ ***e & **** & **** & **** \end{array}$$

Thus, we have a 5-round integral distinguisher for RECTANGLE that requires only 2^4 texts, and the integral $\bigoplus$ zero sum can be detected in at least 6 bits. This has been corroborated by experiments, repeated for several runs. In fact, there is more structure existing in other bits that require much in-depth analysis, indeed empirical results show that the zero sum is detected in 22 bits after 5 rounds.

Analysing the rounds via tuples enables us to precisely track the evolution of multisets through more rounds than previously possible, because tuples allow to represent much richer structure than conventional integral/multiset analysis. This is why using only much smaller amounts of text (2^4 in our case for RECTANGLE) we can track at the bit level granularity for up to five rounds.

4 Concluding Remarks

The Slice-&-Fuse paradigm considered in this paper enables to move between word and bitslice granularities when tracking the evolution of multisets and their constituent tuples through cipher rounds. Towards that aim, our proposed new types of tuples capture more structures than previously known, that are inherent in multisets and yet which have largely remained unexplored until now. Open questions include whether much richer types of tuples exist as constituents of multisets, and developing advanced approaches to track the propagation of tuples through other cipher component operations including modulo multiplications that make up exotic ciphers such as XMX [4] as well as the more celebrated IDEA cipher [1].

References

1. X. Lai, J.L. Massey: A Proposal for a New Block Encryption Standard. In Eurocrypt '90. LNCS 473, 389–404 (1990)
2. L.R. Knudsen: Truncated and Higher Order Differentials. In FSE '94. LNCS 1008, 196–211 (1994)
3. J. Daemen, L. Knudsen, V. Rijmen: The Block Cipher Square. In FSE '97. LNCS 1267, 149–165 (1997)
4. D. M'Raïhi, D. Naccache, J. Stern, S. Vaudenay: Xmx: a Firmware-oriented Block Cipher based on Modular Multiplications. In FSE '97. LNCS 1267, 166–171 (1997)
5. R. Lidl, H. Niederreiter: Finite Fields. Encyclopedia of Mathematics & its Applications 20, Cambridge University Press (1997)
6. C. D'Halluin, G. Bijnens, V. Rijmen, B. Preneel: Attack on Six Rounds of Crypton. In FSE '99. LNCS 1636, 46–59 (1999)
7. C.H. Lim: A Revised Version of Crypton - Crypton V1.0. In FSE '99. LNCS 1636, 31–45 (1999)
8. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, D. Whiting: Improved Cryptanalysis of Rijndael. In FSE '00. LNCS 1978, 213–230 (2000)
9. S. Lucks: The Saturation Attack - A Bait for Twofish. In FSE '01. LNCS 2355, 1–15 (2001)
10. A. Biryukov, A. Shamir: Structural Cryptanalysis of SASAS. In Eurocrypt '01. LNCS 2045, 395–405 (2001)
11. L. Knudsen, D. Wagner: Integral Cryptanalysis. In FSE '02. LNCS 2365, 112–127 (2002)
12. F.-X. Standaert, G. Piret, G. Rouvroy, J.-J. Quisquater, J.-D. Legat: ICEBERG: An Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware. In FSE '04. LNCS 3017, 279–298 (2004)
13. J. Nakahara Jr., D.S. de Freitas, R.C.-W. Phan: New Multiset Attacks on Rijndael with Large Blocks. In Mycrypt '05. LNCS 3715, 277–295 (2005)
14. C. H. Lim, T. Korkishko: mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In WISA '05. LNCS 3786, 243–258 (2006)
15. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe: Present: an Ultra-Lightweight Block Cipher. In CHES '07. LNCS 4727, 450–466 (2007)
16. L.R. Knudsen, V. Rijmen: Known-Key Distinguishers for Some Block Ciphers. In Asiacrypt '07. LNCS 4833, 315–324 (2007)
17. M.R. Z'aba, H. Raddum, M. Henricksen, E. Dawson: Bit-Pattern based Integral Attack. In FSE '08. LNCS 5086, 363–381 (2008)
18. B. Sun, L. Qu, C. Li: New Cryptanalysis of Block Ciphers with Low Algebraic Degree. In FSE '09. LNCS 5665, 180–192 (2009)
19. Y. Wei, B. Sun, C. Li: New Integral Distinguisher for Rijndael-256. In IACR ePrint Archive. Report 559. Available online at: http://eprint.iacr.org/2009/559 (2009)
20. A. Biryukov, A. Shamir: Structural Cryptanalysis of SASAS. Journal of Cryptology. 23(4), 505–518 (2010)
21. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, J. Walker: The Skein Hash Function Family, version 1.3. Submitted to NIST SHA-3 Competition Round 3 (2010)
22. J.-P. Aumasson, G. Leurent, W. Meier, F. Mendel, N. Mouha, R.C.-W. Phan, Y. Sasaki, P. Susil: Tuple Cryptanalysis of ARX with Application to BLAKE and Skein. In ECRYPT II Hash Workshop (Hash '11) (2011)
23. W. Zhang, B. Su, W. Wu, D. Feng, C. Wu: Extending Higher-Order Integral: An Efficient Unified Algorithm of Constructing Integral Distinguishers for Block Ciphers. In ACNS '12. LNCS 7341, 117–134 (2012)
24. Y. Sasaki, L. Wang: Meet-in-the-Middle Technique for Integral Attacks against Feistel Ciphers. In SAC '12. LNCS 7707, 234–251 (2012)

25. J. Borghoff, A. Canteaut, T. Güneysu, E.B. Kavun, M. Knezevic, L.R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S.S. Thomsen, T. Yalcin: PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In Asiacrypt '12. LNCS 7658, 208–225 (2012)
26. J. Lu, Y. Wei, J. Kim, E. Pasalic: The Higher-Order Meet-in-the-Middle Attack and its Application to the Camellia Block Cipher. In Indocrypt '12. LNCS 7668, 244–264 (2012)
27. C. Peng, C. Zhu, Y. Zhu, F. Kang: Practical Symbolic Computation in Block Cipher with application to PRESENT. In IACR ePrint Archive. Report 587. Available online at: http://eprint.iacr.org/2012/587 (2012)
28. J. Jean, I. Nikolić, T. Peyrin, L. Wang, S. Wu: Security Analysis of PRINCE. In FSE '13. LNCS 8424, 92–111 (2013)
29. Y. Sasaki, L. Wang: Bitwise Partial-Sum on HIGHT: a New Tool for Integral Analysis against ARX Designs. In ICISC '13. LNCS 8565, 189–202 (2013)
30. S. Wu, M. Wang: Integral Attacks on Reduced-round PRESENT. In ICICS '13. LNCS 8233, 331–345 (2013)
31. Y. Todo, K. Aoki: FFT Key Recovery for Integral Attack. In CANS '14. LNCS 8813, 64–81 (2014)
32. Q. Wang, Z. Liu, K. Varıcı, Y. Sasaki, V. Rijmen, Y. Todo: Cryptanalysis of Reduced-round SIMON32 and SIMON48. In Indocrypt '14. LNCS 8885, 143–160 (2014)
33. J. Lu, Y. Wei, J. Kim, E. Pasalic: The Higher-Order Meet-in-the-Middle Attck and its Application to the Camellia Block Cipher. Information Processing Letters, 527, 102–122 (2014)
34. Y. Todo: Structural Evaluation by Generalised Integral Property. In EUROCRYPT '15. LNCS 9056, 287–314 (2015)
35. Y. Todo: Integral Cryptanalysis on Full MISTY1. In CRYPTO '15. LNCS 9215, 413–432 (2015)
36. C. Blondeau, T. Peyrin, L. Wang: Known-Key Distinguisher on Full PRESENT. In CRYPTO '15. LNCS 9215, 455–474 (2015)
37. Akshima, D. Chang, M. Ghosh, A. Goel, S.K. Sanadhya: Improved Meet-in-the-Middle Attacks on 7 and 8-round ARIA-192 and ARIA-256. In Indocrypt '15. LNCS 9462, 198–217 (2015)
38. H. Zhang, W. Wu, Y. Wang: Integral Attack against Bit-oriented Block Ciphers. In ICISC '15. LNCS 9558, 102–118 (2015)
39. H. Kosuge, H. Tanaka, K. Iwai, T. Kurokawa: Integral Attack on Reduced-Round RECTANGLE. In IEEE CSCloud '15. 68–73 (2015)
40. Y. Wei: Bit-pattern Based Integral Attack on ICEBERG. In INCOS '15. 370–373 (2015)
41. Y. Sasaki, L. Wang: Bitwise Partial-Sum on HIGHT: a New Tool for Integral Analysis against ARX Designs. IEICE Transactions on Fundamentals, E98A(1), 49–60 (2015)
42. W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, I. Verbauwhede: RECTANGLE: a Bit-slice Lightweight Block Cipher Suitable for Multiple Platforms. Science China Information Sciences, 58(12), 1–15 (2015)
43. I. Dinur, O. Dunkelman, T. Kranz, G. Leander: Decomposing the ASASA Block Cipher Construction. In IACR ePrint Archive. Report 507. Available online at: http://eprint.iacr.org/2015/507 (2015)
44. P. Morawiecki: Practical Attacks on the Round-reduced PRINCE. In IACR ePrint Archive. Report 245. Available online at: http://eprint.iacr.org/2015/245 (2015)
45. R. Posteuca and G. Negara: Integral Cryptanalysis of Round-Reduced PRINCE cipher. Proceedings of the Romanian Academy Series A, 16, 265–269 (2015)
46. C. Boura and A. Canteaut: Another View of the Division Property. In CRYPTO '16. LNCS 9814, 654–682 (2016)

Acknowledgements

We thank the anonymous Mycrypt 2016 reviewers and Ana Sălămean for constructive and critical comments that have improved this paper. RP is supported in part by the Malaysian Ministry of Education's Fundamental Research Grant Scheme under the project *ProvAdverse*.

A Integrals of Crypton and mCrypton

Crypton v1.0 and mCrypton are two block ciphers proposed by C. H. Lim et al. [7, 14]. Crypton was one of the candidates of the AES competition, acting on 128-bit blocks under keys of length 128, 192 or 256 bits whereas mCrypton is its equivalent lightweight version acting on 64-bit blocks under keys of length 64, 96 or 128 bits. Both have the same design principle based on an SPN structure with an Sbox layer and a linear layer composed of a bit permutation and a matrix transposition. The bit permutation could be represented by a matrix multiplication where the MDS bound is not reached (few column elements have an input/output weight of 4 instead of 5 in the case of an MDS transformation). For the rest of this subsection, we will denote the block size as $16n$ with $n = 8$ for Crypton and $n = 4$ for mCrypton.

We are not able to directly exhibit bitslice properties in the cases of Crypton and mCrypton. We conjecture that this fact is linked with the inherent design of the ciphers: the Sbox layer and the bit permutation act on different word sizes leading to impeding the possible properties at bit level. However, we are able to improve the classical integral property used in [6] on Crypton and that also works on mCrypton: we start with one active word at the beginning (whereas the other words are constant) and that gives a zero integral $\bigoplus$ sum in each word after three rounds. As the linear layer does not have the MDS property, we are able to construct a new four-round integral property that works for both Crypton and mCrypton. More precisely, the following 3rd-order integral property holds on 4 rounds requiring only 2^{3n} chosen plaintexts. Indeed:

$$\begin{array}{cccc} \mathsf{A}^{3n} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{A}^{3n} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{A}^{3n} & \mathsf{C} & \mathsf{C} & \mathsf{C} \\ \mathsf{C} & \mathsf{C} & \mathsf{C} & \mathsf{C} \end{array}$$

gives after 4 rounds

$$\begin{array}{cccc} \mathsf{B} & \mathsf{B} & \mathsf{B} & \mathsf{B} \\ \mathsf{B} & \mathsf{B} & \mathsf{B} & \mathsf{B} \\ \mathsf{B} & \mathsf{B} & \mathsf{B} & \mathsf{B} \\ \mathsf{B} & \mathsf{B} & \mathsf{B} & \mathsf{B} \end{array}$$

Thus, we could improve in the case of Crypton the data/time complexity of the 6-round attack presented in [6] by a factor 2^{16} for the time complexity and by a factor 2^8 for the data complexity. Indeed, we add two rounds at the end of the previous 4-round integral property leading to guess 5 key words of the subkeys K^5 and K^6 for a cost of

2^{40} tests whereas to discard false alarms we need to test $5 \cdot 2^{24}$ chosen plaintexts. This attack could be easily adapted to mCrypton leading to an attack on 6 rounds using $5 \cdot 2^{12}$ chosen plaintexts with a time complexity of about 2^{20} tests.