Advanced cryptology (GBX9SY06)

✧

Block ciphers 3

Pierre Karpman

2017-11

1 Substitution boxes

A large number of block ciphers use *substitution boxes* (or S-boxes) as components. Those are simply functions with small (binary) domains that are extensively defined and that may be used in various stages of an algorithm.

Definition 1 (S-box)**.** An *S-box* is a (usually injective) mapping $S: \{0,1\}^b \rightarrow \{0,1\}^{b'}$, where one usually has $b = b' \in \{4, 8\}$.

Early ciphers did not necessarily use injective S-boxes (most famously DES), or even fixed ones (e.g. Blowfish, where large S-boxes are generated as part of the key schedule). However, as this is now much less common, we will focus on the "modern" usage of S-boxes which fit the typical profile of Definition 1, that is 4 or 8-bit permutations that are explicitly defined by a table. Beyond DES and Blowfish-like S-boxes, this excludes among others the 3-bit S-box of PRINTCIPHER [KLPR10], the 5-bit S-box of SHA-3 [Nat15] and the 16-bit S-box of WHIRLWIND [BNN+10]. As an example, we give the table of the third S-box of Serpent [BAK98] in Table 1.

Table 1: Serpent S6 S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	7	2	C	5	8	4	6	B	E	9	1	F	D	3	A	0

S-boxes are often used as the only non-linear component of a cipher. It is thus important that they are themselves sufficiently resistant against attacks that exploit non-linearity such as typically differential, linear and algebraic cryptanalysis. Because of the limited size of their inputs and outputs, S-boxes can be analysed "exhaustively" with respect to the properties underlying these attacks. For instance, while it is untractable to compute the differential probability of a differential for a large (say, 128-bit) permutation, doing so for an 8-bit S-box is very easy. The same can be said of linear approximations and algebraic normal forms. One can adapt the definitions of these notions to the specific case of S-boxes in the following way.

Definition 2 (Differential uniformity of an S-box)**.** Let S be an n-bit S-box. We define its *difference distribution table* (or DDT) as the function δ_S defined extensively by:

$$\delta_S(a,b) := \#\{x \in \mathbf{F}_2^n \,|\, S(x) + S(x+a) = b\}.$$

The *differential uniformity* $\mathscr{D}$ of S is defined as:

$$\max_{(a,b)\neq(0,0)} \delta_S(a,b).$$

Put another way, an n-bit S-box with differential uniformity $\mathscr{D}$ has a maximum differential probability of $\mathscr{D}/2^n$ over its inputs, for any differential.

Definition 3 (Linearity of an S-box)**.** Let S be an n-bit S-box. We define its *linear approximation table* (or LAT) as the function ℓ_S defined extensively by:

$$\ell_S(a,b) := \sum_{x\in\mathbf{F}_2^n} (-1)^{\langle b,S(x)\rangle+\langle a,x\rangle}.$$

The *linearity* $\mathscr{L}$ of S is defined as:

$$\max_{(a,b)\neq(0,0)} |\ell_S(a,b)|.$$

This means that the absolute correlation of any linear approximation of an n-bit S-box of linearity $\mathscr{L}$ is upper-bounded by $\mathscr{L}/2^n$. For example, the S-box of Table 1 has differential uniformity 4 and linearity 8. It turns out that those are in fact optimal for a 4-bit S-box [LP07]

Intuitively, it seems to be the case that an S-box with low differential uniformity and linearity is necessary to design an efficient and secure cipher: anytime a non-trivial difference (say) is present on the inputs of an S-box (one then says that the S-box is *(differentially) active*), their outputs will verify any difference with probability at most $\mathscr{D}/2^n$, which is useful knowledge for a designer. Consequently, 8-bit S-boxes with low uniformity (for instance 4) and linearity seem to be systematically better than 4-bit ones, for instance, because a single-bit input difference to a cipher activates a single S-box. The same arguments can be made w.r.t. the ANF: an 8-bit S-box may have a maximal degree of 7 versus 3 for a 4-bit one, and the Boura et al. upper-bound on the degree of iterated round functions grows comparatively faster with large S-boxes as well.

Despite the above arguments, small 4-bit S-boxes are also used intensively along larger 8-bit ones. In fact, the choice of the size of an S-box is a good example of the difficulty of finding the "right" trade-off in cryptographic designs. A first point is that good 8-bit S-boxes are systematically much more expensive to implement than good 4-bit ones (whether the cost function is hardware circuit, lookup tables, etc.). A second point is that S-boxes are never used in isolation; a good 8-bit S-box used with a poor diffusion layer in an SPN may result in a weaker round function that a 4-bit S-box with a good diffusion layer. All in all, there are many settings where a small S-box is as good a choice as a large one, if not better.

Even when the size of an S-box is fixed, several strategies are available to design an S-box. These include random generation, optimal (or near-optimal) generation, or building the S-box as a small block cipher, from even smaller components. We will now describe each of these approaches.

1.1 Random generation of S-boxes

A simple way to define an S-box is simply to draw all of its entries at random. Given a sufficiently good PRNG, this is elementary if the S-box does not need to be invertible. In the more likely case where the S-box is a permutation, a less trivial algorithm such as the Fisher-Yates/Knuth uniform shuffle can be used. Assuming uniform randomness is available and on input N this algorithm uniformly samples a permutation among all $N!$ permutations of N elements. In effect, this defines an ideal cipher over a small domain. We give a description in Algorithm 1. Note that if line 3 of Algorithm 1 is replaced by $s \overset{\$}{\leftarrow} \{0,\ldots,N-1\}$, one obtains something similar to the shuffle used e.g. in RC4. This modified shuffle is not uniform, as it defines N^N equally likely permutations, and $N! \nmid N^N$.

This strategy for generating S-boxes is not very good. While a uniformly chosen permutation is nearly guaranteed to offer good resistance against differential, linear, and algebraic attacks, it is almost certainly possible to do better w.r.t. to the first two for the typical S-box sizes. Furthermore, a random S-box will likely be expensive to implement, as it is by definition unstructured.

```
Input: A table $T$ of size $N$ holding all $N$ numbers $0 \dots N-1$ each exactly once.
Output: A permutation of $0 \dots N-1$ uniformly selected among the $N!$ possible ones.
1 begin
2     for $i := 0;\ i < N-1$ do
3         $s \xleftarrow{\$} \{i, \dots, N-1\}$
4         Swap $T[i]$ and $T[s]$
5     end
6     return $T$
7 end
```

Algorithm 1: Uniform shuffle

An alternative, more successful approach that can be seen as random generation is to pseudo-randomly generate a circuit or a sequence of instructions implementing a permutation, and then to check if the result has good cryptographic properties [UDI+11]. If one only considers "efficient" circuits (w.r.t. some given cost function), one is guaranteed to obtain an S-box with an efficient implementation, at condition that the algorithm terminates. This approach was quite successful in finding 4-bit S-boxes with very low (in fact minimal) cost, e.g. using only four linear (XOR) and four non-linear (not XOR) two-to-one gates.

1.2 (Near-)Optimal generation

Ideally, one would want to know S-boxes that have (near-)optimal resistance against some classes of attacks. For instance, this could mean having a differential uniformity as close to the minimum of two as possible. Let us first consider the impact of this approach for 4-bit S-boxes. There are only $16! \approx 2^{44.2}$ possible 4-bit S-boxes, which is not a small amount by any means, but still possible to exhaustively enumerate. If the important characteristics of an S-box (say, differential uniformity, linearity and maximum degree on all output bits) can be tested efficiently enough, then it should be possible to sample all possible S-boxes and detect which ones are the best.

This type of exhaustive search has been performed several times [BCBP03, LP07, Saa11], but in a more clever way that is significantly more efficient. The idea is to notice that properties such as differential uniformity and linearity are invariant by composition with an affine mapping. That is, for any S-box S and any $M_i, M_o \in \mathrm{GL}_4(\mathbf{F}_2), \mathbf{a}, \mathbf{b} \in \mathbf{F}_2^4$, the S-box $S' : x \mapsto M_o \times S(M_i \times x \oplus \mathbf{a}) \oplus \mathbf{b}$ is such that $\mathscr{D}_S = \mathscr{D}_{S'}$ and $\mathscr{L}_S = \mathscr{L}_{S'}$. Thus, one only needs to enumerate permutations of 16 elements up to affine equivalence and to keep classes with the best cryptographic properties; each of these class can then be compactly represented by a single S-box[1]. If one then wishes to test additional criteria, for an S-box (and assuming that optimal resistance to differential and linear cryptanalysis remains important, which is usually the case), it is enough to do so on members of the known "optimal" classes. All in all, there are 16 such classes (cf. e.g. [LP07]) of 4-bit S-boxes of differential uniformity 4 and linearity 8. Note however that the degree of an S-box is not invariant by affine composition, neither have to be some global properties of a cipher such as its maximum expected differential probability [CR15]. The notions of optimality and equivalence of S-boxes described above should be understood as relative to some specific criteria, and not as absolute.

For 8-bit S-boxes, restricting to affine-equivalent classes is not enough to make a similar exhaustive search manageable. Yet, it is possible to explicitly design S-boxes with very good cryptographic properties. In fact, such an approach is also possible for smaller sizes, including 4-bit S-boxes. It is however quite more interesting for larger S-boxes because of the limitations of exhaustive enumeration.

[1]It is worth pointing out that the inverse of an S-box S is not necessarily in the same class as S

An efficient way to design very good n-bit S-boxes is to use as a basis the inversion map ι of $\mathbf{F}^*_{2^n}$, extended to zero by $\iota(0) = 0$. One can show that the resulting S-box (using a suitable mapping from $\mathbf{F}_{2^n}$ to $\{0,1\}^n$) has differential uniformity 4 if n is even and 2 if it is odd, and linearity $2^{n/2+1}$ (in the even case) [Nyb93]. For instance, this means that a 4-bit S-box generated by this process is a optimal w.r.t. to these criteria. In the 8-bit case, this means that one can easily generate S-boxes of differential uniformity 4 and linearity 32. The AES S-box, as well as many others, was designed in such a way.

An important remark is that directly taking ι as an S-box can lead to attack if the S-box is used within an $\mathbf{F}_{2^n}$-linear round function. In that case, the whole round function admits a simple algebraic description over that field, which can lead to powerful attacks [JK97]. This should be taken as a further warning that statistical approaches, while important to consider, are by no means the only way to attack a block cipher. In practice, it is however easy to avoid such algebraic attacks, e.g. by composing ι with a binary matrix that has no simple algebraic expression over the larger field $\mathbf{F}_{2^n}$. This is for instance what is done with the AES S-box.

It is remarkable that for n odd, an S-box generated from such an invert mapping has the lowest possible differential uniformity. This is not quite the case for n even, and one knows that the theoretical lower-bound cannot be reached for e.g. $n = 4$. Some isolated examples of differentially-optimal mappings for some even values of n do exist [BDMW10], but it is in general an open problem to determine the best reachable differential uniformity for all cases. As far as 8-bit S-boxes are concerned, this question remains unanswered.

From an implementation perspective, S-boxes based on the inversion are not lightweight; this is particularly true in the 8-bit case. While this is not an issue if the S-box is implemented as a table lookup, this approach is most of the time not the one to be preferred. Even if the prevalence of the AES means that much effort has been spent developing efficient implementations of its S-box, it remains quite more expensive than more structured alternative, which we describe next.

1.3 S-boxes as small block ciphers

It is possible to rethink entirely the design of S-boxes as trying to build small secure block ciphers with efficient implementations. While this statement is ignoring some subtleties (for instance consequences of the fact that S-boxes are unkeyed), this view opens the way to many design possibilities. First, considering that 4-bit S-boxes can be efficiently designed *from scratch*, one only needs to consider larger cases. Then, smaller S-boxes may precisely be used as subcomponents of higher-level constructions that for instance take a 4-bit S-box as input and define an 8-bit one. Finally, these constructions may be borrowed from block cipher round-function design, such as e.g. Feistel structures. We now take this as an opportunity to catalogue many such constructions which have successfully been used in block cipher and S-box design.

1.3.1 Feistel structure

The Feistel structure is altogether one of the oldest in block cipher design. A 3-round Feistel can be used to define a $2n$-bit S-box S from 3 n-bit S-boxes S_0, S_1, S_2 as:

1. $x^L_1 := x^R_0;\quad x^R_1 := S_0(x^R_0) \oplus x^L_0;$
2. $x^L_2 := x^R_1;\quad x^R_2 := S_1(x^R_1) \oplus x^L_1;$
3. $x^L_3 := x^R_2;\quad x^R_3 := S_2(x^R_2) \oplus x^L_2;$

and $S(x^L_0||x^R_0) = x^L_3||x^R_3$.

An interesting property of the Feistel structure is that it can be proven that in a block cipher context, three rounds amplify the resistance to differential and linear cryptanalysis by squaring the differential uniformity and the squared correlation (which is a function of the linearity) of the worst of the three small S-boxes. However, this is a statement that applies to a *keyed* context when each entry to a small S-box is masked with a random key, and where the amplification is obtained for the MEDP and MELP of the resulting cipher. That is, *on average*, an 8-bit S-box built from a 3-round Feistel using smaller components will be much more resistant to statistical attacks than the small S-boxes themselves.

Taken literally, this result is not very useful, as an unkeyed S-box needs to fix all parameters, and average notions are thus meaningless. However, it can be seen as quite encouraging: one can randomly sample many distinct instantiations of this construction, evaluate the characteristics of the result, and stop whenever a satisfactory one has been found. In fact, it is very easy to find the desired amplification properties by using this process. In the 4-to-8 case, this means that one can easily design 8-bit S-boxes with differential uniformity 16 and linearity 64 by using four optimal 4-bit S-boxes as $S_{0,1,2}$. This is quite weaker than, what one would get with an inverse mapping, but not too expensive to implement if the 4-bit S-boxes are well-chosen.

It is interesting to remark that because the Feistel structure does not require the round function to be invertible, one is not restricted to injective 4-bit S-boxes. This observation may lead to better S-boxes, as non-injective mappings may be found that have stronger differential properties. This was used by Canteaut et al. to design an 8-bit S-box from a 3-round Feistel that is very cheap to implement and that has differential uniformity only 8 [CDL15]. However, the linearity of this S-box is 64, which gives it an uneven resistance to typical differential and linear cryptanalysis. Yet there are some contexts where an increased resistance to differential cryptanalysis specifically may be useful.

1.3.2 Misty structure

The Misty structure was first used in the Misty block cipher [Mat97], that also pioneered the recursive approach of building S-boxes as small block ciphers. This structure is very close to a Feistel one, but requires the round function to be invertible. A 3-round Misty benefits from the same amplification properties as a 3-round Feistel. Such a structure, when building a $2n$-bit S-box S from 3 n-bit S-boxes S_0, S_1, S_2 can be defined as:

1. $x_1^L := x_0^R$; $x_1^R := S_0(x_0^L) \oplus x_0^R$;
2. $x_2^L := x_1^R$; $x_2^R := S_1(x_1^L) \oplus x_1^R$;
3. $x_3^L := x_2^R$; $x_3^R := S_2(x_2^L) \oplus x_2^R$;

and $S(x_0^L||x_0^R) = x_3^L||x_3^R$. All in all, the Misty structure also allows to build good large S-boxes from smaller ones, similarly as the Feistel case. Because of invertibility constraints of the small S-boxes, it is however not possible to reach particularly low differential uniformity as in the Feistel case [CDL15].

1.3.3 Lai-Massey structure

Our last example is the Lai-Massey structure, that was introduced in the block cipher IDEA [LMM91]. Similarly as Misty, it requires invertible S-boxes. In its simplest form, this structure builds a $2n$-bit S-box S from 3 n-bit S-boxes S_0, S_1, S_2 as:

1. $x_1 := S_0(x_0^R \oplus x_0^L)$;
2. $x_2^L := S_1(x_0^L \oplus x_1)$;
3. $x_2^R := S_1(x_0^R \oplus x_1)$;

and $S(x_0^L||x_0^R) = x_2^L||x_2^R$. A slightly more complex variant using five S-boxes was used in the (second version of the) 8-bit S-box of the WHIRLPOOL hash function [BR03], which lead to a good differential uniformity of 8 and linearity of 56. When using only three small S-boxes, a typical Lai-Massey 8-bit S-box has differential uniformity and linearity comparable to the average Feistel or Misty case, that is 16 and 64 respectively.

It is interesting to remark that Lai-Massey S-boxes lead to a very simple *impossible differential* pattern: if a pair of inputs to S has the same difference on the two halves x_0^L and x_0^R, there is no difference in the output of S_0 and thus there is *always* a difference in the input (and thence output) of S_1 and S_2. The probability of all differentials of the form $\Delta||\Delta \rightarrow 0||\delta$ and $\Delta||\Delta \rightarrow \delta||0$ is therefore equal to zero. While there is no obvious way to exploit this property in an attack, is should again serve as a useful reminder that as important as they can be, a few key properties are unlikely to capture all important behaviours of a cryptographic mapping.

2 Implementing S-boxes

By definition (at least in our restricted sense), an S-box is an explicit permutation with a small domain. Consequently, a straightforward way to compute the application of an S-box is to define a table that stores the image of x at index x. This table is easy to compute from the definition of the S-box (usually itself given in this very form, see Table 1).

This approach is relatively efficient, but it is not without drawbacks. Mostly, it imposes some minimum size requirement for the code of any algorithm using the S-box, which might not be negligible if an 8-bit S-box is used on constraint platforms, and it requires to access secret-dependent memory locations, which depending on the architecture might lead to the so-called “timing attacks” [TOS10].

We will briefly present two alternative ways of computing S-box applications, that may not always be applicable but that have some merits.

2.1 Parallel table lookups with vector instructions

Vector instructions are CPU instructions that operate on large registers (e.g. 128 or 256 bits) that are logically thought of as holding multiple small elements (e.g. sixteen 8-bit values). Current CPUs support many instructions that take such registers as input and that compute a single instruction with operands all the vectors elements. This paradigm is commonly known as SIMD: “single instruction, multiple data”.

An example of SIMD instructions on Intel/AMD processors s `PADDD`, which takes two 128-bit `xmm` registers (or possibly a 128-bit memory location for the second operand) logically seen as vectors `x` and `y` of four 32-bit values, and that updates the entries of `x` to hold the values `x[0] + y[0]`, `x[1] + y[1]`, etc.

A SIMD instruction that is very useful to implement many cryptographic algorithms is `PSHUFB`, which computes a parallel table-lookup of a 4-to-8-bit table on sixteen 4-bit values. Seeing its 128-bit operands x and y as vectors of sixteen bytes, the result of calling `PSHUFB` x y is to update x as follows:

$$x'[i] = \begin{cases} x[\lfloor y[i] \rfloor_4] & \text{if the most significant bit of } y[i] \text{ is not set} \\ 0 & \text{otherwise} \end{cases}, \; 0 \leq i < 16$$

where $\lfloor \cdot \rfloor_4$ denotes truncation to the four least significant bits. One can see that by taking x to be a table T of sixteen arbitrary 8-bit “images”, and by taking y to be any vector of sixteen 4-bit “indices” $(a, b, c, \ldots)$, calling `PSHUFB` x y will update x to x' as $x'[0] = x[y[0]] = T[a]$, $x'[1] = x[y[1]] = T[b]$, etc. By taking T to be the images of an S-box, a single call to `PSHUFB` performs a parallel application of this S-box on 64 bits of data. This is obviously particularly useful to implement block ciphers using 4-bit S-boxes, but the approach can be successfully extended to a larger S-box as well, if it can be efficiently expressed in terms of smaller S-boxes. This is for instance the case of the 8-bit S-boxes of subsection 1.3, but also to some extent of the one of the AES [Ham09].

Like most SIMD instructions, PSHUFB may be called in C by using compiler intrinsics, in this case _mm_shuffle_epi8. The following Figure 1 is an example implementation of an 8-bit S-box using a Lai-Massey structure, using C intrinsics.

```c
__m128i sub(__m128i x)
{
  __m128i xlo, xhi, xmid;

  __m128i LO_MASK = _mm_set1_epi8(0x0f);
  __m128i LO_SBOX = _mm_set_epi32(0x01030605, 0x0b090e0d, 0x0802070c,
 ↪  0x0f040a00);
  __m128i HI_SBOX = _mm_set_epi32(0x10306050, 0xb090e0d0, 0x802070c0,
 ↪  0xf040a000);

  xhi  = _mm_srli_epi16(x, 4);
  xhi  = _mm_and_si128(xhi, LO_MASK);
  xlo  = _mm_and_si128(x, LO_MASK);
  xmid = _mm_xor_si128(xlo, xhi);

  xmid = _mm_shuffle_epi8(LO_SBOX, xmid);
  xlo  = _mm_xor_si128(xlo, xmid);
  xhi  = _mm_xor_si128(xhi, xmid);

  xlo = _mm_shuffle_epi8(LO_SBOX, xlo);
  xhi = _mm_shuffle_epi8(HI_SBOX, xhi);
  x   = _mm_xor_si128(xlo, xhi);

  return x;
}
```

Figure 1: SSE implementation of an 8-bit S-box using compiler intrinsics.

2.2 Bitsliced implementations

S-boxes being typically small and applied many times in parallel, they are a typical target for vectorization. This can be done as in the above, by using dedicated vector instructions, but also by considering the very usual bitwise operations such as `^`, `&` and `|` in C as operating on vectors of w bits, where w is the bit-length of the operands (typically up to 64). That is, by definition, computing `a ^ b` with `a` and `b` 64-bit registers means computing 64 parallel 1-bit XORs.

This basic property can be used to easily vectorize any Boolean function, as soon as one knows how to compute it explicitly in terms of its input variables. For instance, on a 64-bit architecture, it is not costlier to compute the function $(a,b,c) \mapsto a \cdot b + c$ 64 times than only once: the C code `r = a&b ^ c` results in the bit i of `r` being equal to the above function applied to the bit i of `a`, `b` and `c`.

The biggest hurdle to using bitsliced implementations of S-boxes in practice is to represent the S-box inputs in a suitable way. For instance, if S must be applied to a string of consecutive bits in memory, each input must first be split as bits of the same position in several registers, which may be an expensive process. Consequently, bitsliced implementations are usually used either to implement the encryption of many inputs to the block cipher in parallel (in which case the input format conversion is simpler), as in the case of DES for which this technique was originally introduced in cryptography [Bih97a], or for algorithms that were explicitly designed to take advantage of such techniques, such as Serpent or SHA-3.

Unlike implementations by table lookups (parallel or not), the efficiency of a bitsliced implementation is highly dependent on the one of computing its associated Boolean functions in an explicit way. In other words, one want to find efficient logical circuits implementing the S-box. Note that as already hinted at in subsection 1.1, this can be done either by first specifying an S-box and then searching for efficient circuits (see e.g. [Osv00] in the original context of Serpent), or by generating efficient circuits and testing if they make good S-boxes [UDI+11]. In Figure 2, we give a bitsliced implementation of the S-box of Table 1 (together with the required data input and output transformation).

```
uint8_t serpent_s6(uint8_t x)
{
        uint8_t r0 = x & 1;
        uint8_t r1 = (x >> 1) & 1;
        uint8_t r2 = (x >> 2) & 1;
        uint8_t r3 = (x >> 3) & 1;
        uint8_t r4;

        r2 ^= 1;
        r4  = r3;

        r3 &= r0;
        r0 ^= r4;

        r3 ^= r2;
        r2 |= r4;

        r1 ^= r3;
        r2 ^= r0;

        r0 |= r1;
        r2 ^= r1;

        r4 ^= r0;
        r0 |= r3;

        r0 ^= r2;
        r4 ^= r3;

        r4 ^= r0;
        r3 ^= 1;

        r2 &= r4;

        r2 ^= r3;

        return (r0 | (r1 << 1) | (r4 << 2) | (r2 << 3));
}
```

Figure 2: Bitsliced implementation of the Serpent S6 S-box.

Finally, let us remark that an explicit way of computing an S-box can always be derived from its algebraic normal form. However, if not further simplified, it will usually be rather inefficient.

References

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.

[BDMW10] KA Browning, JF Dillon, MT McQuistan, and AJ Wolfe. An apn permutation in dimension six. *Finite Fields: theory and applications*, 518:33–42, 2010.

[Bih97a] Eli Biham. A fast new DES implementation in software. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings* [Bih97b], pages 260–272.

[Bih97b] Eli Biham, editor. *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*. Springer, 1997.

[BNN$^+$10] Paulo S. L. M. Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Des. Codes Cryptography*, 56(2-3):141–162, 2010.

[BR03] Paulo S.L.M. Barreto and Vincent Rijmen. The WHIRLPOOL Hashing Function, May 2003. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html.

[CDL15] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of lightweight s-boxes using feistel and MISTY structures. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 373–393. Springer, 2015.

[CR15] Anne Canteaut and Joëlle Roué. On the behaviors of affine equivalent sboxes regarding differential and linear attacks. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 45–74. Springer, 2015.

[Ham09] Mike Hamburg. Accelerating AES with vector permute instructions. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2009.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Biham [Bih97b], pages 28–40.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. Printcipher: A block cipher for ic-printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[LP07] Gregor Leander and Axel Poschmann. On the classification of 4 bit s-boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[Mat97] Mitsuru Matsui. New block encryption algorithm MISTY. In Biham [Bih97b], pages 54–68.

[Nat15] National Institute of Standards and Technology. FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015.

[Nyb93] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1993.

[Osv00] Dag Arne Osvik. Speeding up serpent. In *AES Candidate Conference*, pages 317–329, 2000.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4×4-bit s-boxes. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2011.

[TOS10] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptology*, 23(1):37–71, 2010.

[UDI$^+$11] Markus Ullrich, Christophe De Cannière, Sebastian Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of 4×4-bit S-boxes. In *SKEW 2011*, 2011.