Design Strategies for ARX with Provable Bounds: Sparx and LAX (Full Version)*

Daniel Dinu[1], Léo Perrin[1], Aleksei Udovenko[1], Vesselin Velichkov[1], Johann Großschädl[1], Alex Biryukov[1]

first-name.last-name@uni.lu
SnT, University of Luxembourg

Abstract. We present, for the first time, a general strategy for designing ARX symmetric-key primitives with provable resistance against single-trail differential and linear cryptanalysis. The latter has been a long standing open problem in the area of ARX design. The *wide trail design strategy* (WTS), that is at the basis of many S-box based ciphers, including the AES, is not suitable for ARX designs due to the lack of S-boxes in the latter. In this paper we address the mentioned limitation by proposing the *long trail design strategy* (LTS) – a dual of the WTS that is applicable (but not limited) to ARX constructions. In contrast to the WTS, that prescribes the use of small and efficient S-boxes at the expense of heavy linear layers with strong mixing properties, the LTS advocates the use of large (ARX-based) S-Boxes together with sparse linear layers. With the help of the so-called *Long Trail argument*, a designer can bound the maximum differential and linear probabilities for any number of rounds of a cipher built according to the LTS.

To illustrate the effectiveness of the new strategy, we propose Sparx – a family of ARX-based block ciphers designed according to the LTS. Sparx has 32-bit ARX-based S-boxes and has provable bounds against differential and linear cryptanalysis. In addition, Sparx is very efficient on a number of embedded platforms. Its optimized software implementation ranks in the top 6 of the most software-efficient ciphers along with Simon, Speck, Chaskey, LEA and RECTANGLE.

As a second contribution we propose another strategy for designing ARX ciphers with provable properties, that is completely independent of the LTS. It is motivated by a challenge proposed earlier by Wallén and uses the differential properties of modular addition to minimize the maximum differential probability across multiple rounds of a cipher. A new primitive, called LAX, is designed following those principles. LAX partly solves the Wallén challenge.

Keywords: ARX, block ciphers, differential cryptanalysis, linear cryptanalysis, lightweight, wide trail strategy

* Full version of the paper published in the proceedings of ASIACRYPT'16 (©IACR).

1 Introduction

ARX, standing for Addition/Rotation/XOR, is a class of symmetric-key algorithms designed using only the following simple operations: modular addition, bitwise rotation and exclusive-OR. In contrast to S-box-based designs, where the only non-linear elements are the substitution tables (S-boxes), ARX designs rely on modular addition as the only source of non-linearity. Notable representatives of the ARX class include the stream ciphers Salsa20 [1] and ChaCha20 [2], the SHA-3 finalists Skein [3] and BLAKE [4] as well as several lightweight block ciphers such as TEA, XTEA [5], etc. Dinu et al. recently reported [6] that the most efficient software implementations on small processors belonged to ciphers from the ARX class: Chaskey-cipher [7] by Mouha et al., SPECK [8] by the American National Security Agency (NSA) and LEA [9] by the South Korean Electronic and Telecommunications Research Institute.[1]

For the mentioned algorithms, the choice of using the ARX paradigm was based on three observations[2]. First, getting rid of the table look-ups, associated with S-Box based designs, increases the resilience against side-channel attacks. Second, this design strategy minimizes the total number of operations performed during an encryption, allowing particularly fast software implementations. Finally, the computer code describing such algorithms is very small, making this approach especially appealing for lightweight block ciphers where the memory requirements are the harshest.

Despite the widespread use of ARX ciphers, the following problem has remained open up until now.

Open Problem. *Is it possible to design an ARX cipher that is provably secure against single-trail differential and linear cryptanalysis* by design*?*

To the best of our knowledge, there has only been one attempt at tackling this issue. In [10] Biryukov et al. have proposed several ARX constructions for which it is feasible to compute the exact maximum differential and linear probabilities over any number of rounds. However, these constructions are limited to 32-bit blocks. The general case of this problem, addressing any block size, has still remained without a solution.

More generally, the formal understanding of the cryptographic properties of ARX is far less satisfying than that of, for example, S-Box-based substitution-permutation networks (SPN). Indeed, the wide trail strategy [11] (WTS) and the wide trail argument [12] provide a way to design S-box based SPNs with provable resilience against differential and linear attacks. It relies on bounding the number of active S-Boxes in a differential (resp. linear) trail and deducing a lower bound on the best expected differential (resp. linear) probability.

[1] SPECK and the MAC Chaskey are being considered for standardization by ISO.

[2] For SPECK, we can only guess it is the case as the designers have not published the rationale behind their algorithm.

Our Contribution. We propose two different strategies to build ARX-based block ciphers with provable bounds on the maximum expected differential and linear probabilities, thus providing a solution to the open problem stated above.

The first strategy is called the *Long Trail Strategy* (LTS). It borrows the idea of counting the number of active S-Boxes from the wide trail argument but the overall principle is actually the opposite to the wide trail strategy as described in [11]. While the WTS dictates the spending of most of the computational resources in the linear layer in order to provide good diffusion between small S-boxes, the LTS advocates the use of large and comparatively expensive S-Boxes in conjunction with cheaper and weaker linear layers. We formalize this method and describe the *Long Trail argument* that can be used to bound the differential and linear trail probabilities of a block cipher built using this strategy.

Using this framework, we build a family of lightweight block ciphers called Sparx. All three instances in this family can be entirely specified using only three operations: addition modulo 2^{16}, 16-bit rotations and 16-bit XOR. These ciphers are, to the best of our knowledge, the first ARX-based block ciphers for which the probability of both differential and linear trails are bounded. Furthermore, while one may think that these provable properties imply a performance degradation, we show that it is not the case. On the contrary, Sparx ciphers have very competitive performance on lightweight processors. In fact, the most lightweight version – Sparx-64/128 – is in the top 3 for 16-bit micro-controllers according to the classification method presented in [6].

Finally, we propose the LAX construction, where bit rotations are replaced with a more general linear permutation. The bounds on the differential probability are expressed as a function of the branching number of the linear layer. We note that the key insight behind this construction has been published in [13], but its realization has been left as a challenge.

Outline. First, we introduce the notations and concepts used throughout the paper in Section 2. In Section 3, we describe how an ARX-based cipher with provable bounds can be built using an S-Box-based approach and how the method used is a particular case of the more general *Long Trail Strategy.* Section 4 contains the specification of the Sparx family of ciphers, the description of its design rationale and a discussion about the efficiency of its implementation on microcontrollers. The LAX structure is presented in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

We use $\mathbb{F}_2$ to denote the set $\{0, 1\}$. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $(a, b) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ and $x \in \mathbb{F}_2^n$. We denote the probability of the differential trail $(a \overset{d}{\to} b)$ by $\Pr[f(x) \oplus f(x \oplus a) = b]$ and the correlation of the linear approximation $(a \overset{\ell}{\to} b)$ by $\big(2 \Pr[a \cdot x = b \cdot f(x)] - 1\big)$ where $y \cdot z$ is the scalar product of y and z.

In an iterated block cipher, not all differential (respectively linear) trails are possible. Indeed, they must be coherent with the overall structure of the round function. For example, it is well known that a 2-round differential trail for the AES with less than 4 active S-Boxes is impossible. To capture this notion, we use the following definition.

Definition 1 (Valid Trail). *Let f be an n-bit permutation. A trail $a_0 \rightarrow \ldots \rightarrow a_r$ for r rounds of f is a* valid trail *if $Pr[a_i \rightarrow a_{i+1}] > 0$ for all i in $[0, r-1]$. The set of all valid r-round differential (respectively linear) trails for f is denoted $\mathcal{V}_\delta(f)^r$ (resp. $\mathcal{V}_\ell(f)^r$).*

We use the acronyms MEDCP and MELCC to denote resp. *maximum expected differential characteristic probability* and *maximum expected linear characteristic correlation* – a signature introduced earlier in [14]. The MEDCP of the keyed function $f_{k_i} : x \mapsto f(x \oplus k_i)$ iterated over r rounds is defined as follows:

$$\mathrm{MEDCP}(f^r) = \max_{(\Delta_0 \rightarrow \ldots \Delta_r) \in \mathcal{V}_\delta(f)^r} \prod_{i=0}^{r-1} \Pr[\Delta_i \overset{d}{\rightarrow} \Delta_{i+1}],$$

where $\Pr[\Delta_i \overset{d}{\rightarrow} \Delta_{i+1}]$ is the expected value of the differential probability of $\Delta_i \overset{d}{\rightarrow} \Delta_{i+1}$ for the function f_k when k is picked uniformly at random. $\mathrm{MELCC}(f^r)$ is defined analogously. Note that $\mathrm{MEDCP}(f^r)$ and $\left(\mathrm{MEDCP}(f^1)\right)^r$ are *not* equal.

As designers, we thrive to provide upper bounds for both $\mathrm{MEDCP}(f^r)$ and $\mathrm{MELCC}(f^r)$. Doing so allows us to compute the number of rounds f needed in a block cipher for the probability of all trails to be too low to be usable. In practice, we want $\mathrm{MEDCP}(f^r) \ll 2^{-n}$ and $\mathrm{MELCC}(f^r) \ll 2^{-n/2}$ where n is the block size.

While this strategy is the best known, the following limitations must be taken into account by algorithm designers.

1. The quantities $\mathrm{MEDCP}(f^r)$ and $\mathrm{MELCC}(f^r)$ are relevant only if we make the *Markov assumption*, meaning that the differential and linear probabilities are independent in each round. This would be true if the subkeys were picked uniformly and independently at random but, as the master key has a limited size, it is not the case.
2. These quantities are averages taken over all possible keys: it is not impossible that there exists a weak key and a differential trail T such that the probability of T is higher than $\mathrm{MEDCP}(f^r)$ for this particular key. The same holds for the linear probability.
3. These quantities deal with unique trails. However, it is possible that several differential trails share the same input and output differences, thus leading to a higher probability for said differential transition. This so-called *differential effect* can be leveraged to decrease the data complexity of differential attack. The same holds for linear attacks where several approximations may form a linear hull.

Still, this type of bound is the best that can be achieved in a generic fashion (to the best of our knowledge). In particular, this is the type of bound provided by the wide trail argument used in the AES.

3 ARX-Based Substitution-Permutation Network

In this section, we present a general design strategy for building ARX-based block ciphers borrowing techniques from SPN design. The general idea is to build a SPN with ARX-based S-boxes instead of with S-boxes based on look-up tables (LUT). The proofs for the bound on the MEDCP and MELCC are inspired by the wide trail argument introduced in the design of the AES [12]. However, because of the use of large S-Boxes, the method used relies on a different type of interaction between the linear and non-linear layers. We call the corresponding design strategy the *Long Trail Strategy*. It is quite general and could be also applied in other contexts e.g. for non-ARX constructions.

First, we present possible candidates for the ARX-based S-Box and, along the way, identify the likely reason behind the choice of the rotation constants in SPECK-32. Then, we describe the Long Trail Strategy in more details. Finally, we present two different algorithms for computing a bound for the MEDCP and MELCC of block ciphers built using a LT strategy. We also discuss how to ensure that the linear layer provides sufficient diffusion.

3.1 ARX-Boxes

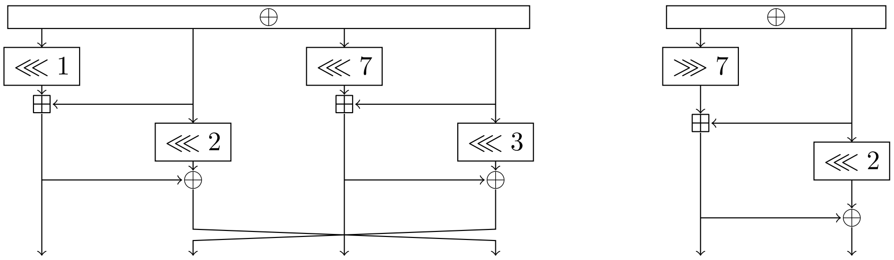
Definition 2 (ARX-box). *An* ARX*-box is a permutation on m bits (where m is much smaller than the block size) which relies entirely on addition, rotation and XOR to provide both non-linearity and diffusion. An* ARX*-box is a particular type of S-Box.*

Possible constructions for ARX-boxes can be found in a recent paper by Biryukov et al. [10]. A first one is based on the MIX function of Skein [3] and is called Marx-2. The rotation amounts, namely $\{1, 2, 7, 3\}$, were chosen so as to minimize the differential and linear probabilities. The key addition is done over the full state. The second construction is called Speckey and consists of one round of Speck-32 [8] with the key added to the full state instead of only to half the state as in the original algorithm. The two constructions Marx-2 and Speckey are shown in Fig. 1a and 1b. The differential and linear bounds for them are given in Table 1. While it is possible to choose the rotations used in Speckey in such a way as to slightly decrease the differential and linear bounds[3], such rotations are more expensive on small microcontrollers which only have instructions implementing rotations by 1 and by 8 (in both directions). We infer, although we cannot prove it, that the designers of Speck-32 made similar observations.

3.2 Naive Approaches and Their Limitations

A very simple method to build ARX-based ciphers with provable bounds on MEDCP and MELCC is to use a SPN structure where the S-boxes are replaced

[3] Both can be lowered by a factor of 2 if we choose rotations $(9, 2), (9, 5), (11, 7)$ or $(7, 11)$ instead of $(7, 2)$.



(a) Marx-2. (b) Speckey.

Fig. 1: Key addition followed by the candidate 32-bit ARX-boxes, Marx-2 and Speckey. The branch size is 8 bits for Marx-2, 16 bits for Speckey.

Table 1: Maximum expected differential characteristic probabilities (MEDCP) and maximum expected absolute linear characteristic correlations (MELCC) of Marx-2 and Speckey ($\log_2$ scale); r is the number of rounds.

	r	1	2	3	4	5	6	7	8	9	10
Marx-2	MEDCP(M^r)	−0	−1	−3	−5	−11	−16	−22	−25	−29	−35
	MELCC(M^r)	−0	−0	−1	−3	−5	−8	−10	−13	−15	−17
Speckey	MEDCP(S^r)	−0	−1	−3	−5	−9	−13	−18	−24	−30	−34
	MELCC(S^r)	−0	−0	−1	−3	−5	−7	−9	−12	−14	−17

by ARX operations for which we can compute the MEDCP and MELCC. This is indeed the strategy we follow but care must be taken when actually choosing the ARX-based operations and the linear layer.

Let us for example build a 128-bit block cipher with an S-Box layer consisting in one iteration of Speckey on each 32-bit word and with an MDS linear layer, say a multiplication with the `MixColumns` matrix with elements in $GF(2^{32})$ instead of $GF(2^8)$. The MEDCP bound of such a cipher, computed using a classical wide trail argument, would be equal to 1! Indeed, there exists probability 1 differentials for 1-round Speckey so that, regardless of the number of active S-Boxes, the bound would remain equal to 1. Such an approach is therefore not viable.

As the problem identified above stems from the use of 1-round Speckey, we now replace it with 3-round Speckey where the iterations are interleaved with the addition of independent round keys. The best linear and differential probabilities are no longer equal to 1, meaning that it is possible to build a secure cipher using the same layer as before provided that enough rounds are used. However, such a cipher would be very inefficient. Indeed, the MDS bound imposes that 5 arx-boxes are active every 2 rounds, so that the MEDP bound is equal to $p_d^{5r/2}$ where r is the number of rounds and p_d is the best differential probability of the ARX-box (3-rounds Speckey). To push the bound below 2^{-128} we need at least 18 SPN rounds, meaning 54 parallel applications of the basic ARX-round! We will show that, with our alternative approach, we can obtain the same bounds with much fewer rounds.

3.3 The Long Trail Design Strategy

Informed by the shortcomings of the naive design strategies described in the previous section, we devised a new method to build ARX-based primitives with provable linear and differential bounds. It is based on the following observation.

Observation 1 (Impact of Long Trails) *Let $d(r)$ and $\ell(r)$ be the* MEDCP *and* MELCC *of some* ARX*-box iterated r times and interleaved with the addition of independent subkeys. Then, in most cases:*

$$d(qr) \ll d(r)^q \text{ and } \ell(qr) \ll \ell(r)^q .$$

In other words, in order to diminish the MEDCP *and* MELCC *of a construction, it is better to allow long trails of* ARX*-boxes without mixing.*

For example, if we look at Speckey, the MEDCP for 3 rounds is 2^{-3} and that of 6 rounds is 2^{-15} which is far smaller than $(2^{-3})^2 = 2^{-6}$ (see Table 1). Similarly, the MELCC for 3 rounds is 2^{-1} and after 6 rounds it is $2^{-7} \ll (2^{-1})^2$.

In fact, a similar observation has been made by Nikolić when designing the CAESAR candidate family Tiaoxin [15]. It was later generalized to larger block sizes in [16], where Jean and Nikolić present, among others, the AES-based $\mathcal{A}_{\oplus}^2$ permutation family. It uses a partial S-Box layer where the S-Box consists of 2 AES rounds and a word-oriented linear layer in such a way that some of the S-Box calls can be chained within 2-round long trails. Thus, they may use the 4-round bound on the number of active 8-bit AES S-Boxes, which is 25, rather than twice the 2-round bound, which would be equal to 10 (see Table 2). Their work on this permutation can be interpreted as a particular case of the observation above.

Table 2: Bound on the number of active 8-bit S-Boxes in a differential (or linear) trail for the AES.

# R	1	2	3	4	5	6	7	8	9	10
# Active S-Boxes	1	5	9	25	26	30	34	50	51	55

Definition 3 (Long Trail). *We call* Long Trail (LT) *an uninterrupted sequence of calls to an* ARX*-box interleaved with key additions. No difference can be added into the trail from the outside. Such trails can happen for two reasons.*

1. *A* Static Long Trail *occurs with probability 1 because one output word of the linear layer is an unchanged copy of one of its input words.*
2. *A* Dynamic Long Trail *occurs within a specific differential trail because one output word of the linear layer consists of the XOR of one of its input words with a non-zero difference and a function of words with a zero difference. In this way the output word of the linear layer is again equal to the input word as in a Static LT, but here this effect has been obtained dynamically.*

Definition 4 (Long Trail Strategy). *The Long Trail Strategy is a design guideline: when designing a primitive with a rather weak but large S-Box (say, an ARX-based permutation), it is better to foster the existence of long trails rather than to have maximum diffusion in each linear layer.*

This design principle has an obvious caveat: although slow, diffusion is necessary! Unlike the WTS, in this context it is better to trade some of the power of the diffusion layer in favor of facilitating the emergence of long trails.

The Long Trail Strategy is a method for building secure and efficient ciphers using a large but weak S-Box S such that we can bound the MEDCP (and MELCC) of several iterations of $x \mapsto S(x \oplus k)$ with independent round keys. In this paper, we focus on the case where S consists of ARX operations but this strategy could have broader applications such as, as briefly discussed above, the design of block ciphers operating on large blocks using the AES round function as a building block.

In a way, this design method is the direct opposite of the wide trail strategy as it is summarized by Daemen and Rijmen in [11] (emphasis ours):

> Instead of spending most of the resources on large S-boxes, the wide trail strategy aims at designing the round transformation(s) such that there are no trails with a low bundle weight. In ciphers designed by the wide trail strategy, *a relatively large amount of resources is spent in the linear step* to provide high multiple-round diffusion.

The long trail approach *minimizes* the amount of resources spent in the linear layer and does spend most of the resources on large S-Boxes. Still, as discussed in the next section, the method used to bound the MEDCP and MELCC in the Long Trail Strategy is heavily inspired by the one used in the wide trail strategy.

A Cipher Structure for the LT Strategy We can build block ciphers based on the Long Trail Strategy using the following two-level structure. First, we must choose an S-Box layer operating on w words in parallel. The composition of a key addition in the full state and the application of this S-Box layer is called a *round*. Several rounds are iterated and then a word-oriented linear mixing layer is applied to ensure diffusion between the words. The composition of r rounds followed by the linear mixing layer is called a *step*[4], as described in Fig. 2. The encryption thus consists in iterating such steps. We used this design strategy to build a block cipher family, SPARX, which we describe in Section 4.

[4] This terminology is borrowed from the specification of LED [17] which also groups several calls of the round function into a step.

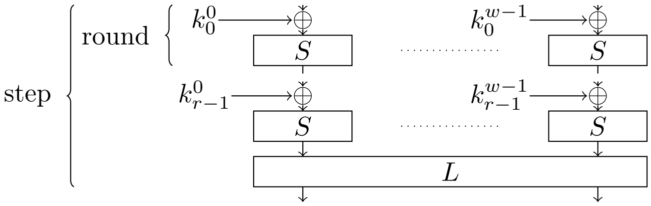


Fig. 2: A cipher structure for the LT strategy.

Long Trail-Based Bounds In what follows we only discuss differential long trails for the sake of brevity. Linear long trails are treated identically.

Definition 5 (Truncated LT Decomposition). *Consider a cipher with a round function operating on w words. A truncated differential trail is a sequence of values of $\{0,1\}^w$ describing whether an S-Box is active at a given round. The* LT Decomposition *of a truncated differential trail is obtained by grouping together the words of the differential trails into long trails and then counting how many active long trails of each length are present. It is denoted $\{t_i\}_{i\geq 1}$ where t_i is equal to the number of truncated long trails with length i.*

Example 1. Consider a 64-bit block cipher using a 32-bit S-Box, one round of Feistel network as its linear layer and 4 steps without a final linear layer. Consider the differential trail $(\delta_0^L, \delta_0^R) \rightarrow (\delta_1^L, \delta_1^R) \rightarrow (0, \delta_2^R) \rightarrow (\delta_3^L, 0)$ (see Fig. 3 where the zero difference is dashed). Then this differential trail can be decomposed into 3 long trails represented in black, blue and red: the first one has length 1 and δ_0^R as its input; the second one has length 2 and δ_0^L as its input; and the third one has length 3 and δ_1^L as its input so that the LT decomposition of this trail is $\{t_1 = 1, t_2 = 1, t_3 = 1\}$. Using the terminology introduced earlier, the first two trails are Static LT, while the third one is a Dynamic LT.

Theorem 1 (Long Trail Argument). *Consider a truncated differential trail T covering r rounds consisting of an S-Box layer with S-Box S interleaved with key additions and some linear layer. Let $\{t_i\}_{i\geq 1}$ be the LT decomposition of T. Then the probability p_D of any fully specified differential trail fitting in T is upper-bounded by*

$$p_D \leq \prod_{i\geq 1} \left(\mathrm{MEDCP}(S^i)\right)^{t_i}$$

where $\mathrm{MEDCP}(S^i)$ is an upper-bound on the probability of a differential trail covering i iterations of S.

Proof. Let $\Delta_{i,s} \overset{d}{\rightarrow} \Delta_{j,s+1}$ denote any differential trail occurring at the S-Box level in one step, so that the S-Box with index i at step s sees the transition $\Delta_{i,s} \overset{d}{\rightarrow} \Delta_{j,s+1}$. By definition of a long trail, we have in each long trail a chain of differential trails $\Delta_{i_0,s_0} \overset{d}{\rightarrow} \Delta_{i_1,s_0+1} \overset{d}{\rightarrow} ... \overset{d}{\rightarrow} \Delta_{i_t,s_0+t}$ which, because of the lack

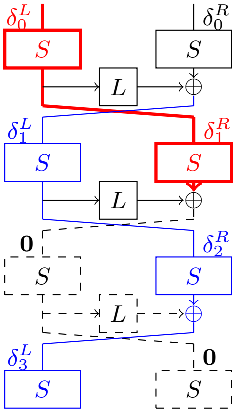


Fig. 3: An example of active LT decomposition.

of injection of differences from the outside, is a *valid trail* for t iterations of the S-Box. This means that the probability of any differential trail following the same sequence of S-boxes as in this long trail is upper-bounded by $\text{MEDCP}(S^t)$. We simply bound the product by the product of the bounds to derive the theorem. □

3.4 Choosing the Linear Layer: Bounding the MEDCP and MELCC while Providing Diffusion

In order to remain as general as possible, in this section we do not consider the details of a specific S-Box but instead we focus on fleshing out design criteria for the linear layer. All the information for the S-Box that is necessary to follow the explanation is the MEDCP and MELCC of its r-fold iterations including the key additions e.g. the data provided in Table 1 for our ARX-box candidates.

As the linear layers we consider may be weaker than usual designing SPN, it is also crucial that we ensure that ciphers built using such a linear layer are not vulnerable to integral attacks [18], in particular those based on the division property [19]. Incidentally, this gives us a criteria quantifying the diffusion provided by several steps of the cipher.

In this section, we propose two methods for bounding the MEDCP and MELCC of several steps of a block cipher. The first one is applicable to any linear layer but is relatively inefficient, while the second one works only for a specific subset of linear layers but is very efficient.

When considering truncated differential trails, it is hard to bound the probability of the event that differences in two or more words cancel each other in the linear layer. Therefore, for simplicity we assume that such cancellations happen *for free* i.e. with probability 1. Due to this simplification, we expect our bounds to be higher (i.e. looser) than the tight bounds. In other words, we *underestimate* the security of the cipher. Note that we also exclude the cases where the full state at some round has zero difference as the latter is impossible due to the cipher being a permutation.

Algorithms for Bounding MEDCP and MELCC of a cipher. In this sub-section we propose generic approaches that do not depend on the number of rounds per step. In fact, to fully avoid the confusion between *rounds* and *steps* in what follows we shall simply refer to SPN *rounds*.

One way to bound the MEDCP and MELCC of a cipher is as follows:

1. Enumerate all possible truncated trails composed of active/inactive S-boxes.
2. Find an optimal decomposition of each trail into long trails (LT).
3. Bound the probability of each trail using the product of the MEDCP (resp. MELCC) of all active long trails i.e. by applying the Long Trail Argument (see Theorem 1) on the corresponding optimal trail decomposition.
4. The maximum bound over all trails is the final upper bound.

This approach is feasible only for a small number of rounds, because the number of trails grows exponentially. This case is still of large interest so we sketch an effective algorithm for the only nontrivial step, the second one, in Appendix A.3. The algorithm is based on a recursive dynamic programming approach and has time complexity $O(wr^2)$, where w is the number of S-Boxes applied in parallel in each S-Box layer and r is the number of rounds.

As noted, the most complicated step in the above procedure is finding an optimal decomposition of a given truncated trail into long trails. The difficulty arises from the so-called *branching*: situation in which a long trail may be extended in more than one way. Recall that our definition of LT (cf. Definition 3) relies on the fact that there is no linear transformation on a path between two S-Boxes in a LT. The only transformations allowed are some XORs. Therefore, branching happens only when some output word of the linear layer receives two or more active input words without modifications. In order to cut off the branching effect (and thus to make finding the optimal decomposition of a LT feasible), we can put some additional linear functions that will modify the contribution of (some of) the input words. Equivalently, when choosing a linear layer we simply do not consider layers which cause branching of LTs. As we will show later, this restriction has many advantages.

To simplify our study of the linear layer, we introduce a matrix representation for it. In a block cipher operating on w words, a linear layer may be expressed as a $w \times w$ block matrix. We will denote zero and identity sub-matrices by 0 and 1 respectively and an unspecified (arbitrary) sub-matrices by L. This information is sufficient for analyzing the high-level structure of a cipher. Using this notation, the linear layers to which we restrict our analysis have matrices where each column has at most one 1.

For the special subset of linear layers outlined above, we present an algorithm for obtaining MEDCP and MELCC bounds, that is based on a dynamic programming approach. Since there is no LT branching, any truncated trail consists of disjoint sequences of active S-Boxes. By Observation 1, we can treat each such sequence as a LT to obtain an optimal decomposition. Because of this simplification, we can avoid enumerating all trails by grouping them in a particular way.

We proceed round by round and maintain a set of best trails up to an equivalence relation, which is defined as follows. For all S-Boxes at the current last round s, we assign a number, which is equal to the length of the LT that covers this S-Box, or zero if the S-Box is not active. We say that two truncated trails for s steps are equivalent if the tuples consisting of those numbers (lengths of LTs) are the same for both trails. This equivalence captures the possibility to replace some prefix of a trail by an equivalent one without breaking the validity of the trail or its LT decomposition. The total probability, however, can change. The key observation here is that from two equivalent trails we can keep only the one with the highest current probability. Indeed, if the optimal truncated trail for all r rounds is an extension of the trail for s rounds with lower probability, we can take the first s rounds from the trail with higher probability without breaking anything and obtain a better trail, which contradicts the assumed optimality.

The pseudo-code for the algorithm is given in Appendix A.3.

This algorithm can be used to bound the probability of linear trails. Propagation of a linear mask through some linear layer can be described by multiplying the mask by the transposed inverse of the linear layer's matrix. In our matrix notation we can easily transpose the matrix but inversion is harder. However, we can build the linear trails bottom-up (i.e. starting from the last round): in this case we need only the transposed initial matrix. Our algorithm does not depend on the direction, so we obtain bounds on linear trails probabilities by running the algorithm on the transposed matrix using the linear bounds for the iterated S-box.

Ensuring Resilience Against Integral Attacks As illustrated by the structural attack against SASAS and a recent generalization [20] to ciphers with more rounds, a SPN with few rounds may be vulnerable to integral attacks. This attack strategy has been further improved by Todo [19] who proposed the so-called *division property* as a means to track which bit should be fixed in the input to have a balanced output. He also described an algorithm allowing an attacker to easily find such distinguishers.

We implemented this algorithm to search for division-property-based integral trails covering as many rounds as possible. With it, for each matrix candidate we compute a maximum number of rounds covered by such a distinguisher. This quantity can then be used by the designer of the primitive to see if the level of protection provided against this type of attack is sufficient or not.

Tracking the evolution of the division property through the linear layer requires special care. In order to do this, we first make a copy of each word and apply the required XORs from the copy to the original words. Due to such state expansion, the algorithm requires both a lot of memory and time. In fact, it is even infeasible to apply on some matrices. To overcome this issue, we ran the algorithm with reduced word size. During our experiments, we observed that such an optimization may only result in longer integral characteristics and that this side effect occurs only for very small word sizes (4 or 5 bits). In light of this, we conjecture that the values obtained in these particular cases are upper bounds and are very close to the values which could be obtained without reducing the word size.

4 The Sparx Family of Ciphers

In this Section, we describe a family of block ciphers built using the framework laid out in the previous section. The instance with block size n and key size k is denoted Sparx-n/k.

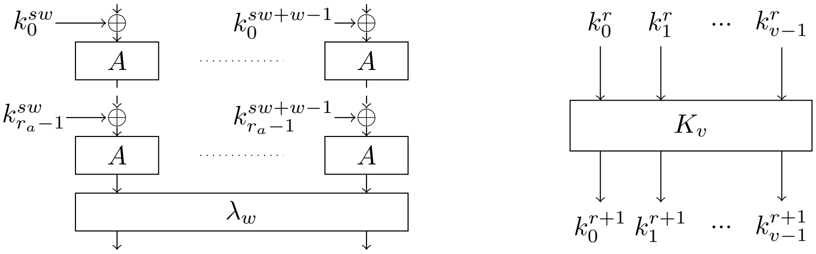
4.1 High Level View

The plaintexts and ciphertexts consist of $w = n/32$ words of 32 bits each and the key is divided into $v = k/32$ such words. The encryption consists of n_s steps, each composed of an ARX-box layer of r_a rounds and a linear mixing layer. In the ARX-box layer, each word of the internal state undergoes r_a rounds of Speckey, including key additions. The v words in the key state are updated once r_a ARX-boxes have been applied to one word of the internal state. The linear layers λ_w for $w = 2, 4$ provide linear mixing for the w words of the internal state.

This structure is summarized by the pseudo-code in Algorithm 1. The structure of one round is represented in Fig. 4, where A is the 32-bit ARX-box consisting in one unkeyed Speck-32 round. We also use A^a to denote a rounds of Speckey with the corresponding key additions (see Fig. 5a).

Algorithm 1 Sparx encryption
Inputs plaintext $(x_0, ..., x_{w-1})$; key $(k_0, ..., k_{v-1})$
Output ciphertext $(y_0, ..., y_{w-1})$

```
Let $y_i \leftarrow x_i$ for all $i \in [0, ..., w-1]$
for all $s \in [0, n_s - 1]$ do
    for all $i \in [0, w-1]$ do
        for all $r \in [0, r_a - 1]$ do
            $y_i \leftarrow y_i \oplus k_r$
            $y_i \leftarrow A(y_i)$
        end for
        $(k_0, ..., k_{v-1}) \leftarrow K_v\big((k_0, ..., k_{v-1})\big)$          ▷ Update key state
    end for
    $(y_0, ..., y_{w-1}) \leftarrow \lambda_w\Big((y_0, ..., y_{w-1})\Big)$          ▷ Linear mixing layer
end for
Let $y_i \leftarrow y_i \oplus k_i$ for all $i \in [0, ..., w-1]$          ▷ Final key addition
return $(y_0, ..., y_{w-1})$
```



(a) Round function of SPARX.

(b) Key schedule.

Fig. 4: A high level view of step s of SPARX.

The different versions of SPARX all share the same definition of A. However, the permutations λ_w and K_v depend on the block and key sizes. The different members of the SPARX-family are specified below. The round keys can either be derived on the fly by applying K_v on the key state during encryption or they can be precomputed and stored. The first option requires less RAM, while the second is faster. The only operations needed to implement any instance of SPARX are:

- addition modulo 2^{16}, denoted $\boxplus$,
- 16-bit exclusive-or (XOR), denoted $\oplus$, and
- 16-bit rotation to the left or right by i, denoted respectively $x \lll i$ and $x \ggg i$.

We claim that no attack using less than 2^k operations exists against SPARX-n/k in neither the single-key nor in the related-key setting. We also faithfully declare that we have not hidden any weakness in these ciphers. SPARX is free for use and its source code is available in the public domain [5].

4.2 Specification

Table 3 summarizes the different SPARX instances and their parameters. The quantity $\min_{\text{secure}}(n_s)$ corresponds to the minimum number of steps for which we can prove that the MEDCP is below 2^{-n}, that the MELCC is below $2^{-n/2}$ for the number of rounds per step chosen and for which we cannot find integral distinguishers covering this amount of steps.

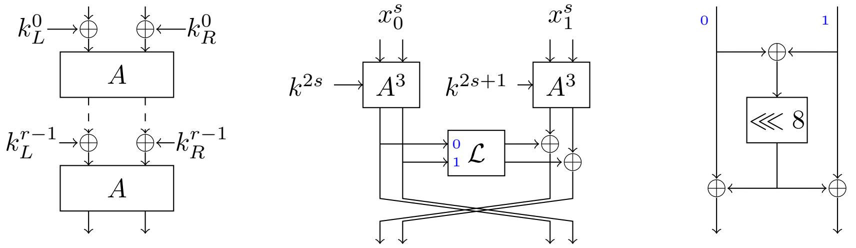
SPARX-64/128 The lightest instance of SPARX is SPARX-64/128. It operates on two words of 32 bits and uses a 128-bit key. There are 8 steps and 3 rounds per step. As it takes 5 steps to achieve provable security against linear and differential attacks, our security margin is at least equal to 37% of the rounds. Furthermore, while our Long Trail argument proves that 5 steps are sufficient to ensure that there are no single-trail differential and linear distinguishers, we do not expect this bound to be tight.

[5] See https://www.cryptolux.org/index.php/SPARX.

Table 3: The different Sparx instances.

	Sparx-64/128	Sparx-128/128	Sparx-128/256
# State words w	2	4	4
# Key words v	4	4	8
# Rounds/Step r_a	3	4	4
# Steps n_s	8	8	10
Best Attack (# rounds)	15/24	22/32	24/40
$\min_{\text{secure}}(n_s)$	5	5	5

The linear layer λ_2 simply consists of a Feistel round using $\mathcal{L}$ as a Feistel function. The general structure of a step of Sparx-64/128 is provided in Fig. 5b. The 128-bit permutation used in the key schedule has a simple definition summarized in Fig. 6, where the counter r is initialized to 0. It corresponds to the pseudo code given in Algorithm 2, where $(z)_L$ and $(z)_R$ are the 16-bit left and right halves of the 32-bit word z.



(a) A_k^r.

(b) Step structure.

(c) $\mathcal{L}$.

Fig. 5: A high level view of Sparx-64/128. Branches have a width of 16 bits (except for the keys in the step structure).

The $\mathcal{L}$ function is borrowed from Noekeon [21] and can be defined using 16- or 32-bit rotations. It is defined as a Lai-Massey structure mapping a 32-bit value $x||y$ to $x \oplus \big((x \oplus y) \lll 8\big)||y \oplus \big((x \oplus y) \lll 8\big)$. Alternatively, it can be seen as a mapping of a 32-bit value z to $z \oplus (z \lll^{32} 8) \oplus (z \ggg^{32} 8)$ where the rotations are over 32 bits.

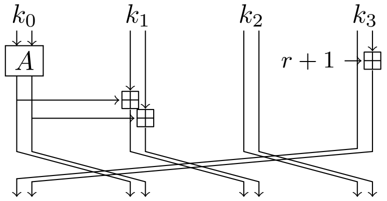
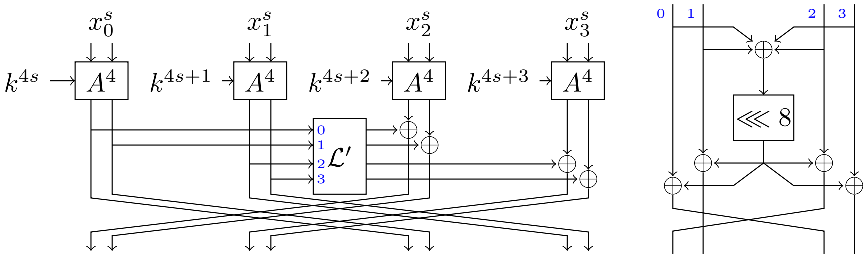


Fig. 6: K_4^{64} (used in Sparx-64/128).

```
$r \leftarrow r + 1$
$k_0 \leftarrow A(k_0)$
$(k_1)_L \leftarrow (k_1)_L + (k_0)_L \mod 2^{16}$
$(k_1)_R \leftarrow (k_1)_R + (k_0)_R \mod 2^{16}$
$(k_3)_R \leftarrow (k_3)_R + r \mod 2^{16}$
$k_0, k_1, k_2, k_3 \leftarrow k_3, k_0, k_1, k_2$
```

Algorithm 2: Pseudo-code of K_4^{64}

Sparx-128/128 and Sparx-128/256 For use cases in which a larger block size can be afforded, we provide Sparx instances with a 128-bit block size and 128- or 256-bit keys. They share an identical step structure which is fairly similar to Sparx-64/128. Indeed, the linear layer relies again on a Feistel function except that $\mathcal{L}$ is replaced by $\mathcal{L}'$, a permutation of $\{0,1\}^{64}$. Both Sparx-128/128 and Sparx-128/256 use 4 rounds per step but the first uses 8 steps while the last uses 10.



(a) Step structure.

(b) $\mathcal{L}'$.

Fig. 7: The step structure of both Sparx-128/128 and Sparx-128/256.

The Feistel function $\mathcal{L}'$ can be defined as follows. Let $a||b||c||d$ be a 64-bit word where each $a, ..., d$ is 16-bit long. Let $t = (a \oplus b \oplus c \oplus d) \lll 8$. Then $\mathcal{L}'(a||b||c||d) = c \oplus t \mid\mid b \oplus t \mid\mid a \oplus t \mid\mid d \oplus t$. This function can also be expressed using 32-bit rotations. Let $x||y$ be the concatenation of two 32-bit words and $\mathcal{L}'_b$ denote $\mathcal{L}'$ without its final branch swap. Let $t = \big((x \oplus y) \ggg^{32} 8\big) \oplus \big((x \oplus y) \lll^{32} 8\big)$, then $\mathcal{L}'_b(x||y) = x \oplus t||y \oplus t$. Alternatively, we can use $\mathcal{L}$ to compute $\mathcal{L}'_b$ as follows: $\mathcal{L}'_b(x||y) = y \oplus \mathcal{L}(x \oplus y)||x \oplus \mathcal{L}(x \oplus y)$.

These two ciphers, Sparx-128/128 and Sparx-128/256, differ only by their number of steps and by their key schedule. The key schedule of Sparx-128/128 needs a 128-bit permutation K_4^{128} described in Fig. 8 and Algorithm 3 while Sparx-128/256 uses a 256-bit permutation K_4^{256}, which is presented in both Fig. 9 and Algorithm 4.

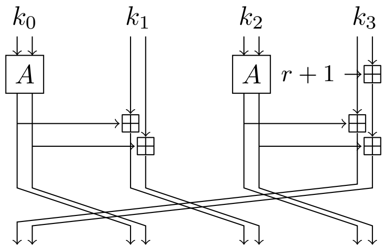


Fig. 8: The 128-bit permutation K_4^{128} used in Sparx-128/128.

```
$r \leftarrow r + 1$
$k_0 \leftarrow A(k_0)$
$(k_1)_L \leftarrow (k_1)_L + (k_0)_L \mod 2^{16}$
$(k_1)_R \leftarrow (k_1)_R + (k_0)_R \mod 2^{16}$
$k_2 \leftarrow A(k_2)$
$(k_3)_L \leftarrow (k_3)_L + (k_2)_L \mod 2^{16}$
$(k_3)_R \leftarrow (k_3)_R + (k_2)_R + r \mod 2^{16}$
$k_0, k_1, k_2, k_3 \leftarrow k_3, k_0, k_1, k_2$
```

Algorithm 3: Pseudo-code of K_4^{128}

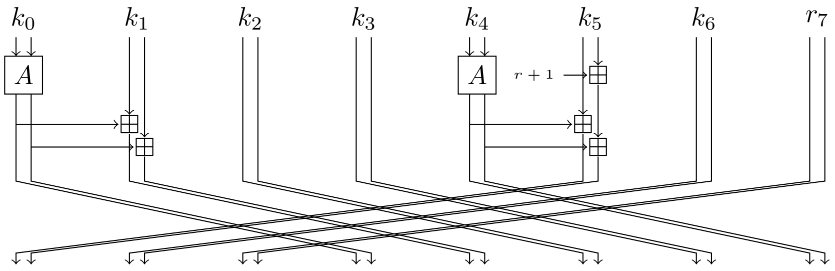


Fig. 9: The 256-bit permutation K_8^{256} used in Sparx-128/256.

Algorithm 4 Sparx-128/256 key schedule permutation K_8^{256}.

```
$r \leftarrow r + 1$
$k_0 \leftarrow A(k_0)$
$(k_1)_L \leftarrow (k_1)_L + (k_0)_L \mod 2^{16}$
$(k_1)_R \leftarrow (k_1)_R + (k_0)_R \mod 2^{16}$
$k_4 \leftarrow A(k_4)$
$(k_5)_L \leftarrow (k_5)_L + (k_4)_L \mod 2^{16}$
$(k_5)_R \leftarrow (k_5)_R + (k_4)_R + r \mod 2^{16}$
$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7 \leftarrow k_5, k_6, k_7, k_0, k_1, k_2, k_3, k_4$
```

4.3 Design Rationale

Choosing the arx-box. We chose the round function of Speckey/Speck-32 over Marx-2 because of its superior implementation properties. Indeed, its smaller total number of operations means that a cipher using it needs to do fewer operations when implemented on a 16-bit platform. Ideally, we would have used an ARX-box with 32-bit operations but, at the time of writing, no such function has known differential and linear bounds (cf. Table 1) for sufficiently many rounds.

We chose to evaluate the iterations of the ARX-box over each branch rather than in parallel because such an order decreases the number of times each 32-bit branch must be loaded in CPU registers. This matters when the number of registers is too small to contain both the full key and the full internal state of the cipher and does not change anything if it is not the case.

Mixing Layer, Number of Steps and Rounds per Step. Our main approach for choosing the mixing layer was exhaustive enumeration of all matrices suitable for our long trail bounding algorithm from Section 3.4 and selecting the final matrix according to various criteria, which we will discuss later.

For SPARX-64/128, there is only one linear layer fulfilling our design criteria: one corresponding to a Feistel round. For such a structure, we found that the best integral covers 4 steps (without the last linear layer) and that, with 3 rounds per step, the MEDCP and MELCC are bounded by 2^{-75} and 2^{-38}. These quantities imply that no single trail differential or linear distinguisher exists for 5 or more steps of SPARX-64/128.

For SPARX instances with 128-bit block we implemented an exhaustive search on a large subset of all possible linear layers. After some filtering, we arrived at roughly 3000 matrices. For each matrix we ran our algorithm from Section 3.4 to obtain bounds on MEDCP and MELCC for different values of the number of rounds per step (r_a). We also ran the algorithm for searching integral characteristics described in Section 3.4.

Then, we analyzed the best matrices and found that there is a matrix which corresponds to a Feistel-like linear layer with the best differential/linear bound for $r_a = 4$. This choice also offered good compromise between other parameters, such as diffusion, strength of the ARX-box, simplicity and easiness/efficiency of implementation. It also generalizes elegantly the linear layer of SPARX-64/128. We thus settled for this Feistel-like function.

For more details on the selection procedure and other interesting candidates for the linear layer we refer the reader to Appendix A.4.

The Linear Feistel Functions The linear layer obtained using the steps described above is only specified at a high level, it remains to define the linear Feistel functions $\mathcal{L}$ and $\mathcal{L}'$. The function $\mathcal{L}$ that we have chosen has been used in the Lai-Massey round constituting the linear layer of NOEKEON [21]. We reuse it here because it is cheap on lightweight processors as it only necessitates one rotation by 8 bits and 3 XORs. It also provides some diffusion as it has branching number 3. Its alternative representation using 32-bit rotations allows an optimized implementation on 32-bit processors.

Used for a larger block size, the Feistel function $\mathcal{L}'$ is a generalization of $\mathcal{L}$: it also relies on a Lai-Massey structure as well as a rotation by 8 bits. The reason behind these choices are the same as before: efficiency and diffusion. Furthermore, $\mathcal{L}'$ must also provide diffusion between the branches. While this is achieved by the XORs, we further added a branch swap in the bits of highest weight. This ensures that if only one 32-bit branch is active at the input of $\mathcal{L}'$ then two branches are active in its output. Indeed, there are two possibilities: either the output of the rotation is non-zero, in which case it gets added to the other branch and spreads to the whole state through the branch swap. Otherwise, the output is equal to 0, which means that the two 16-bit branches constituting the non-zero 32-bit branch hold the same non-zero value. These will then be spread over the two output 32-bit branches by the branch swap. The permutation $\mathcal{L}'$ also breaks the 32-bit word structure, which can help prevent the spread of integral patterns.

Key Schedule The key schedules of the different versions of SPARX have been designed using the following general guidelines.

First, we look at criteria related to the implementation. To limit code size, components from the round function of SPARX are re-used in the key-schedule itself. To accommodate cases where the memory requirements are particularly stringent, we allow an efficient on-the-fly computation of the key.

We also consider cryptographic criteria. For example, we need to ensure that the keys used within each chain of 3 or 4 ARX-boxes are independent from one another. As we do not have enough entropy from the master key to generate truly independent round keys, we must also ensure that the round-keys are as different as possible from one another. This implies a fast mixing of the master key bits in the key schedule. Furthermore, in order to prevent slide attacks [22], we chose to have the round keys depend on the round index. Finally, since the subkeys are XOR-ed in the key state, we want to limit the presence of high probability differential pattern in the key update. Diffusion in the key state is thus provided by additions modulo 2^{16} rather than exclusive-or. While there may be high probability patterns for additive differences, these would be of little use because the key is added by an XOR to the state.

As with most engineering tasks, some of these requirements are at odds against each other. For example, it is impossible to provide extremely fast diffusion while also being extremely lightweight. Our designs are the most satisfying compromises we could find.

4.4 Security Analysis

Single Trail Differential/Linear Attack By design and thanks to the Long Trail argument, we know that there is no differential or linear trail covering 5 steps (or more) with a useful probability for any instance of SPARX. Therefore, the 8 steps used by SPARX-64/128 and SPARX-128/128 and the 10 used by SPARX-128/256 are sufficient to ensure resilience against such attacks.

Attacks Exploiting a Slow Diffusion We consider several attacks in this category, namely impossible and truncated differential attacks, meet-in-the middle attacks as well as integral attacks.

When we chose the linear layers, we ensured that they prevented division-property-based integral attacks, meaning that they provide good diffusion. Furthermore, the Feistel structure of the linear layer makes it easy to analyse and increases our confidence in our designs. In the case of 128-bit block sizes, the Feistel function $\mathcal{L}'$ has branching number 3 in the sense that if only one 32-bit branch is active then the two output branches are active. This prevents attacks trying to exploit patterns at the branch level. Finally, this Feistel function also breaks the 32-bit word structure through a 16-bit branch swap which frustrates the propagation of integral characteristics.

Meet-in-the-middle attacks are further hindered by the large number of key additions. This liberal use of the key material also makes it harder for an attacker to guess parts of it to add rounds at the top or at the bottom of, say, a differential characteristic.

Best Attacks The best attacks we could find are integral attacks based on Todo's division property. The attack against SPARX-64/128 covers 15/24 rounds and recovers the key in time 2^{101} using 2^{37} chosen plaintexts and 2^{64} blocks of memory. For 22-round SPARX-128/128, we can recover the key in time 2^{105} using 2^{102} chosen plaintexts and 2^{72} blocks of memory. Finally, we attack 24-round SPARX-128/256 in time 2^{233}, using 2^{104} chosen plaintexts and 2^{202} blocks of memory. A description of these attacks as well as the description of some time/data tradeoffs are provided in the Appendix A.2.

4.5 Software Implementation

Next we describe how SPARX can be efficiently implemented on three resource constrained microcontrollers widely used in the Internet of Things (IoT), namely the 8-bit Atmel ATmega128, the 16-bit TI MSP430, and the 32-bit ARM Cortex-M3. We support the described optimization strategies with performance figures extracted from assembly implementations of SPARX-64/128 and SPARX-128/128 using the FELICS open-source benchmarking framework [23]. We use the same tool to get the most suitable implementations of SPARX for the two IoT-specific usage scenarios described in [6]. The first scenario uses a block cipher to encrypt 128 bytes of data using CBC mode, while the second encrypts 128 bits of data using a cipher in CTR mode. The most suitable implementation for a given usage scenario is selected using the *Figure of Merit (FOM)* defined in [6]:

$$\mathrm{FOM}(i_1, i_2, i_3) = \frac{p_{i_1,AVR} + p_{i_2,MSP} + p_{i_3,ARM}}{3},$$

where the performance parameter $p_{i,d}$ aggregates the code size, the RAM consumption, and the execution time for implementation i according to the requirements of the usage scenario. The smaller the FOM value of an implementation in a certain use case, the better (more suitable) is the implementation for that particular use case. Finally, we compare the results of our implementations with the results available on the tool's website.[6]

Table 4: Performance characteristics of the main components of Sparx

Component	AVR		MSP		ARM	
	cycles	registers	cycles	registers	cycles	registers
A	16	$4+1$	9	2	11	$1+3$
A^{-1}	19	4	9	2	12	$1+3$
λ_2 – 1-step	24	$8+1$	11	$4+3$	5	$2+1$
λ_2 – 2-steps	12	8	7	$4+1$	3	2
λ_4 – 1-step	48	$16+2$	36	$8+1$	16	$4+5$
λ_4 – 2-steps	24	$16+2$	13	$8+1$	12	$4+4$

Implementation Aspects In order to efficiently implement Sparx on a resource constrained embedded processor, it is important to have a good understanding of its instruction set architecture (ISA). The number of general-purpose registers determines whether the entire cipher's state can be fitted into registers or whether a part of it has to be spilled to RAM. Memory operations are generally slower than register operations, consume more energy and increase the vulnerability of an implementation to side channel attacks [24]. Thus, the number of memory operations should be reduced as much as possible. Ideally the state should only be read from memory at the beginning of the cryptographic operation and written back at the end. Concerning the three targets we implemented Sparx for, they have 32 8-bit, 12 16-bit, and 13 32-bit general-purpose registers, which result in a total capacity of 256 bits, 192 bits, and bits bytes for AVR, MSP, and ARM, respectively.

The Sparx family's simple structure consists only of three components: the ARX-box A and its inverse A^{-1}, the linear layer λ_2 or λ_4 (depending on the version), and the key addition. The key addition (bitwise XOR) does not require additional registers and its execution time is proportional to the ratio between the operand width and the target device's register width. The execution time in cycles and the number of registers required to perform A, A^{-1}, λ_2, and λ_4 on each target device are given in Table 4.

The costly operation in terms of both execution time and number of required registers is the linear layer. The critical point is reached for the 128-bit linear

[6] We submitted our implementations of Sparx to the FELICS framework. Up to date results are available at https://www.cryptolux.org/index.php/FELICS.

layer λ_4 on MSP, which requires 13 registers. Since this requirement is above the number of available registers, a part of the state has to be saved onto the stack. Consequently, the execution time increases by 5 cycles for each `push` – `pop` instruction pair.

A 2-step implementation uses a simplified linear layer without the most resource demanding part – the branch swaps. It processes the result of the left branch after the first step as the right branch of the second step and similarly the result of the right branch after the first step as the left branch of the second step. This technique reduces the number of required registers and improves the execution time at the cost of an increase in code size. The performance gain is a factor of 2 on AVR, 2.7 on MSP, and 1.3 on ARM.

Table 5: Different trade-offs between the execution time and code size for encryption of a block using Sparx-64/128 and Sparx-128/128. Minimal values are given in bold.

Implementation	Block	AVR			MSP			ARM		
	size	Time	Code	RAM	Time	Code	RAM	Time	Code	RAM
	[bits]	[cyc.]	[B]	[B]	[cyc.]	[B]	[B]	[cyc.]	[B]	[B]
1-step rolled	64	1789	**248**	2	1088	**166**	14	1370	**176**	28
1-step unrolled	64	1641	424	**1**	907	250	12	1100	348	**24**
2-steps rolled	64	1677	356	2	1034	232	10	1331	304	28
2-steps unrolled	64	**1529**	712	**1**	**853**	404	**8**	**932**	644	**24**
1-step rolled	128	4553	**504**	11	2809	**300**	26	3463	**348**	44
1-step unrolled	128	4165	1052	**10**	2353	584	24	2784	884	40
2-steps rolled	128	4345	720	11	2593	432	18	3399	620	40
2-steps unrolled	128	**3957**	1820	**10**	**2157**	1004	**16**	**2377**	1692	**36**

The linear transformations $\mathcal{L}$ and $\mathcal{L}'$ exhibit interesting implementation properties. For each platform there is a different optimal way to perform them. The optimal way to implement the linear layers on MSP is using the representations from Fig. 5c and Fig. 7b. On ARM the optimal implementation performs the rotations directly on 32-bit values. The function $\mathcal{L}$ can be executed on AVR using 12 XOR instructions and no additional registers. On the other hand, the optimal implementation of $\mathcal{L}'$ on AVR requires 2 additional registers and takes 24 cycles. [7]

The linear layer performed after the last step of Sparx can be dropped without affecting the security of the cipher, but it turns out that it results in poorer overall performances. The only case when this strategy helps is when top

[7] For more details please see the implementations submitted to the FELICS framework (`https://www.cryptolux.org/index.php/FELICS`).

execution time is the main and only concern of an implementation. Thus we preferred to keep the symmetry of the step function and the overall balanced performance figures.

The salient implementation-related feature of Sparx family of ciphers is given by the simple and flexible structure of the step function depicted in Fig. 4, which can be implemented using different optimization strategies. Depending on specific constraints, such as code size, speed, or energy requirements to name a few, the rounds inside the step function can be rolled or unrolled; one or two step functions can be computed at once. The main possible trade-offs between the execution time and code size are explored in Table 5.

Except for the 1-step implementation of Sparx-128/128 on MSP, which needs RAM memory to save the cipher's state, all other RAM requirements are determined only by the process of saving the context onto the stack at the begging of the measured function. Thus, the RAM consumption of a pure assembly implementation would be zero, except for the 1-step rolled and unrolled implementations of Sparx-128/128 on MSP.

Table 6: The performance metrics of the balanced (globally efficient) implementations of Sparx-64/128 and Sparx-128/128 as revealed using the Figure of Merit (FOM) defined in FELICS.

Block	AVR			MSP			ARM			FOM
size	Time	Code	RAM	Time	Code	RAM	Time	Code	RAM	
[bits]	[cyc.]	[B]	[B]	[cyc.]	[B]	[B]	[cyc.]	[B]	[B]	
Scenario 1 – Encryption of 128 bytes of data using CBC mode										
64	30256	358	10	16113	338	22	19131	456	56	8.6
128	37984	614	19	24056	404	36	30466	428	68	12.9
Scenario 2 – Encryption of 128 bits of data using CTR mode										
64	4397	662	51	2261	580	52	2338	654	72	8.3
128	5478	1184	74	3057	1036	72	2935	1468	104	12.4

Due to the 16-bit nature of the cipher, performing A and A^{-1} on a 32-bit platform requires a little bit more execution time and more auxiliary registers than performing the same operations on a 16-bit platform. The process of packing and unpacking a state register to extract and store back the two 16-bit branches of A or A^{-1} adds a performance penalty. The cost is amplified by the fact that the flexible second operand can not be used with a constant to extract the least or most significant 16 bits of a 32-bit register. Thus an additional masking register is required.

The simple key schedules of Sparx-64/128 and Sparx-128/128 can be implemented in different ways. The most efficient implementation turns out to be the one using the 1-iteration rolled strategy. Another interesting approach is the 4-iterations unrolled strategy, which has the benefit that the final permutation is achieved for free by changing the order in which the registers are stored in the round keys. This strategy increases the code size by up to a factor of 4, while the execution time is on average 25% better.

Although we do not provide performance figures for Sparx-128/256, we emphasize that the only differences with respect to implementation aspects between Sparx-128/256 and Sparx-128/128 are the key schedules and the different number of steps.

Evaluation and Comparison We evaluate the performance of our implementations of Sparx using FELICS in the two aforementioned usage scenarios. The key performance figures are summarized in Table 6. The balanced results are achieved using the 1-step implementations of Sparx-64/128 and Sparx-128/128.

Table 7: Top 10 best implementations in Scenario 1 (encryption key schedule + encryption and decryption of 128 bytes of data using CBC mode) ranked by the Figure of Merit (FOM) defined in FELICS. The results for all ciphers are the current ones from the Triathlon Competition at the moment of submission. The smaller the FOM, the better the implementation.

Rank	Cipher	Block size	Key size	Scenario 1 FOM
1	Speck	64	128	5.0
2	Chaskey-LTS	128	128	5.0
3	Simon	64	128	6.9
4	RECTANGLE	64	128	7.8
5	LEA	128	128	8.0
6	**Sparx**	**64**	**128**	**8.6**
7	**Sparx**	**128**	**128**	**12.9**
8	HIGHT	64	128	14.1
9	AES	128	128	15.3
10	Fantomas	128	128	17.2

Then we compare the performance of Sparx with the current results available on the Triathlon Competition at the time of submission. [8] As can be seen in Table 7 the two instances of Sparx perform very well across all platforms and rank very high in the FOM-based ranking. The forerunners are the NSA designs Simon and Speck, Chaskey, RECTANGLE and LEA, but, apart from

[8] Up to date results are available at https://www.cryptolux.org/index.php/FELICS.

RECTANGLE, none of them provides provable bounds against differential and linear cryptanalysis.

Besides the overall good performance figures in the two usage scenarios, the following results are worth mentioning:

– the execution time of Sparx-64/128 on MSP is in the top 3 of the fastest ciphers in both scenarios thanks to its 16-bit oriented operations;
– the code size of the 1-step rolled implementations of Sparx-64/128 and Sparx-128/128 on MSP is in the top 5 in both scenarios as well as in the small code size and RAM table for scenario 2;
– the 1-step rolled implementation of Sparx-64/128 breaks the previous minimum RAM consumption record on AVR in scenario 2;
– the execution time of the 2-steps implementation of Sparx-64/128 in scenario 2 is in the top 3 on MSP, in the top 5 on AVR, and in the top 7 on ARM; it also breaks the previous minimum RAM consumption records on AVR and MSP.

Given its simple and flexible structure as well as its very good overall ranking in the Triathlon Competition of lightweight block ciphers, the Sparx family of lightweight ciphers is suitable for applications on a wide range of resource constrained devices. The absence of look-up tables reduces the memory requirements and provides, according to [24], some intrinsic resistance against power analysis attacks.

5 Replacing Rotations with Linear Layers: the LAX Construction

In this section we outline an alternative strategy for designing an ARX cipher with provable bounds against differential and linear cryptanalysis. It is completely independent from the Long Trail Strategy outlined in the previous sections and uses the differential properties of modular addition to derive proofs of security.

5.1 Motivation

In his Master thesis [13] Wallén posed the challenge to design a cipher that uses only addition modulo-2 and GF(2)-affine functions, and that is provably resistant against differential and linear cryptanalysis [13, Sect. 5]. In this section we partially solve this challenge by proposing a construction with provable bounds against single-trail differential cryptanalysis (DC).

5.2 Theoretical Background

Definition 6 (xdp$^+$)**.** *The* XOR *differential probability (DP) of addition modulo* 2^n *is defined as:*

$$\mathrm{xdp}^+(\alpha, \beta \to \gamma) = 2^{-2n} \cdot \#\{(x, y) : ((x \oplus \alpha) + (y \oplus \beta)) \oplus (x + y) = \gamma\} \enspace ,$$

where α, β and γ are n-bit XOR differences and x and y are n-bit values.

The XOR linear correlation of addition modulo 2^n (xlc^+) is defined in a similar way. Efficient algorithms for the computation of xdp^+ and xlc^+ have been proposed resp. in [25] and [26,27,28]. These results also reveal the following property. The magnitude of both xdp^+ and $|\mathrm{xlc}^+|$ is inversely proportional to the number of bit positions at which the input/output differences (resp. masks) differ. For xdp^+, this fact is formally stated in the form of the following proposition.

Proposition 1 (Bound on xdp^+). *The differential probability xdp^+ is upper-bounded by 2^{-k}, where k is the number of bit positions, excluding the MSB, at which the bits of the differences are not equal:*

$$\mathrm{xdp}^+(\alpha, \beta \to \gamma) \leq 2^{-k} : k = \#\{i : \neg(\alpha[i] = \beta[i] = \gamma[i]), 0 \leq i \leq w - 2\}$$

Proof. Follows from [25, Alg. 2, Sect. 4].

A similar proposition also holds for $|\mathrm{xlc}^+|$ (see e.g. [10]). Proposition 1 provides the basis of the design strategy described in the following section.

5.3 The LAX Construction

LAX is a block cipher construction with $2n$-bit block and n-bit words. We investigate three instances of LAX designated by the block size: LAX-16, LAX-32 and LAX-64. A brief description of the round function of LAX-$2n$, shown in Fig. 10 (left), is given below.

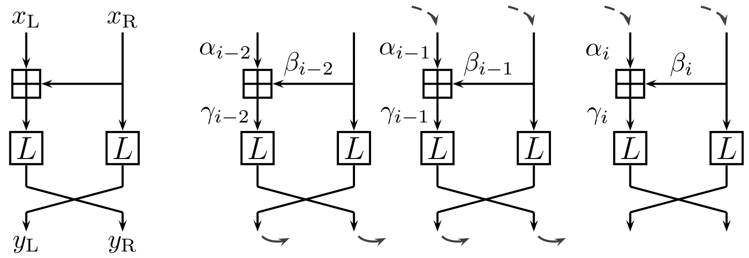


Fig. 10: **Left:** the round function of LAX; **Right:** three round differential of LAX.

Let L be an $n \times n$ binary matrix that is (a) invertible and (b) has branch number $d > 2$. With $\ell(x)$ is denoted the multiplication of the n-bit vector x by the matrix L: $\ell(x) = Lx$. Note that due to condition (b) it follows that $\forall x \neq 0 : h(x) + h(\ell(x)) \geq d$, where $h(x)$ is the Hamming weight of x.

The round function $\mathcal{A}(\cdot)$ of LAX-$2n$ maps a pair of n-bit words $(x_{\mathrm{L}}, x_{\mathrm{R}})$ to a pair of n-bit words $(y_{\mathrm{L}}, y_{\mathrm{R}})$ as follows (see Fig. 10 (left)):

$$(y_{\mathrm{L}}, y_{\mathrm{R}}) = \mathcal{A}(x_{\mathrm{L}}, x_{\mathrm{R}}) = (\ell(x_{\mathrm{R}}),\ \ell(x_{\mathrm{L}} \boxplus x_{\mathrm{R}}))\ \ .$$

The matrix L is chosen as the non-identity part of the generator matrix G of a systematic $[2n, n, d]$ linear code over GF(2) such that $G = [I\ L]$. More specifically, the matrices L for LAX-16, LAX-32 and LAX-64 are derived from the following codes respectively: $[16, 8, 5]$, $[32, 16, 8]$ and $[64, 32, 10]$. Note that the matrix of LAX-32 is the same as the one used in block cipher ARIA [29].

5.4 Bounds on the Differential Probability of LAX

Lemma 1. *For all differences $\alpha \neq 0$, the differential $(\alpha, \alpha \rightarrow \alpha)$ is impossible.*

Proof. Let $\mathrm{xdp}^{+}(\alpha, \beta \rightarrow \gamma) \neq 0$ for some differences α, β and γ. The statement of the lemma follows from the following two properties of xdp^{+} [25]. First, it must hold that $\alpha[0] \oplus \beta[0] \oplus \gamma[0] = 0$. Second, if $\alpha[i] = \beta[i] = \gamma[i]$ for some $0 \leq i \leq n-2$, then it must hold that $\alpha[i+1] \oplus \beta[i+1] \oplus \gamma[i+1] = \alpha[i]$. Since we want that $\alpha = \beta = \gamma$, from the first property it follows that $\alpha[0] = \beta[0] = \gamma[0] = 0$. Given that, due to the second property it follows that $\alpha[i] = \beta[i] = \gamma[i] = 0$, $\forall i \geq 1$. Therefore the only value of α for which $\mathrm{xdp}^{+}(\alpha, \beta \rightarrow \gamma) \neq 0$ and $\alpha = \beta = \gamma$ is $\alpha = 0$. □

Theorem 2 (Differential bound on 3 rounds of LAX-$2n$). *The maximum differential probability of any trail on 3 rounds of* LAX-$2n$ *is $2^{-(d-2)}$, where d is the branch number of the matrix L.*

Proof. Let $(\alpha_{i-1}, \beta_{i-1}, \gamma_{i-1})$, $(\alpha_i, \beta_i, \gamma_i)$ and $(\alpha_{i+1}, \beta_{i+1}, \gamma_{i+1})$ be the input/output differences of the addition operations in three consecutive rounds of LAX-$2n$ and let $p_k = \mathrm{xdp}^{+}(\alpha_k, \beta_k \rightarrow \gamma_k)$ for $k \in \{i-1, i, i+1\}$ (see Fig. 10 (right)). We have to show that $p_{i-1}p_ip_{i+1} \leq 2^{-(d-2)}$ or, equivalently, that $\log_2 p_{i-1} + \log_2 p_i + \log_2 p_{i+1} \leq -(d-2)$. Denote with $h(x)$ the Hamming weight of the word x and with $h^*(x)$ the Hamming weight of x, excluding the MSB. Note that $h^*(x) \leq h(x) - 1$. We consider two cases:

Case 1: $\beta_{i-1} \neq \gamma_{i-1}$. By Proposition 1 we have that $\log_2 p_{i-1} \leq -h^*(\beta_{i-1} \oplus \gamma_{i-1})$ and $\log_2 p_i \leq -h^*(\alpha_i \oplus \beta_i)$. Since $\beta_i = \ell(\gamma_{i-1})$ and $\alpha_i = \ell(\beta_{i-1})$ (see Fig. 10 (right)) and using the linearity of $\ell(\cdot)$ we have that $-h^*(\alpha_i \oplus \beta_i) = -h^*(\ell(\beta_{i-1} \oplus \gamma_{i-1}))$. As $\beta_{i-1} \neq \gamma_{i-1}$ it follows that $h^*(\beta_{i-1} \oplus \gamma_{i-1}) \neq 0$ and $h^*(\ell(\beta_{i-1} \oplus \gamma_{i-1})) \neq 0$. Thus we derive:

$$\log_2 p_{i-1} + \log_2 p_i \leq -h^*(\beta_{i-1} \oplus \gamma_{i-1}) - h^*(\ell(\beta_{i-1} \oplus \gamma_{i-1}))\ \ .$$

From the properties of L it follows that $-h(\beta_{i-1} \oplus \gamma_{i-1}) - h(\ell(\beta_{i-1} \oplus \gamma_{i-1})) \leq -d$ and so $-h^*(\beta_{i-1} \oplus \gamma_{i-1}) - h^*(\ell(\beta_{i-1} \oplus \gamma_{i-1})) \leq -(d-2)$. Therefore:

$$\log_2 p_{i-1} + \log_2 p_i \leq -(d-2)\ \ .$$

Case 2: $\beta_{i-1} = \gamma_{i-1} \neq 0$. In this case $\alpha_i = \beta_i = \ell(\beta_{i-1}) = \ell(\gamma_{i-1})$. Due to Lemma 1 it follows that $\gamma_i \neq \beta_i$. Therefore we can apply the argument from *Case 1* on rounds i and $i+1$ to derive the statement of the theorem in this case. □

5.5 Experimental Results

We have implemented the search algorithm proposed in [10] in order to find the probabilities of the best differential trails in LAX-16 and LAX-32. In Table 8, we compare the results to the theoretical bounds computed using Theorem 2.

Table 8: Best differential probabilities and best absolute linear correlations ($\log_2$ scale) for up to 12 rounds of LAX.

	# Rounds	1	2	3	4	5	6	7	8	9	10	11	12
LAX-16	p_{best}	+0	−2	−4	−7	−8	−11	−13	−16	−18	−20	−23	−25
	c_{best}	+0	+0	−1	−2	−3	−5	−5	−7	−8	−9	−10	−11
	p_{bound}			−3			−6			−9			−12
LAX-32	p_{best}	+0	−2	−6	−9	−11	−16	−18	−20	−24	−28	−29	−34
	c_{best}	+0	+0	+0	−4	−4	−8	−8	−8	−8	−12	−12	−16
	p_{bound}			−6			−12			−18			−24

Clearly the bound from Theorem 2 does not hold for the linear case. The problem is the "three-forked branch" in the LAX round function that acts as an XOR when the inputs are linear masks rather than differences. Thus, LAX only provides differential bounds and the full solution to the Wallén challenge still remains an open problem.

6 Conclusion

In this paper we presented, for the first time, a general strategy for designing ARX primitives with provable bounds against differential (DC) and linear cryptanalysis (LC) – a long standing open problem in the area of ARX design. The new strategy, called the Long Trail Strategy (LTS) advocates the use of large and computationally expensive S-boxes in combination with very light linear layers (the so-called Long Trail Argument). This makes the LTS to be the exact opposite of the Wide Trail Strategy (WTS) on which the AES (and many other SPN ciphers) are based. Moreover, the proposed strategy is not limited to ARX designs and can easily be applied also to S-box based ciphers.

To illustrate the effectiveness of the LTS we have proposed a new family of lightweight block ciphers, called SPARX, designed using the new approach. The family has three instances depending on the block and key sizes: Sparx-64/128, Sparx-128/128 and Sparx-128/256. With the help of the Long Trail Argument we prove resistance against single-trail DC and LC for each of the three instances of Sparx. In addition, we analyze the new constructions against a wide range of attacks such as impossible and truncated differentials, meet-in-the-middle and integral attacks. Our analysis did not find any attack covering at least 70% of the rounds of any of the three instances. The latter ensures a security margin of more than 30% for all instances of Sparx.

Beside (provable) security the members of the Sparx family are also very efficient. We have implemented them in software on three resource constrained microcontrollers widely used in the Internet of Things (IoT), namely the 8-bit Atmel ATmega128, the 16-bit TI MSP430, and the 32-bit ARM Cortex-M3. According to the FELICS open-source benchmarking framework our implementations of Sparx-64/128 and Sparx-128/128 rank respectively 6 and 7 in the list of top 10 most software efficient lightweight ciphers. In addition, the execution time of Sparx-64/128 on MSP is in the top 3 of this list. To the best of our knowledge, this paper is the first to propose a practical ARX design that has both arguments for provable security and competitive performance.

A secondary contribution of the paper is the proposal of an alternative strategy for ARX design with provable bounds against differential cryptanalysis. It is independent of the LTS and uses the differential properties of modular addition to derive proofs of security. As an illustration of this approach, the LAX family of constructions is described. The provable security of LAX against linear cryptanalysis is left as an open problem.

7 Acknowledgements

The work of Daniel Dinu and Léo Perrin is supported by the CORE project ACRYPT (ID C12-15-4009992) funded by the Fonds National de la Recherche, Luxembourg. The work of Aleksei Udovenko is supported by the Fonds National de la Recherche, Luxembourg (project reference 9037104). Vesselin Velichkov is supported by the Internal Research Project CAESAREA of the University of Luxembourg (reference I2R-DIR-PUL-15CAES). The authors thank Anne Canteaut for useful discussions regarding error correcting codes.

References

1. Bernstein, D.J.: New Stream Cipher Designs: The eSTREAM Finalists. Springer Berlin Heidelberg (2008) 84–97
2. Bernstein, D.J.: ChaCha, a variant of Salsa20. In: Workshop Record of SASC. Volume 8. (2008)
3. Niels, F., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein hash function family. Submission to NIST **(round 3)** (2010)
4. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 Proposal BLAKE. https://131002.net/blake/blake.pdf (2010)

5. Needham, R.M., Wheeler, D.J.: Tea extensions. Technical report, Cambridge University, Cambridge, UK (October 1997)
6. Dinu, D.D., Le Corre, Y., Khovratovich, D., Perrin, L., Großschädl, J., Biryukov, A.: Triathlon of Lightweight Block Ciphers for the Internet of Things. In: NIST Workshop on Lightweight Cryptography 2015, National Institute of Standards and Technology (NIST) (2015)
7. Mouha, N., Mennink, B., Herrewege, A., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In: Selected Areas in Cryptography – SAC 2014: 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers. Springer International Publishing, Cham (2014) 306–323
8. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive **2013** (2013) 404
9. Hong, D., Lee, J.K., Kim, D.C., Kwon, D., Ryu, K.H., Lee, D.G.: LEA: A 128-bit block cipher for fast encryption on common processors. In: Information Security Applications. Springer (2013) 3–27
10. Biryukov, A., Velichkov, V., Le Corre, Y.: Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck. In Peyrin, T., ed.: Fast Software Encryption. Volume 3557 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2016) To Appear
11. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Cryptography and Coding: 8th IMA International Conference Cirencester, UK, December 17–19, 2001 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2001) 222–238
12. Daemen, J., Rijmen, V.: The design of Rijndael: AES-the advanced encryption standard. Springer (2002)
13. Wallén, J.: On the Differential and Linear Properties of Addition. Master's thesis, Helsinki University of Technology (2003)
14. Keliher, L., Sui, J.: Exact maximum expected differential and linear probability for 2-round Advanced Encryption Standard. IET Information Security **1**(2) (2007) 53–57
15. Nikolić, I.: Tiaoxin-346. Submission to the CAESAR competition (2015)
16. Jean, J., Nikolić, I.: Efficient Design Strategies Based on the AES Round Function. In Peyrin, T., ed.: Fast Software Encryption. Volume 3557 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2016) To Appear
17. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Cryptographic Hardware and Embedded Systems – CHES 2011: 13th International Workshop, Nara, Japan, September 28 – October 1, 2011. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 326–341
18. Knudsen, L., Wagner, D.: Integral Cryptanalysis. In: Fast Software Encryption: 9th International Workshop, FSE 2002 Leuven, Belgium, February 4–6, 2002 Revised Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2002) 112–127
19. Todo, Y.: Structural Evaluation by Generalized Integral Property. In: Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Springer Berlin Heidelberg, Berlin, Heidelberg (2015) 287–314
20. Biryukov, A., Khovratovich, D.: Decomposition attack on SASASASAS. Cryptology ePrint Archive, Report 2015/646 (2015) http://eprint.iacr.org/.
21. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie proposal: NOEKEON. In: First Open NESSIE Workshop. (2000) 213–230

22. Biryukov, A., Wagner, D.: Slide Attacks. In: Fast Software Encryption: 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (1999) 245–259
23. Dinu, D.D., Biryukov, A., Großschädl, J., Khovratovich, D., Le Corre, Y., Perrin, L.a.: FELICS–Fair Evaluation of Lightweight Cryptographic Systems. In: NIST Workshop on Lightweight Cryptography 2015, National Institute of Standards and Technology (NIST) (2015)
24. Biryukov, A., Dinu, D., Großschädl, J.: Correlation power analysis of lightweight block ciphers: From theory to practice. In: International Conference on Applied Cryptography and Network Security – ACNS 2016. Volume 9696 of Lecture Notes in Computer Science., Springer (2016) 537–557
25. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In Matsui, M., ed.: FSE. Volume 2355 of LNCS., Springer (2001) 336–350
26. Wallén, J.: Linear Approximations of Addition Modulo 2^n. In Johansson, T., ed.: FSE 2003. Volume 2887 of Lecture Notes in Computer Science., Springer (2003) 261–273
27. Nyberg, K., Wallén, J.: Improved linear distinguishers for SNOW 2.0. In: Fast Software Encryption, Springer (2006) 144–162
28. Dehnavi, S.M., Rishakani, A.M., Shamsabad, M.R.M.: A More Explicit Formula for Linear Probabilities of Modular Addition Modulo a Power of Two. Cryptology ePrint Archive, Report 2015/026 (2015) http://eprint.iacr.org/.
29. Kwon, D., Kim, J., Park, S., Sung, S.H., Sohn, Y., Song, J.H., Yeom, Y., Yoon, E.J., Lee, S., Lee, J., Chee, S., Han, D., Hong, J.: New Block Cipher: ARIA. In: Information Security and Cryptology - ICISC 2003: 6th International Conference, Seoul, Korea, November 27-28, 2003. Revised Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2004) 432–445
30. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2001) 213–230

A Appendix

A.1 Test Vectors for Sparx

Test vectors are shown as 16-bit words in hexadecimal notation.

A.2 Best Attacks Against Sparx

An Integral Distinguisher for 128-bit Blocks .

Consider an instance of Sparx operating on 128-bit blocks. Using Todo's division property, we found that if we fix the left-most 32-bit word of the plaintext and let the other 3 take all possible 2^{96} values, then the output of the two right-most word at the end of the last S-Box layer of the 5th step are balanced. This pattern is destroyed by the linear layer of the 5th step.

However, because half of the A function merely undergoes linear transformations, and because the LSB of modular addition is a linear function, it is possible

Sparx-64/128

key	0011 2233	4455 6677	8899 aabb	ccdd eeff
plaintext	0123 4567	89ab cdef		
ciphertext	2bbe f152	01f5 5f98		

Sparx-128/128

key	0011 2233	4455 6677	8899 aabb	ccdd eeff
plaintext	0123 4567	89ab cdef	fedc ba98	7654 3210
ciphertext	1cee 7540	7dbf 23d8	e0ee 1597	f428 52d8

Sparx-128/256

key	0011 2233	4455 6677	8899 aabb	ccdd eeff
	ffee ddcc	bbaa 9988	7766 5544	3322 1100
plaintext	0123 4567	89ab cdef	fedc ba98	7654 3210
ciphertext	3328 e637	14c7 6ce6	32d1 5a54	e4b0 c820

to extend this distinguisher by one more round. First, the last operations of A can be inverted for free because they are linear, as summarized in Figure 11, to obtain new 16-bit values $a, b, ..., h$, as defined in the same picture where we also define the quantities t, u, v and w.

Consider a structure of size 2^{96} obtained by encrypting 2^{96} plaintexts where the left-most 32-bit word is fixed and the other take all possible values. We use $z[j]$ to denote the j-th bit of a 16-bit word z where the ordering is from LSB to MSB. Let i be the index of the ciphertexts in our structure. Then the following equation holds with probability 1:

$$\bigoplus_{i=0}^{2^{96}-1} \big((e^i[0] \oplus f^i[0]) \oplus f^i[14] \oplus (g^i[0] \oplus h^i[0]) \oplus h^i[14]\big) \oplus (b^i[10] \oplus f^i[10]) \ = \ 0. \quad (1)$$

Indeed, the sum of these values and the corresponding key bits yields the value of $u^i[10]$ which sums to 0 over the structure, regardless of the key bits. If we look at $w^i[10]$ instead of $u^i[10]$, we can derive another equation:

$$\bigoplus_{i=0}^{2^{96}-1} \big((e^i[0] \oplus f^i[0]) \oplus f^i[14] \oplus (g^i[0] \oplus h^i[0]) \oplus h^i[14]\big) \oplus (d^i[10] \oplus h^i[10]) \ = \ 0. \quad (2)$$

Equations (1) and (2) hold for any key and for any such structure of 2^{96} ciphertexts. They can be used to attack both Sparx-128/128 and Sparx-128/256.

Integral Attack Against 22/(24)-round Sparx-128/128(256) .

We use this distinguisher to attack 22-round Sparx-128/128. It is naturally extended to attack 6 steps (24 rounds) of Sparx-128/256 by guessing keys for two more rounds. We will use the partial sums technique introduced by Ferguson *et al.* in [30]. The idea of the attack is to encrypt several structures of plaintexts

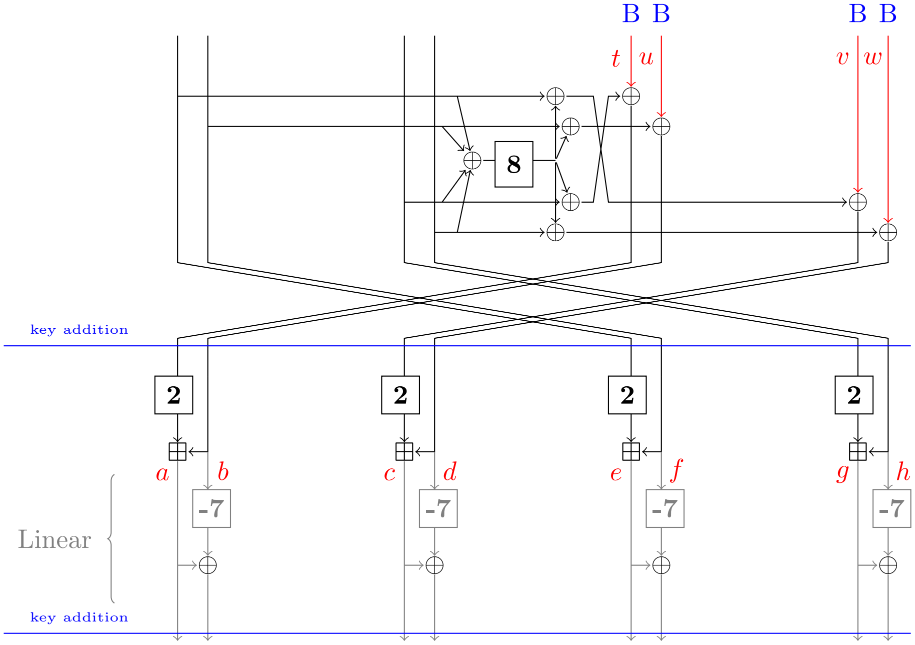


Fig. 11: Principle of the integral attack against SPARX instances operating on 128 bits. Bold numbers denote bit rotations to the left. The letter B denotes balanced 16-bit words at the end of the S-Box layer of the fifth step.

and then by using partial sums to split the key guessing work into two parts and finally to filter out wrong keys. An outline of the attack follows.

1. Encrypt 64 structures of 2^{96} plaintexts such that the left-most word is fixed inside each structure and the other three take all possible values.
2. For each structure and each word position:
   (a) Store all word values which occur an odd number of times in this position in all ciphertexts from the structure. On average there will be 2^{31} such values per structure per position.
3. Initialize a hash table indexed by 128-bit blocks.
4. For all 2^{64} possible keys K_0, K_1:
   (a) For each structure, decrypt one round of Speckey on all stored words for positions 0 and 1 using keys K_0 and K_1 respectively. Compute the contribution of decrypted bits to sums from Equations (1) and (2). Such contributions form a 128-bit string s (two bits per structure).
   (b) Add $K_0||K_1$ to the hash table with index s.
5. For all 2^{64} possible keys K_2, K_3:
   (a) Decrypt one round of Speckey on all stored words for positions 2 and 3 using keys K_2, K_3 and compute the contributions similarly to the previous step. Since XOR of the contributions from left and right halves must be equal to zero for each structure, the contributions must be equal.
   (b) Check if the 128-bit contribution string is in the hash table. If it is there, get the corresponding key $K_0||K_1$ and save $K_0||K_1||K_2||K_3$ as a key candidate.
6. On average we will obtain 1 key candidate for the last round. Then we can decrypt the last round and run another attack or we can exploit the key schedule and reconstruct all round keys.

Step 2 requires $64 \cdot 2^{96} \cdot 4 = 2^{104}$ simple operations. Complexity of steps 4 and 5 is equal to $2^{64} \cdot 64 \cdot 2 \cdot 2^{31} = 2^{102}$ Speckey round decryptions. Therefore full attack complexity can be bounded by 2^{105} operations. The data complexity is equal to $64 \cdot 2^{96} = 2^{102}$. To store the hashtable we need around $2^{64} \cdot (2 \cdot 64 + 64) < 2^{72}$ memory blocks.

The 24-round attack on Sparx-128/256 is very similar. We encrypt 192 structures and guess keys required to decrypt three Speckey rounds instead of one. The complexity then is dominated by steps 4 and 5 and is equal to $2 \cdot 2^{192} \cdot 192 \cdot 2 \cdot 2^{31} < 2^{233}$ operations. The data complexity is $192 \cdot 2^{96} < 2^{104}$ chosen plaintexts. The memory complexity is around $2^{192} \cdot (2 \cdot 192 + 192) < 2^{202}$ blocks.

We note that by exploiting the key schedule we can reduce the complexities by not guessing a repeating key material. However, we did not manage to attack more rounds.

Integral Attack Against 15-round Sparx-64/128 .

If we encrypt a structure built by setting the left side to a constant and letting the right side take all possible values using Sparx-64/128 then, with probability 1, the right side after 3 steps (minus the last linear layer) must take all 2^{32} possible values. Using multi-set terminology, a permutation on the right side of the input becomes a permutation on the right side of the output. We can use this property as a distinguisher for 12 rounds. Furthermore, it turns out that the division property does not yield any stronger result in this case.

As for the attacks against 128-bit block variants, we can append one round after the end of the 4th step and derive a linear equation linking the LSB of the modular addition on the right with 2 other bits from the right hand side and 1-bit from the left hand side. The sum of these bits over the ciphertexts in a structure must be equal to 0.

We append two rounds at the end of this distinguisher to attack 5 steps of this block cipher. To do this, we repeat the following procedure several times.

1. Encrypt a structure of 2^{32} plaintexts.
2. Invert the last linear linear.
3. For all 64-bit key k_L, partially decrypt 2 rounds for the left-hand side and store k_L in a list indexed by the xor of the 3 interesting bits from this side.
4. For all 64-bit key k_R, partially decrypt 2 rounds for the right-hand side and store k_R in list indexed by the unique interesting bit from this side.

Each time we repeat this procedure, we re-distribute the key guesses in list indexed by the concatenation of the indices in the successive structures. For example, if k^R is in list 1 for the first structure and list 0 for the second one, we place it in a general list with index $01 = 1$. If it is placed in list 1 by the third structure, we move it to the general list $101 = 5$, and so on.

If we repeat this procedure u times, we obtain 2^{128-u} key candidates of 128 bits from which we deduce full key candidates that are tested with trial encryptions. We thus have a time/data tradeoff: with only one structure, we can only recover 1 bit from the key. Conversely, with all 128 structures, we recover the full key.

The treatment of each structure requires 2^{32+64} operations and the storage of 2^{32} ciphertexts. At all times, we must also store all 2^{64} candidates for both k_L and k_R. The complexity of the final brute-force is 2^{128-u}. If $u \geq 32$, the bottle-neck in terms of time is the treatment of all u structures.

In the end, we attack 5 steps (i.e. 15 rounds) of Sparx-64/128 using about 2^{64} blocks of memory, $2^{32} \times u$ chosen plaintexts and $u2^{96} = 2^{128-u}$ operations. In particular, if we use 32 structure, we can break this cipher in time 2^{101} using 2^{37} chosen plaintexts.

A.3 Algorithms for Bounding MEDCP and MELCC of an SPN Cipher Using a Long Trail Strategy

Decomposing a truncated trail into long trails Here we sketch an algorithm for finding an optimal decomposition of a given truncated trail into long trails.

First, note that the trail can be represented as a graph, where nodes are active S-Boxes and an edge corresponds to a possible connection of two S-Boxes in a long trail. Moreover, this graph is a forest. Indeed, an S-Box can't receive two edges from the previous round, because it contradicts a definition of long trail - there must be a single difference coming in. For each tree in the forest, we choose the root to be the S-Box from the earliest round, which is determined uniquely by the same reason. Then, for any node its children may be only in the next round.

The goal then is to cover all nodes with disjoint vertical paths, such that the product of the paths' probabilities is minimal. By the path probability we understand the respective long trail's probability. The simplest (and the worst) solution is to choose paths consisting of single nodes. Note that this solution already gives some upper bound and by finding a better decomposition we improve this bound.

We propose an algorithm based on recursive dynamic programming approach. For each node, we recursively solve the sub-problem for the subtree rooted at that node. But we need to compute some additional information besides the best decomposition of the subtree. Consider the optimal decomposition of the whole forest into such paths and consider the long trail which goes through the current subtree's root. Clearly, if we fix this long trail, the rest of the subtree becomes completely independent and has to be decomposed optimally. Therefore, from the subtree we need to know only the probability of this decomposition and the length of the long trail's part in the subtree. We don't know the optimal length beforehand, therefore we store the best probabilities for all possible lengths. Another view on this is that we group all possible subtree decompositions by length of the long trail which goes through the subtree root and for each such length we greedily choose the minimum probability. Then, when we obtain such tables for all children of some node, we can easily compute the table for the node itself - we check all possible ways to choose a child of the node and the length of the long trail which goes through the child and we try to join the current node to that long trail. Then the corresponding probability is the product of the best probabilities of the other children with the probability corresponding to the children's long trail and the probability stored in the children's table respectively for that length.

Once a table for a node is computed, there will be only one pass through it - when we will compute the table for its parent. Therefore, the node's contribution to the complexity is at most $O(r)$, where r is the height of its subtree and is bounded by the number of rounds. The total complexity of the algorithm then is at most $O(wr^2)$, where wr corresponds to the total number of S-Boxes in the cipher.

Efficient algorithm for special linear layers. In Section 3.4 we explained an algorithm to bound MEDCP and MELCC of an SPN-based cipher, in which the linear layer has a particular property.

The pseudo-code is given in Algorithm 5. It is possible to modify the algorithm such that it returns the best truncated trail too. To do this we can maintain a mapping of trails to their "parent" trail so that when we obtain the final probability, we can reconstruct the full trail step by step from the end.

A.4 Choosing the linear layer for Sparx-128/128 and Sparx-128/256

In this Appendix we describe in details the procedure that we followed when designing the linear layers for 128-bit Sparx instances. The basic approach is to exhaustively check all possible linear layers for which we could prove MEDCP and MELCC bounds.

Recall that the constraint on matrices from Section 3.4 is that there must be *at most* one 1 in each column and in each row (because we run the algorithm on the matrix transpose as well). For simplicity and implementation reasons, we now consider matrices with *exactly* one 1 in each row and column. Such matrices also foster the appearance of long trails.

The matrices we look at correspond to permutations of 4 words with some zeroes possibly replaced by special elements which we denote by L. Though there may be several elements L in the matrix, it is not necessary that all the corresponding small linear functions are equal. The total number of such matrices is $4! \cdot 2^{12} = 98304$.

First, we observe that for any matrix M and for any word permutation matrix P, the matrices M and $P \times M \times P^{-1}$ are equivalent to reordering arx-boxes in the whole cipher. Thus, we keep only one representative from each such class. Next, we drop the matrices which do not provide full diffusion. We applied two different techniques to check this and the results matched. The first way is to use symbolic computation. The second one is based on replacing each L with some random small matrix (e.g. 5×5) and evaluating the matrix several times on all inputs with one active word. We stop if the number of steps reached the limit of 20 or if the full diffusion is reached earlier. After this filtering step we had only 3282 matrices left.

For all reasonable numbers of steps and rounds in a step, we ran our algorithm to obtain bounds on MELCC and MEDCP. We also searched for integral characteristics using the division property in order to both ensure good diffusion and to estimate resilience against this type of attack.

Note that the integral characteristic search does not depend on the number of rounds per step because we analyze only the high-level structure. However, the MELCC and MEDCP bounds do depend on this value, so we have to make the choice. Two rounds per step completely contradict our analysis simplification about randomness of the ARX box. Whereas for five or more rounds per step we have to take fewer steps and the cipher may become susceptible to structural attacks (for example [20]). Therefore, we consider only the cases $r_a = 3$ and $r_a = 4$.

Also note that some matrices have many "L", making them both hard to analyse and to implement. To prevent this issue, we considered different cases based on the number of words which are copied from the input to the output

Algorithm 5 Finding the best bound on the DP of a truncated trail (for LC the matrix should be transposed).
Inputs number of steps r; $w \times w$ matrix M over $\{0, 1, L\}$, with at most one 1 at each column; non-decreasing bounds on DP (or LC) $(P[1], \ldots, P[r])$
Output probability of the best truncated trail *best_prob*

```
Let $S_0 \leftarrow \{0,1\}^w \setminus \{0^w\}$, $pr_0[s] = 1.0$ for all $s \in S_0$     ▷ 0 - inactive, 1 - LT of length 1.
for all $i \in [0, r-1]$ do
    Let $S_{i+1} = \{\}$
    for all $s \in S_i$ do
        for all $(s', p') \in Extensions(s, pr_i[s])$ do
            add $s'$ to set $S_{i+1}$
            $pr_{i+1}[s'] \leftarrow max(pr_{i+1}[s'], p')$
        end for
    end for
end for
return $max(pr_r[s]$ for $s \in S_r)$

function EXTENSIONS(s, p)
    Let $out\_states \leftarrow []$
    for all $cancel \in \{false, true\}^w$ do
        Let $s' \leftarrow 0^w, p' \leftarrow p$
        for all $o \in [0, w-1]$ do
            Let $mask \leftarrow$ (if $s_i > 0$ then $M_{o,i}$ else 0 for $i \in [0, w-1]$)
            if $mask$ contains single 1 then
                $i \leftarrow$ index of 1 in $mask$
                $s'[o] \leftarrow s[i] + 1$
                $p' \leftarrow p' + P[s[i]+1] - P[s[i]]$                ▷ Extending an LT
            else if $mask$ contains single $L$ then
                $s'[o] \leftarrow 1$                ▷ An LT is broken by the linear layer
                $p' \leftarrow p' + P[1]$
            else if $mask$ contains at least two nonzero elements then
                if cancel[o] then
                    $s'[o] \leftarrow 0$                ▷ Differences cancelled
                else
                    $s'[o] \leftarrow 1$
                    $p' \leftarrow p' + P[1]$                ▷ Differences not cancelled
                end if
            end if
        end for
        if $s' \neq 0^w$ then
            append $(s', p')$ to $out\_states$
        end if
    end for
    return $out\_states$
end function
```

without change. More copies results in easier and more efficient implementation, easier identification of long trails but weaker diffusion.

Finally, we selected the best matrices according to one of the following two criteria.

1. Minimizing the differential/linear trail probability. We compute the number of steps when the trail probability bound derived by the algorithm is less than 2^{-128} for differential trails and less than 2^{-64} for linear.
2. Minimizing the number of steps of the integral characteristic found with division property.

The results are given in Table 9 and Table 10, where $+S$ denotes an additional ARX-box layer.

Table 9: The best linear layers for $r_a = 3$.

#words copied	optim. for	best int. char.	min. rounds (diff./linear)	matrix
0	diff./linear	4	7/7	`[10L0,010L,L001,0L10]`
	diffusion	2+S	8/8	`[10L0,01L0,LLL1,001L]`
1	diff./linear	4+S	7/7	`[001L,0001,10L0,L10L]`
	diffusion	3+S	7/8	`[001L,0001,100L,L1LL]`
2	diff./linear	7+S	6/6	`[00L1,1000,L100,0010]` (a)
	diffusion	3+S	8/11	`[0010,0001,1LLL,L1LL]`
	tradeoff	4+S	7/7	`[0001,1L0L,0100,0L1L]` (b)
3	diff./linear	9+S	7/7	`[LL01,1000,0100,0010]`
	diffusion	7+S	8/9	`[LLL1,1000,0100,0010]`

The results show that heavier matrices (without words copied) lead to better diffusion, as expected, whereas for linear/differential security the matrices with 2 words copied give best results for both $r_a = 3$ and $r_a = 4$. Though heavy matrices can give a good compromise between these two criteria, they are hard to implement, to study and, in particular it is hard to implement their inverses. Thus, we decided to stick to light matrices.

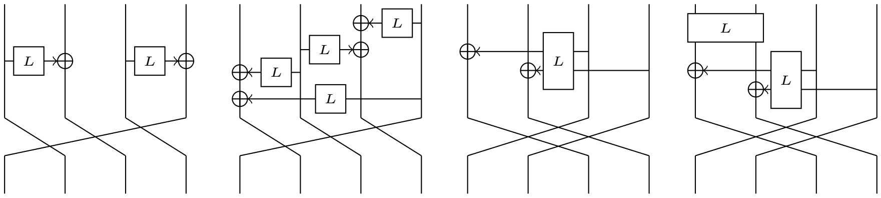
The most interesting matrices are marked by (a),(b),(c),(d) and the structures of the corresponding layers are depicted in Fig. 12. For $r_a = 3$ the matrix with the best differential/linear security, (a), yields an integral characteristic covering almost 8 steps. Another interesting matrix, (b), requires 7 steps which corresponds to 21 rounds. For $r_a = 4$, we can achieve differential/linear security in 5 steps (20 rounds) using matrix (c): the MEDCP for 20 rounds is bounded by 2^{-138} and the MELCC by 2^{-68}. Notably, this matrix is a Feistel round

Table 10: The best linear layers for $r_a = 4$.

#words copied	optim. for	best int. char.	min. rounds (diff./linear)	matrix
0	diff./linear	3	5/5	[L010,00L1,1L0L,01L0]
	diffusion	2+S	6/5	[10L0,01L0,LLL1,001L]
1	both	3+S	5/5	[10LL,01L0,LLL1,0010]
2	diff./linear	4+S	5/5	[0010,0001,10LL,01LL] (c)
	diffusion	3+S	5/6	[0010,0001,1LLL,L1LL] (d)
3	both	7+S	5/6	[LLL1,1000,0100,0010]

operating on 32 bit words. Matrix (d) is similar but it adds additional mixing between the two left branches, which improves diffusion but slightly weakens differential/linear security.

A cipher built with $r_a = 4$ and matrix (c) provides a good compromise between differential/linear security, diffusion, strength of the ARX-box, simplicity and easiness/efficiency of implementation. It also generalizes elegantly the linear layer of Sparx-64/128. We thus settle for this Feistel-like function. For convenience, we decided to use its mirrored version.



(a) (b) (c) (d)

Fig. 12: Possible linear layers.