On Reverse-Engineering S-Boxes with Hidden Design Criteria or Structure

Alex Biryukov(✉) and Léo Perrin

University of Luxembourg, SnT, Walferdange, Luxembourg
{alex.biryukov,leo.perrin}@uni.lu

Abstract. S-Boxes are the key components of many cryptographic primitives and designing them to improve resilience to attacks such as linear or differential cryptanalysis is well understood. In this paper, we investigate techniques that can be used to reverse-engineer S-box design and illustrate those by studying the S-Box F of the Skipjack block cipher whose design process so far remained secret. We first show that the linear properties of F are far from random and propose a design criteria, along with an algorithm which generates S-Boxes very similar to that of Skipjack. Then we consider more general S-box decomposition problems and propose new methods for decomposing S-Boxes built from arithmetic operations or as a Feistel Network of up to 5 rounds. Finally, we develop an S-box generating algorithm which can fix a large number of DDT entries to the values chosen by the designer. We demonstrate this algorithm by embedding images into the visual representation of S-box's DDT.

Keywords: S-box design criteria · Skipjack · Linearity · Functional decomposition problem · Efficient implementation

1 Introduction

Non-linearity in cryptographic primitives is usually provided by so-called S-Boxes, functions which map a few inputs bits to a few output bits and which are often specified as look-up tables. These have been a topic of intensive research since their properties are crucial for resilience of a cipher against differential [1–3] and linear [4,5] attacks. Further, the structure or the method used to build the S-Box can provide other benefits.

Indeed, the structure of an S-Box can be leveraged for instance to improve the implementation of a primitive using it. The hash function Whirlpool [6] and the block ciphers Khazad [7], Fantomas, Robin [8] and Zorro [9] among others use 8×8 bits S-Boxes built from smaller 4×4 ones, since storing several 4×4 permutations as tables of 16 4-bits nibbles is more memory efficient than storing one 8×8 permutation as a table of 256 bytes. Except for implementation advantage, knowledge of the internal structure helps to produce more efficient masked implementations against side-channel attacks, a notable example here being the AES [10] with its algebraic S-box based on a power function.

R. Gennaro and M. Robshaw (Eds.): CRYPTO 2015, Part I, LNCS 9215, pp. 116–140, 2015.
DOI: 10.1007/978-3-662-47989-6_6

In some cases the design process of an S-Box might be kept secret for the purpose of implementing white-box cryptography, as described e.g. in [11]. In this paper, Biryukov et al. describe a memory-hard white-box encryption scheme based on a Substitution-Permutation Network where the S-Boxes are very large and are built using a so-called ASASA or ASASASA structure where "A" denotes an affine layer and "S" a non-linear S-Box layer. Preventing an adversary from decomposing these S-Boxes into their "A" and "S" layers is at the core of the security claims for this scheme.

Moreover such memory-hard white-box implementations with hidden structure of components can be of use in crypto-currencies, for example in cases where an entity is interested in issuing a crypto-currency of its own. One of the dangers is that powerful adversaries may launch a 51 % attack taking control of the mining process. Memory hard S-Boxes with hidden structure can offer a distinct advantage in such setting since efficient implementation of the proof-of-work function may be kept secret by the owners of the currency.

Examples of algorithms for which the components are known but the rationale behind their choice is not (at least at the time of release), are the block ciphers designed by or with the help of the US National Security Agency (NSA), namely the DES [12], Skipjack [13], SIMON and SPECK [14] (the last two do not use S-Boxes though). Although the design criteria for the S-Boxes of DES were later released [15] they were kept secret for 20 years in order to hide the existence of differential cryptanalysis, a technique only known by IBM and NSA at the time. Skipjack also uses an S-Box, denoted F, which is a permutation of $\{0,1\}^8$. However, nothing was known so far about how this S-Box was chosen.

Our Contribution. Different methods can be used to recover the hidden structure of an S-Box. We propose that a cryptanalyst follows the strategy given below to try and decompose an unknown S-Box S:

1. Draw the "Pollock" visual representation of the LAT and DDT of S (see Sect. 4).
2. Check whether the linear and differential properties of S are compatible with a random function/permutation (see Sect. 2).
3. Compute the signature $\sigma(S)$ of S.
4. If $\sigma(S)$ is even, you may:
   (a) Try an attack on SASAS [16],
   (b) Try to distinguish S from a Feistel Network with XOR, using the distinguishers in [17],
   (c) If one of the Feistel Network distinguishers worked, run `DecomposeFeistel` $(S, R, \oplus)$ for an appropriate R (see Sect. 3.2).
5. Regardless of $\sigma(S)$, run `DecomposeFeistel`$(S, R, \boxplus)$ for $R \in [2, 5]$ (see Sect. 3.2).
6. Regardless of $\sigma(S)$, run `BreakArithmetic`(S) (see Sect. 3.1).

We study in Sect. 2 the seemingly average linear properties of F. After a careful investigation and despite the fact that these properties are not impressive, we show that the probability for a random permutation of $\{0,1\}^8$ to have linear

properties at least as good as those of F is negligible. This implies three things. First, F was not chosen uniformly at random. Second, F is very unlikely to have been picked among random candidates according to some criteria. Third, the method used to build it improved the linear properties. We also provide a candidate algorithm which can be used to generate S-Boxes with very similar differential and linear properties.

In Sect. 3 we consider a general problem of decomposition of an S-box with hidden structure and describe two algorithms which can be used to decompose S-Boxes based on: (a) multiple iterations of simple arithmetic operations (for ex. like those found in a typical microprocessor) and (b) Feistel Networks with up to five independent rounds. The first algorithm is an optimised tree-search and the second one involves a SAT-solver.

Finally, we show in Sect. 4 how visual representations of the difference distribution table (DDT) or the linear approximation table (LAT) of an S-Box can help a cryptographer to spot non-randomness at a glance. As a bonus, we present an algorithm which generates non-bijective S-Boxes such that large set of entries in their DDT are set according to the designer's choices. We illustrate it by embedding images in the visual representation of the S-Box's DDT.

2 Partially Reverse-Engineering the S-Box of Skipjack

2.1 Overview of the S-Box of Skipjack and Useful Definitions

Skipjack is a block cipher with a block size of 64 bits and key size of 80 bits. The interested reader may refer to the official specification [13] or to the best attack on the cipher [18], an impossible differential attack leveraging its particular round structure. Further analysis trying to discover the design criteria of Skipjack is given in [19,20].

Skipjack's specification contains and 8×8 bit bijective S-box which is called "F-Table" and which is given as a lookup table (we list it in the Appendix A). In order to study it we need to introduce the following concepts.

Definition 1 (Permutations Set). *We denote $\mathfrak{S}_{2^n}$ the set of all the permutations of $\{0,1\}^n$.*

Definition 2 (Difference Distribution Table). *Let $s : \{0,1\}^n \to \{0,1\}^n$ be a function. Its* difference distribution table (DDT) *is a $2^n \times 2^n$ matrix where the number at line i and column j is*

$$d_{i,j} = \#\{x \in \{0,1\}^n | s(x \oplus i) \oplus s(x) = j\}.$$

The maximum coefficient in this table (minus the first line and column) is the differential uniformity *of s which we denote $\Delta(s)$: $\Delta(s) = \max_{i>0,j>0}(d_{i,j})$.*

Differential cryptanalysis relies on finding differential transitions with high probabilities, i.e. pairs (a, b) such that $s(x \oplus a) \oplus s(x) = b$ has many solutions which is equivalent to $d_{a,b}$ being high. Therefore, cryptographers usually attempt

to use S-Boxes s with as low a value of $\Delta(s)$) as possible. A function differentially 2-uniform, the best possible, is called *Almost Perfect Nonlinear* (APN). The existence of APN permutations of $GF(2^n)$ for even n was only proved recently by Browning[1] et al. [21] in the case $n = 6$, while the case $n = 8$ and beyond still remains an open problem. Hence, the differential uniformity of the S-Boxes of the AES [10] and of most modern S-Box based ciphers is equal to 4.

The distribution of the coefficients in the DDT of Skipjack is summarized in Table 1 along with the theoretical distribution identified in [22] for a random permutation of $GF(2^8)$. As we can see it is differentially 12-uniform, the same as you would expect from a random permutation, which is surprising since minimizing the differential uniformity is usually one of the corner stones of provable resilience against differential attacks.

Table 1. Distribution of the coefficients in the DDT of F.

Coefficient	Number	Proportion (%) in F	Poisson(1/2) (%)
0	39104	60.14	60.65
2	20559	31.62	30.33
4	4855	7.467	7.582
6	686	1.055	1.264
8	69	0.106	0.158
10	5	0.008	0.016
12	2	0.003	0.002

We briefly mention the linear properties of F before studying them thoroughly in Sect. 2.2. In particular, we define the Linear Approximations Table of an S-Box.

Definition 3 (Linear Approximations Table). *Let $s : \{0,1\}^n \to \{0,1\}^n$ be a function. Its* linear approximations table (LAT) *is a $2^n \times 2^n$ matrix where the number at line i and column j is*

$$c_{i,j} = \#\{x \in \{0,1\}^n | x \cdot i = s(x) \cdot j\} - 2^{n-1} = \frac{1}{2} \sum_{x \in \{0,1\}^m} (-1)^{i \cdot x \oplus j \cdot s(x)}$$

with "$\cdot$" denoting the scalar product. The maximum absolute value of the $c_{i,j}$ is the linearity of s, *$\Lambda(s)$, where $\Lambda(s) = \max_{i>0,j>0}(|c_{i,j}|)$.*

The quantity $c_{i,j}$ has different names in the literature. It is called "*bias*" or "*Imbalance*" of the Boolean function $x \mapsto i \cdot x \oplus j \cdot s(x)$ in, for example, [22]. In papers from the Boolean functions community, it is more often defined in terms of *Walsh Spectrum*, the Walsh Spectrum of a Boolean function being the

[1] The fact that Browning works at the NSA shows that this agency values theoretical considerations, which makes the simplicity of F all the stranger.

multiset $\{c_{i,j}/2\}_{i\geq 0, j\geq 0}$. The maximum coefficient in the LAT of F is $\Lambda(F) = 28$ and it occurs in absolute value 3 times.

For the sake of completeness, we also give the sizes of the cycles in which F can be decomposed: 2, 10, 45, 68, 131.

2.2 The Linear Properties are Too Good to be True

Figure 1 contains the distribution of the value of the coefficients of the LAT (minus the first line and column) along with the theoretical proportions for a random permutation of $GF(2^8)$ described below.

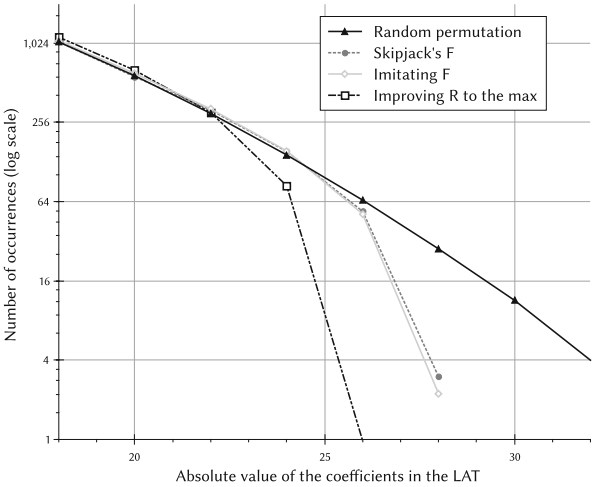


Fig. 1. Coefficients of the LAT of F, random permutations and some outputs of `Improve-`$R(s)$.

The probability distribution for the coefficients $c_{i,j}$ in the LAT of a permutation of $\mathfrak{S}_{2^n}$ is described in [23]:

$$P[c_{i,j} = 2z] = \frac{\binom{2^{n-1}}{2^{n-2}+z}^2}{\binom{2^n}{2^{n-1}}}.$$

Using Sect. 3.4 of [22], we derived that $\Lambda(s)$ has a mean over all permutations $s \in \mathfrak{S}_{2^8}$ of approximately 34.8 which is notably larger than for F since $\Lambda(F) = 28$.

Given the probability distribution of the coefficients of the LAT, it is easy to compute the probability that $\Lambda(f) \leq 28$ assuming that f is a permutation chosen uniformly at random and that the coefficients' values correspond to independent sample of the same distribution. Note that there are only $(2^8 - 1)^2$ such trials because the first line and column are ignored here.

$$P[\Lambda(f) \leq 28] = \Big(\sum_{j=-14}^{14} P[c_{i,j} = 2j] \Big)^{(2^8-1)^2} \approx 2^{-25.62}.$$

This probability is low but it would be feasible to generate a set of about 2^{26} random permutations from $\mathfrak{S}_{2^8}$ and compute the LAT for each of them. In such a set, the best S-Box s should verify $\Lambda(s) = 28$. However, we must also take into account that in order to resist linear cryptanalysis it is not only best to have a low maximum value, it is also better to have a low number of occurrences of it. In this regard, F and its only three occurrences of 28 could almost be considered as having a maximum value of 26 for which $P[\Lambda(f) = 26] = 2^{-66.4}$.

More rigorously, we compute the probability to have at most q coefficients equal to 28 in the LAT of a permutation picked uniformly at random from $\mathfrak{S}_{2^8}$. If we let $p(2i) = P[c_{i,j} = 2i]$, then this probability is equal to $P_{28,q}$ where

$$P_{28,q} = \sum_{j=0}^{q} \left[\binom{(2^8-1)^2}{j} \big(p(28) + p(-28)\big)^j \Big(\sum_{k=-13}^{13} p(2k) \Big)^{(2^8-1)^2-j} \right].$$

Unsurprisingly, we find that this probability is equal to $2^{-66.4}$ for $q = 0$, i.e. the probability to have $\Lambda(s) \leq 26$. It also converges to $2^{-25.6} = P[\Lambda(s) \leq 28]$ when q increases. For $q = 3$, the case of Skipjack's F, we find:

$$P_{28,3} = 2^{-54.4}.$$

The probability for a random permutation to have linear properties comparable to those of Skipjack's F is thus at most $2^{-54.4}$. Hence, we claim:

- F was not chosen uniformly at random in $\mathfrak{S}_{2^8}$,
- the designers of Skipjack did not generate many random permutation to then pick the best according to some criteria as they would need to have generated at least about 2^{55} S-Boxes,
- the method used to build F improved its linear properties.

2.3 A Possible Design Criteria

We tried to create an algorithm capable of generating S-Boxes with linear and differential properties similar to those of F. It turns out that such an algorithm is easy to write. First, we introduce a quantity we denote $R(f)$ and define as follows:

$$R(f) = \sum_{\ell \geq 0} N_\ell \cdot 2^\ell,$$

where N_ℓ counts coefficients with absolute value ℓ in the LAT of f: $N_\ell = \#\{c_{i,j} \in (\text{LAT of } f), |c_{i,j}| = \ell\}$.

Algorithm 1 starts from a random permutation s of $\mathfrak{S}_{2^8}$ and returns a new permutation s' such that $R(s') < R(s)$ and such that s' is identical to s except for two entries x and y which are swapped: $s'(x) = s(y)$ and $s'(y) = s(x)$. It works by identifying one of the highest coefficient in the LAT, removing it through swapping two entries and checking whether $R(s)$ was actually improved. This algorithm can be used in two different ways: either we keep iterating it until it

Algorithm 1. `Improve-`$R(s)$

```
c := LAT of s
Find a, b such that |c_{a,b}| = Λ(s)
L := empty list
for all x ∈ {0,1}^8 do
    if a · x = b · f(x) then
        Append x to L
    end if
end for
for all (x, y) ∈ L^2, x ≠ y do
    s' = s ; s'(x) = s(y) ; s'(y) = s(x)
    if R(s') < R(s) then
        return s'
    end if
end for
return Fail
```

reaches a point at which no swap can improve $R(s)$ or we stop as soon as $R(s)$ is below an arbitrary threshold.

We implemented both variants. For the second one, we stop when $R(s) < 10^{10}$ because $R(F) \approx 10^{9.92}$. We denote N_ℓ the average number of coefficient with absolute value ℓ in the LAT or the DDT of the S-Boxes obtained. For the LAT, $\log_2(N_\ell)$ is given in Table 2 and in Fig. 1; for the DDT it is in Table 3. "Random" corresponds to the average over 200 S-Boxes picked uniformly at random in $\mathfrak{S}_{2^8}$; "F" to the distribution for the S-Box of Skipjack; "F-like" to the average over 100 S-Boxes obtained using `Improve-`$R()$ and stopping when $R(s) < 10^{10}$; "best" to the average over 100 S-Boxes obtained using `Improve-`$R()$ and stopping only when it fails.

Table 2. Distribution of $\log_2(N_\ell)$ in the LAT of different S-Boxes.

ℓ	Random	F	F-like	best $R()$
20	9.164	9.147	9.230	9.311
22	8.220	8.308	8.336	8.247
24	7.173	7.267	7.280	6.400
26	6.041	5.755	5.688	0.000
28	4.826	1.585	1.157	-
30	3.506	-	-	-
32	2.146	-	-	-
34	0.664	-	-	-

Using `Improve-`$R()$ with an appropriate threshold allows us to create S-Boxes with both linear and differential properties very close to F. However, in order to achieve this, we need to choose a threshold value computed from F

Table 3. Distribution of $\log_2(N_\ell)$ in the DDT of different S-Boxes.

ℓ	Random	F	F-like	Best $R()$
0	15.265	15.246	15.250	15.227
2	14.270	14.327	14.314	14.380
4	12.277	12.245	12.257	12.210
6	9.693	9.422	9.492	9.126
8	6.701	6.109	6.198	5.265
10	3.374	2.322	2.287	0.714
12	−0.059	1.000	−1.786	−5.059
14	−4.059	-	−5.059	-

and which does not correspond to anything specific. In fact, to the best of our knowledge, the quantity $R(s)$ does not have any particular importance unlike for instance the linearity $\Lambda(s)$. Still, replacing $R(s)$ by the linearity $\Lambda(s)$ or a pair $(\Lambda(s), \#\{(i,j), c_{i,j} = \Lambda(s)\})$ yields S-Boxes which are very different from F. Such S-Boxes indeed have a value of $N_{\Lambda(s)-2}$ much higher than in the random case, which is not the case for F.

While our definition of $R(s)$ may seem arbitrary, it is the only one we could find that leads to linear properties similar to those of F. For instance it may have been tempting to base $R(s)$ on the square of ℓ which is used when computing the correlation potential of a linear trail, a quantity useful when looking for linear attacks. We would thus define $R(s) = \sum_{\ell \geq 0} N_\ell \ell^2$. However this quantity is worthless as an optimization criteria since it is constant: Parseval's equality on the Walsh spectrum of a Boolean function imposes that the sum of the $(c_{i,j})^2$ over each column is equal to 2^{2n-2}.

To conclude: we have found new non-random properties of the S-box of Skipjack which are improving its strength against linear cryptanalysis and we developed and algorithm which could be used to generate such S-boxes.

2.4 Public Information About the Design of Skipjack

The only information indirectly published by the NSA on Skipjack corresponds to an "Interim Report" [24] written by external cryptographers and it contains no information on the specifics of the design. The most relevant parts of this report as far as the S-Box is concerned are the following ones.

> SKIPJACK was designed to be evaluatable [...]. In summary, SKIPJACK is based on some of NSA's best technology. Considerable care went into its design and evaluation in accordance with the care given to algorithms that protect classified data.

Furthermore, after the "leakage" of an alleged version of Skipjack to usenet[2], Schneier replied with a detailed analysis of the cipher [26] which contained in particular the following quote indicating that the S-box was changed in August 1992.

> The only other thing I found [through documents released under FOIA] was a SECRET memo. [...] The date is 25 August 1992. [...] [P]aragraph 1 reads:
> 1. (U) The enclosed Informal Technical Report revises the F-table in SKIPJACK
> 2. No other aspect of the algorithm is changed.

Note also that the first linear cryptanalysis of DES [4] had not been published yet in August 1992 when the F-Table was changed. Gilbert et al. suggested at CRYPTO'90 [27] to use linear equation to help with key guessing in differential attack to attack FEAL. This block cipher was later attacked at CRYPTO'91 [28] and EUROCRYPT'92 [29] using directly some linear equations involving plaintext, ciphertext and key bits. We can but speculate about a connection between these papers and the change of S-Box of Skipjack.

3 Algorithm Decomposing Particular Structures

A powerful tool able to discard quickly some possible structures for an S-Box is its *signature*, as shown in Lemma 1.

Definition 4 (Permutation Signature). *A permutation s of $\{0,1\}^n$ has an odd signature if and only if it can be decomposed into an odd number of transpositions, a transposition being a function permuting two elements of $\{0,1\}^n$. Otherwise, its signature is even.*

The signature of $f \circ g$ is even if and only if f and g have the same signature.

Lemma 1. *The following $b \times b$ permutations always have an even signature:*

- *Feistel Networks using XOR to combine the output of the Feistel function with the other branch,*
- *Substitution-Permutation Networks for which the diffusion layer is linear in $GF(2)^b$ or can be decomposed into a sequence of permutations ignoring a fraction of the internal state.*

Proof. Let b be the block size of the block ciphers considered. The proof for the case of Feistel Networks with XOR can be found in [30].

[2] An anonymous member of `sci.crypt` posted what they claimed to be Skipjack at a time when this algorithm was still classified [25]. Although the algorithm described, "S-1", turned out to be different from Skipjack as we know it, it used similar notations — the S-Box is called "F-Table" — and the key-schedule leads to identical round keys being used every 5 rounds, just like in the actual Skipjack.

Let us look at substitution permutation networks. An S-Box layer consists in the parallel application of several invertible S-Boxes operating on n bits, with n dividing b. This operation can be seen as the successive application of the S-Box on each n bit block, one after another. Such an operation ignores 2^{b-n} bits, meaning that its cycle decomposition consists in 2^{b-n} replicas of the same set of cycles. Since 2^{b-n} is even, the application of each S-Box is even; which in turn implies that the successive application of the S-Box on each block is even. More generally, any permutation which can be decomposed into a sequence of sub-permutations ignoring a fraction of the internal state is even. The fact that permutations linear in $GF(2)^b$ are even is showed in the proof of Lemma 2 in [31]. □

The restriction put on the diffusion layer of SPN's is usually not important, e.g. the diffusion layer of the AES fits the requirement. However, for small block sizes, it must be taken into account.

So far, we have proved that F has been built in contrast to being picked out of a set of random S-Boxes according to some criteria. The signature of F is odd so Lemma 1 implies that F cannot be a Feistel Network with XOR. The generic attack on the SASAS structure [16] fails on F, meaning that it is not a simple SPN either. Finally, F is not affine equivalent to a monomial of $GF(2^n)$ like for instance the S-Box of the AES. Indeed, such functions have the same coefficients in the lines of their DDT, only the order is different. This observation lead to the definition of the differential spectrum by Blondeau et al. [32]. It also implies that, for a monomial, the number of coefficients equal to d in its DDT must divide $2^n - 1$. As it is not the case for F, we can also rule out this structure.

However, this is not sufficient to conclude that F does not have a particular structure. It could be based on simple operations such as rotations, addition modulo 2^n and multiplication available in a typical microprocessor (thus offering the designer a benefit of memory-efficient implementation) or on a Feistel Network which uses modular addition to combine the output of the Feistel function with the other branch. We study these two possibilities in this section by first describing an algorithm capable of decomposing S-Boxes built from multiple simple arithmetic operations and then by presenting a new attack recovering all Feistel functions of a small Feistel Network of up to 5-rounds regardless of whether XOR or modular addition is used.

The purpose of the algorithms we present in this section can be linked to the more general *Functional Decomposition Problem (FDP)* tackled notably over two rounds in [33]. In this paper, Faugère et al. introduce a general algorithm capable of decomposing $h = (h_1, ..., h_u)$ into $\big(f_1(g_1, ..., g_n), ..., f_u(g_1, ..., g_n)\big)$ where the h_i's, f_i's and g_i's are polynomials of n variables. The time complexity of this algorithm (see Theorem 3 of [33]) is lower bounded by $\mathrm{O}\big(n^{3\cdot(d_f d_g - 1)}\big)$ where d_f (respectively d_g) is the maximum algebraic degree of the f_i's (respectively the g_i's). Note that this lower bound on the time complexity is not tight. In fact, the ratio n/u of the number of input variables over the number of coordinates of h is also of importance, the lower being the better.

3.1 Iterated Simple Arithmetic Permutation

A plausible assumptions for an efficient yet compact S-box design is that the S-box is constructed using a formula containing basic instructions available in the microprocessor. Indeed, a simple code:

```
for (i = 0; i < 3; i++) {
    y = a * (ROTL8((b * y) ^ c, d)) ^ e;
}
```

generates an S-box which may have a differential uniformity better than Skipjack's F's for a proper choice of constants a, b, c, d and e.

We introduce `BreakArithmetic`(s), an optimized tree-search capable of recovering the simple operations used to create such an S-Box constructed as an arbitrary sequence of basic processor instructions. It is based on the following observation. Suppose that $s = \phi_r \circ ... \circ \phi_1$, where the ϕ_i's are one of the following algebraic operations: constant XOR, constant addition modulo 2^n, multiplication by a constant modulo 2^n and bit rotation by a constant. Then $s \circ \phi_1^{-1} = \big(\phi_r \circ ... \circ \phi_1\big) \circ \phi_1^{-1} = \phi_r \circ ... \circ \phi_2$, meaning that $s \circ \phi_1^{-1}$ is "less complex", "closer from the identity" than s itself. The aim of this algorithm is to peel of the ϕ_i's one after another by performing a tree-search among all possible simple operations which selects operations to consider first based on how closer they get us to the identity.

In order for this to work, we need to capture the concept of "distance to the identity" using an actual metric which can be implemented efficiently. We chose to base this metric on the DDT since it is less expensive to compute than the LAT[3]. We define the following metric: $M(s) = \sum_{\ell \geq 2} N_\ell (\ell - 2)^2$. Our tree-search privileges candidates ϕ_1 such that $M(s \circ \phi_1^{-1})$ is closer from $M(\mathrm{Id})$, where Id is the identity function.

Our implementation of this algorithm is for example capable of recovering the decomposition of $s : x \mapsto \psi\big(\psi\big(\psi(x)\big)\big)$ with $\psi : x \mapsto 0xa7 \cdot \big((3 \cdot x \oplus 0x53) >>> 4\big) \oplus 0x8b$. However, our algorithm could not find any such decomposition for Skipjack's F despite running for 96 hours on a CUDA computer with more than 1000 cores for fast computation of the DDT.

3.2 Decomposing Feistel Structures

Another possible structure for F which is compatible with its having an odd signature is a Feistel Network where the XOR is replaced by a modular addition. In this section, we describe an algorithm which uses a SAT-solver to recover the Feistel functions of small Feistel Networks which use either XOR or modular addition. We describe below the key idea of this attack, namely the encoding of the truth table of each Feistel function using Boolean variables and then how we can use this encoding to actually decompose a small Feistel Network.

[3] One can also notice that linear operations do not alter the DDT profile of the permutation and thus one has to recompute the metric only after non-linear operations.

Methods to distinguish Feistel Networks from random permutations have been actively investigated, notably in the work by Luby and Rackoff [34] as well as by Patarin [35,36]. Here, we present a method which goes beyond distinguishing: it actually recovers all the Feistel functions for up to 5-rounds of Feistel Networks with low branch width.

Encoding of the Feistel Function. Let $f : \{0,1\}^n \to \{0,1\}^n$ be an unknown function. We associate to each of its output bits i on each possible input x a unique variable z_i^x. The truth-table of f is thus as shown in Table 4 for $n = 3$. We encode the fact that a vector of Boolean variables $y_i, i \in [0, n-1]$ is the output of f given input variables $x_i, i \in [0, n-1]$ using the truth-table of f by building a CNF[4] involving $\{x_i\}_{i<n}, \{y_i\}_{i<n}$ and $\{z_i^x\}_{i<n,\ x<2^n}$ which is true if and only if $(y_{n-1}, ..., y_0) = f(x_{n-1}, ..., x_0)$.

Table 4. The variables used to encode an unknown function $f : \{0,1\}^3 \to \{0,1\}^3$, where $(y_2, y_1, y_0) = f(x_2, x_1, x_0)$.

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	z_2^0	z_1^0	z_0^0
0	0	1	z_2^1	z_1^1	z_0^1
...	...	...	...	...	...
1	1	1	z_2^7	z_1^7	z_0^7

We denote $\mathrm{bit}_i(b)$ the i-th of the binary expansion of any integer $b < 2^n$ in little-endian notation so that $b = \sum_{i<n} \mathrm{bit}_i(n) 2^{n-i}$. We also denote a^1 the variable a itself and a^0 its negation. The procedure used to build this CNF is based on the following implication: if $\{x_i\}_{i<n}$ corresponds to the binary expansion of an integer $x < 2^n$ and $\{y_i\}_{i<n}$ to the binary expansion of the integer $y = f(x)$, then $y_i \oplus z_i^x = 0$ for all $i < n$. Using the notations we just introduced, this idea can be written as n implications, the conjunction of which for $j < n$ must hold:

$$\Big(\bigwedge_{i<n} x_i^{\mathrm{bit}_i(x)}\Big) \implies \big(y_j \oplus z_j^x = 0\big).$$

Each of these can be turned into a CNF made of two clauses using that $(a \implies b) \equiv (a^0 \vee b^1)$, that $(a \oplus b = 0) \equiv \big((a^1 \vee b^0) \wedge (a^0 \vee b^1)\big)$ and basic linear algebra as follows:

$$\Big(\big(\bigvee_{i<n} x_i^{1-\mathrm{bit}_i(x)}\big) \ \vee \ y_j^1 \vee z_j^0\Big) \wedge \Big(\big(\bigvee_{i<n} x_i^{1-\mathrm{bit}_i(x)}\big) \ \vee \ y_j^0 \vee z_j^1\Big).$$

If we concatenate the CNF generated in this way for all values of $x < 2^n$, we obtain a CNF which we denote "CNF$\big(f, \{x_i\}, \{y_i\}\big)$" with $2n2^n$ clauses involving $n2^n + 2n$ variables. It holds if and only if the assignment of the variables $\{x_i\}_{i<n}$ and $\{y_i\}_{i<n}$ is such that $(y_{n-1}, ..., y_0) = f(x_{n-1}, ..., x_0)$.

[4] A formula in *Conjunctive Normal Form* is the conjunction of multiple *clauses*, each of them being the disjunction of some possibly negated variables.

Generating the Full CNF and Solving. Using $\text{CNF}\big(f, \{x_i\}, \{y_i\}\big)$, a SAT-solver and the full codebook of a S-Box $S : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$, we can recover the Feistel functions used to generate S if it was indeed generated using a Feistel network or prove that it was not constructed in this fashion using `DecomposeFeistel`$(S, R, \text{operation})$ (see Algorithm 2). To describe it, we introduce variables $\{x_i^r\}_{i<2n}, \{y_i^r\}_{i<n}$ and, if the combining function is a modular addition instead of a XOR, $\{c_i^r\}_{i<n}$ for $r < R$ where R is the number of rounds we consider were used. These are summarized in Fig. 2.

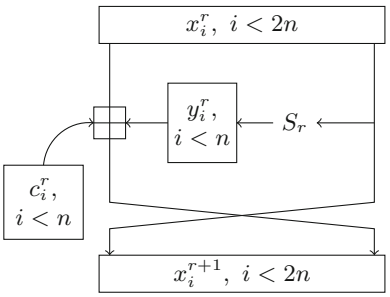


Fig. 2. The variables used to encode round r of a Feistel Network operating on blocks of $2n$ bits.

The general idea consists in building the CNF representation of the fact that $S(p) = c$ for each input/output pair (p, c) separately, concatenate these CNF's and then have a SAT-solver solve the CNF obtained in this fashion. To each Feistel functions is associated a unique set of $n2^n$ variables as described in the previous section. These are used when encoding that half of the internal state at round $r + 1$ of the Feistel Network goes through the corresponding Feistel function. The only difficulty left is the combination of the left branch with the output of the Feistel function. In the case where a XOR is used, we can simply encode that $x_i^{r+1} = y_i^r \oplus x_{i+n}^r$ separately for each bit i. However, in the case of a modular addition, we need to introduce a new set of variables for each evaluation of the addition corresponding to the carry bits: $\{c_i^r\}_{i<n}$. The addition is then encoded into a CNF using the CNF encoding of the following equations:

$$x_{i+1}^r = c_i^r \oplus x_{i+n}^r \oplus y_i^r,$$
$$c_{i+1}^r = \big(c_i^r \wedge x_{i+n}^r\big) \vee \big(y_i^r \wedge x_{i+n}^r\big) \vee \big(c_i^r \wedge y_i^r\big).$$

A useful heuristic when trying to decompose more than 4 rounds is to look for decompositions with particular patterns in the sequence of the Feistel functions. For instance, decomposing a 5-rounds Feistel Network with round functions $(S_a, S_b, S_c, S_d, S_a)$ is easier than decomposing a similar structure with round functions $(S_a, S_b, S_c, S_d, S_e)$ if this knowledge is hard-coded in the CNF by using the same sets of variables to encode both S_e and S_a. In this case,

Algorithm 2. `DecomposeFeistel`$(S, R, \text{operation})$

$C :=$ empty CNF
for all $p \in [0, 2^n - 1]$ **do**
  for all $r \in [0, R-1]$ **do**
    $\{x_{i+n}^{p,r+1}\}_{i<n} = \{x_i^{p,r}\}_{i<n}$
    $C :=$ Concatenation of CNF$\left(S_r, \{x_i^{p,r}\}_{i<n}, \{y_i^{p,r}\}\right)$ and C
    if operation is $\oplus$ **then**
      Append CNF repr. of $\{x_i^{p,r+1}\}_{i<n} = \{y_i^{p,r}\}_{i<n} \oplus \{x_{i+n}^{p,r}\}_{i<n}$ to C
    else
      Append CNF repr. of $\{x_i^{p,r+1}\}_{i<n} = \{y_i^{p,r}\}_{i<n} \boxplus \{x_{i+n}^{p,r}\}_{i<n}$ to C
    end if
  end for
  for all $i \in [0, n-1]$ **do**
    Append clause only made of literal $(x_i^{p,0})^{\text{bit}_i(p)}$ to C
    Append clause only made of literal $(x_i^{p,R})^{\text{bit}_i(S(p))}$ to C
  end for
end for
Run SAT-solver on C
if C is satisfiable **then**
  Extract truth-table of all S_r's from the variable assignment
  return "Feistel Network with R rounds"
else
  return "**Not** a Feistel Network with R rounds"
end if

`DecomposeFeistel`$(S, R, \text{operation})$ also takes the assumed sequence of the S-Boxes as an additional input.

Another improvement comes from the observation that constants can be XOR-ed (or added/subtracted) in the input of Feistel functions in the first $R-2$ rounds — provided they are cancelled by XOR-ing (or adding/substracting) in the later rounds — without changing the output of the function. Using this, we can arbitrarily decide that the first Feistel functions all map, say, 0 to 0. This simplification of the CNF helps the SAT-solver a lot and is actually necessary to attack 5 independent rounds.

We implemented Algorithm 2 and used the SAT-solver Minisat [37] to solve the CNF formula generated. The time taken to decompose S-Boxes actually made of small Feistel Networks is smaller than the time taken to discard an S-Box which is not based on such a structure. Decomposing 8×8 S-Boxes built using 4-rounds Feistel Networks, regardless of whether $\oplus$ or $\boxplus$ is used, takes less than a second on a regular desktop PC[5] and discarding S-Boxes built in other ways requires about 5 seconds. Decomposing 5-rounds requires a bit less than a minute but discarding this structure takes longer, for instance 3 min to prove that F is not a 5-rounds $\oplus$-Feistel and 23 min to show that is it not a 5-rounds $\boxplus$-Feistel. It is also possible to attack larger instances provided enough RAM is

[5] The PC used for the experiments has a Intel(R) Core(TM) i7-3770 CPU (3.40GHz) for a cpu and 8 Go of RAM.

available. A 4-rounds Feistel Network corresponding to a 14×14 S-Box can be broken in about 2 hours using up to about 38 Go of RAM[6].

The CNF formulas equivalent to F being a Feistel Network with 3,4 or 5 rounds, using either $\oplus$ or $\boxplus$ are all unsatisfiable, meaning that F is not a Feistel Network with at most 5 rounds.

For the sake of completeness, we mention the existence of another time efficient attack on 5-round Feistel Networks by Gaëtan Leurent based on a boomerang-like property [38]. Indeed one of the open problems is how far cryptanalytic techniques can go in analysis of ciphers with small block, where the full code-book is available to the attacker.

4 From an S-Box to a Picture and Back Again

In order to distinguish an S-Box from a random one we propose a new method which we call *Pollock's Pattern Recognition*[7]. It is based on turning the DDT and the LAT of the S-Box into a picture and then use the natural pattern finding power of the human eye to identify not-random properties. We also describe a method to perform (partially) the inverse operation: *Seurat's Steganography*[8]. It creates an S-Box such that an image is embeded in the picture representation of its DDT.

4.1 Pollock's Pattern Recognition

As is clear from Sect. 2, the distribution of the coefficients in the LAT of an S-Box provides a powerful tool to distinguish a random-looking S-Box from a permutation chosen uniformly at random from the set of all permutations. We suggest here another method for looking at these coefficients which can also be applied to the DDT. The idea is to look at the whole table at once, be it a DDT or LAT, and then rely on the pattern matching capabilities of the pair human eye/human brain to possibly discard that the S-Box was chosen uniformly at random. In order to look at the whole table, we associate to the values of the coefficients different colors. Exactly which color scale to use is a question which can only be answered by trying different ones. As an illustration of the power of this method, we provide pictures allowing us to discard the randomness of 4 S-Boxes using merely a quick glance in Appendix B.

Zorro. The S-Box of this cipher [9] is based on a 4-rounds Feistel Network with a complex diffusion layer. As a consequence, the algorithm presented in

[6] This experiment was performed on a single core of a dedicated server with 500 Go of RAM.

[7] The pictures obtained in this fashion have a strong abstract feel to them, hence a name refering to the painter Jackson Pollock for this algorithm.

[8] As will be explained later, this algorithm works by drawing the image to embed point after point just like in a *pointillist* painting, hence the name of the painter who invented this method.

Sect. 3.2 fails on it. The picture representation of its LAT, given in Fig. 4a, contains "stripes". These correspond to coefficients equal to 6 (orange) and 2 (green). These never appear for half of the input masks according to a repeating pattern. Such a behaviour is not expected from a random permutation. The color scheme was chosen so as to highlight this property. We note that the congruence modulo 2^k for some k of the coefficients of the LAT is related to the algebraic degree of $i \cdot x \oplus j \cdot S(x)$ as explained for example in [39] (Proposition 6.1).

CLEFIA. This block cipher [40] uses two distinct S-Boxes. The one denoted S_0 has a particular structure based on smaller 4×4 S-Boxes. The LAT of this S-Box is given in Fig. 4b: note the "dents" on the top and left side of the picture as well as the low number of colors compared to Fig. 4c which also depicts a LAT and uses the same color-scale. This low number of colors is a consequence of the fact that no coefficient in the LAT is congruent to 2 modulo 4 which in turn is related to this S-Box having an algebraic degree equal to 6 on all of its coordinates. Neither this nor the "dents" are expected from a random permutation.

SAFER+. This block cipher [41] uses an S-Box based on exponentiation in $\mathbb{Z}/256\mathbb{Z}$. Its LAT is given in Fig. 4c; note in particular the vertical lines whih appear in this representation.

Arithmetic. The DDT can also be used in the same fashion. For example, we can look at the DDT of an S-Box generated using a simple algebraic expression similar to those discussed in Sect. 3.1, namely $s : x \mapsto \psi\big(\psi(x)\big)$ with $\psi : x \mapsto 3 \cdot \big((3 \cdot x \oplus 0x53) >>> 4\big) \oplus 0x8b$. The representation of its DDT is in Fig. 4d. Note the white rectangles corresponding to subsets of impossible differentials and the loose similarity between the top left and bottom right quadrants on one hand and the top right and bottom left quadrants on the other hand. None of these characteristics are expected from the DDT of a random permutation. Note that with 3 iterations of ϕ this S-box becomes reasonably good.

We however were not able to spot any particular pattern in the Pollock representation of neither the DDT nor the LAT of Skipjack's F. Such representations are given respectively in Figs. 3a and b in Appendix B. We used the function `matrix_plot` from the SAGE [42] software package to draw the Pollock representations.

4.2 Seurat's Steganography

In this section, we present an algorithm allowing the creation of a non-bijective S-Box such that the picture representation of its DDT contains a particular image. Since we draw this image dot after dot like in *pointillism* and since it hides said image, we call the method we present below *Seurat's Steganography*. The pictures we embed are black and white, the white parts corresponding to places where differentials are impossible and black parts to places where the differentials have non-zero probability.

The Algorithm. We define white and black equations as those giving the corresponding pixel color in the Pollock representation of the DDT of an S-Box.

White Equations. $W_{a,b} : \forall x \in \{0,1\}^m,\ S(x+a) + S(x) \neq b.$

Black Equations. $B_{a,b} : \exists x \in \{0,1\}^m,\ S(x+a) + S(x) = b.$

The inputs considered in this Section are:

B The complete list of the black equations.
T_w A table of booleans of size $u \times v$ (the dimensions of the image) where $T_w[a,b]$ is false if and only if the pixel at (a,b) cannot be white.
S A partially unspecified S-Box such that all equations B_j for $j < i$ hold and such that none of the W_j has a solution for any j.
i The index of the equation in B for which we need to find a solution.

We first need a sub-routine checking if adding an entry $S(x) = y$ to a partially assigned S-Box, i.e. an S-Box for which some of the outputs are unspecified, leads to at least one of the white equations not holding anymore. It is described in Algorithm 3.

Algorithm 3. checkW(S, x, y, T_w).

```
for all a ∈ {0,1}^m, if S(x+a) is specified, do
    if T_w[a, S(x+a)+S(x)] is false then return false
end for
return true
```

We now describe Seurat's Steganoraphy, namely Algorithm 4, which uses two lists of equations to iteratively build an S-Box such that a particular picture appears in its DDT. It works by first making a list L of all the ways entries could be added to the S-Box in order to satisfy the black equation B_i. If none are found, the function fails. The function is finally called recursively on the candidates found to look for a solution for the next equation. If no solution are found for the next equation, the function fails.

Some optimizations are possible. First of all, it is not necessary to write this algorithm using recursion. It is also not necessary to let L be as large as possible. In fact $|L| \leq 2$ is sufficient, although $|L| = 1$ does not work unless the picture is very simple. It is also possible to allow some noise by tweaking CheckW(S, x, y, T_w) to return `true` with low probability for pairs (x, y) even if they blacken a white pixel.

Two outputs of this algorithm are presented in Appendix C: the S-Boxes are given along with the Pollock representation of their DDT which clearly show the pictures we chose to embed in them. The differential and linear properties of the S-Box described in Table 6 are close from what would be expected from a random function (differential uniformity of 14, linearity of 39), meaning that it could be used in a context were a 8×8 random function would be sufficient.

Algorithm 4. Seurat(S, B, T_w, i).

δ_{in} := input difference in B_i
δ_{out} := output difference in B_i
L := empty list of S-Boxes
if B_i is already satisfied by S **then**
  Append S to L and **return** L
end if
for all $x \in \{0,1\}^m$ **do**
  if $S(x)$ is not defined but $S(x + \delta_{\text{in}})$ is defined **then**
    $y = S(x + \delta_{\text{in}}) + \delta_{\text{out}}$
    if CheckW(S, x, y, T_w) **then**
      $S' = S$; $S'(x) = y$
      Append S' to L
    end if
  else if $S(x + \delta_{\text{in}})$ is not defined either **then**
    for all $y \in \{0,1\}^n$ **do**
      if CheckW(S, x, y, T_w) and CheckW$(S, x + \delta_{\text{in}}, y + \delta_{\text{out}}, T_w)$ **then**
        $S' = S$; $S'(x) = y$; $S'(x + \delta_{\text{in}}) = y + \delta_{\text{out}}$
        Append S' to L
      end if
    end for
  end if
end for
If L is still empty **then return** `Fail`
for all $S' \in L$ **do**
  If Seurat$(S', B, T_w, i + 1)$ does not fail **then return** S'
end for
return `Fail`

Counting Possible S-Boxes. Let S be a random function from $\{0,1\}^m$ to $\{0,1\}^n$. Then $W_{a,b}$ holds if and only if $d_{a,b} = 0$, which happens with probability $P[d_{a,b} = 0] = \exp\left(-2^{m-n-1}\right)$ because the coefficients in the DDT of a random function follow approximately a Poisson distribution with parameter $1/2$ (see [22]). Hence, if we have b black equations, w white ones and if we consider that their having solutions are independent events, then the probability that an S-Box has the correct image at the center of its DDT is $P_{\text{success}} = \left(\exp(-2^{m-n-1})\right)^w \times \left(1 - \exp(-2^{m-n-1})\right)^b$. In the case where $m = n$, we use that $\log_2(\exp(-1/2)) \approx -1.35$ and that $\log_2(1 - \exp(-1/2)) \approx -0.72$ to approximate this probability by

$$P_{\text{success}} = 2^{-(0.72 \cdot w + 1.35 \cdot b)}.$$

As there are 2^{n2^n} possible $n \times n$ S-Boxes, we expect to have very roughly the following amount of solutions:

$$N_{\text{Solutions}} = 2^{n2^n - (0.72 \cdot w + 1.35 \cdot b)}.$$

Therefore, we need $0.72 \cdot w + 1.35 \cdot b < n2^n$ in order to have a non-empty set of S-Box with the image we want inside their DDT. Black pixels are about twice as expensive as white ones according to this model. However, in practice, it is only possible to build a S-Box such that its DDT contains a black square of size 22×22 or a white one of size 62×62 without any noise, meaning that black pixels are, from our algorithm's point of view, about 8 times more expensive. Stirling's equation gives an approximate number of $2^{(n-1.44)\cdot 2^n}$ permutations of $\{0,1\}^n$, so we need that $0.72 \cdot w + 1.35 \cdot b < (n - 1.44)2^n$ in order for permutations with the correct black/white pixels to exist with non negligible probability. However, our algorithm will require significant changes in order to search for permutations.

Since our algorithm does not require the pixels to be organised inside a square, we can also use it to force white or black pixels to appear anywhere in the DDT of an S-Box. This could be used to place a sort of trapdoor by for instance ensuring that a truncated differential compatible with the general structure of a cipher is present. Another possible use could be to "sign" a S-Box: Alice would agree with Bob to generate a S-Box for him and tell him before hand where some black/white pixels will be. Bob can then check that those are placed as agreed.

5 Conclusion

Knowledge of the internal structure of an S-box gives clear advantages to the designer of a cipher in terms of efficient or side-channel resistant implementation. It is also crucial in the white-box or crypto-currency setting. Hiding the S-box's structure can be also a way to hide superior cryptanalysis techniques or trapdoors.

In this paper we have introduced several approaches and algorithms to decompose an S-Box with unknown structure and we illustrated them by studying the S-Box of the NSA's block cipher Skipjack. This allowed us to rule out some possible structure, and to prove that its linear properties are too unlikely to have happened at random. We also provided an algorithm capable of generating very similar S-Boxes (Table 5).

An open problem related to this work is the study of block ciphers with small block sizes: how far can cryptanalysis go given a whole codebook? How many rounds of small-block Feistel Network or SPN is it feasible to break?

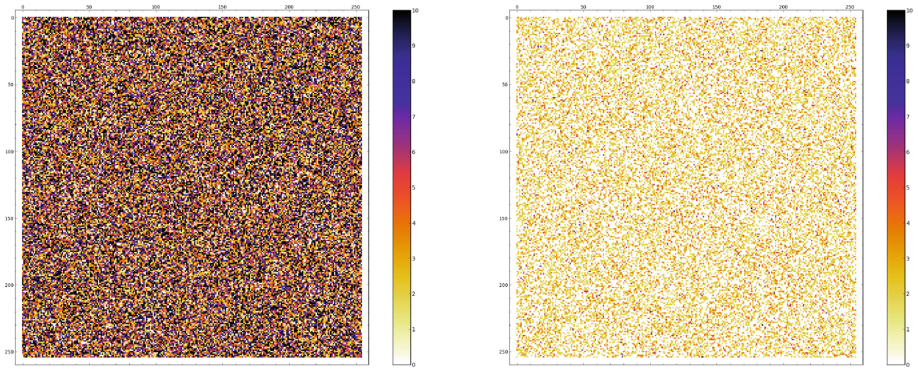
Acknowledgement. We thank the CRYPTO reviewers for their helpful comments. We also thank Anne Canteaut for pointing out the connection between algebraic degree and congruence of the coefficient of the LAT modulo 2^k. The work of Léo Perrin is supported by the CORE ACRYPT project (ID C12-15-4009992) funded by the *Fonds National de la Recherche* (Luxembourg).

A The S-Box of Skipjack

Table 5. Skipjack's S-Box, F, in hexadecimal notation. For example, $F(7a) = d6$.

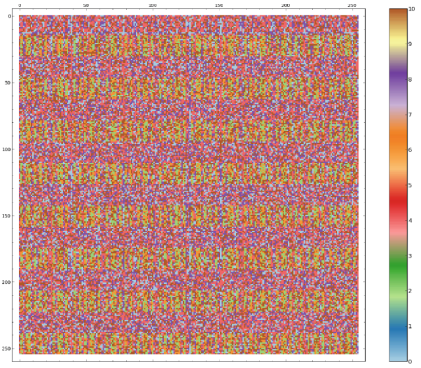
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	a3	d7	09	83	f8	48	f6	f4	b3	21	15	78	99	b1	af	f9
1.	e7	2d	4d	8a	ce	4c	ca	2e	52	95	d9	1e	4e	38	44	28
2.	0a	df	02	a0	17	f1	60	68	12	b7	7a	c3	e9	fa	3d	53
3.	96	84	6b	ba	f2	63	9a	19	7c	ae	e5	f5	f7	16	6a	a2
4.	39	b6	7b	0f	c1	93	81	1b	ee	b4	1a	ea	d0	91	2f	b8
5.	55	b9	da	85	3f	41	bf	e0	5a	58	80	5f	66	0b	d8	90
6.	35	d5	c0	a7	33	06	65	69	45	00	94	56	6d	98	9b	76
7.	97	fc	b2	c2	b0	fe	db	20	e1	eb	d6	e4	dd	47	4a	1d
8.	42	ed	9e	6e	49	3c	cd	43	27	d2	07	d4	de	c7	67	18
9.	89	cb	30	1f	8d	c6	8f	aa	c8	74	dc	c9	5d	5c	31	a4
a.	70	88	61	2c	9f	0d	2b	87	50	82	54	64	26	7d	03	40
b.	34	4b	1c	73	d1	c4	fd	3b	cc	fb	7f	ab	e6	3e	5b	a5
c.	ad	04	23	9c	14	51	22	f0	29	79	71	7e	ff	8c	0e	e2
d.	0c	ef	bc	72	75	6f	37	a1	ec	d3	8e	62	8b	86	10	e8
e.	08	77	11	be	92	4f	24	c5	32	36	9d	cf	f3	a6	bb	ac
f.	5e	6c	a9	13	57	25	b5	e3	bd	a8	3a	01	05	59	2a	46

B Picture Representation of the DDT and LAT of Some S-Boxes

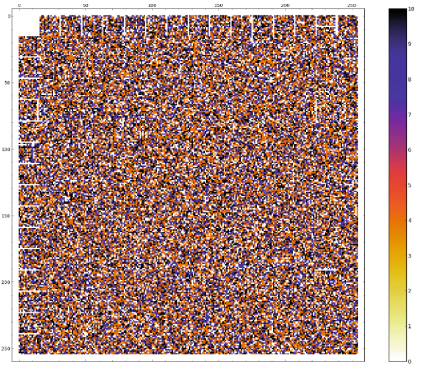


(a) The LAT of Skipjack's F (b) The DDT of Skipjack's F

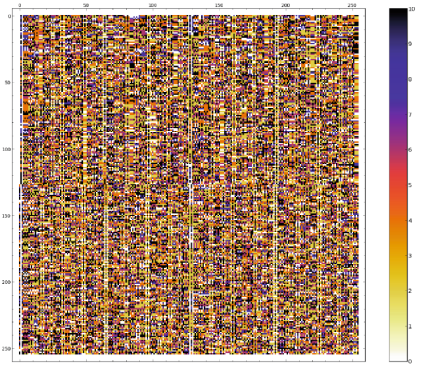
Fig. 3. The Pollock representations of the LAT and DDT of Skipjack's F. For both, 0 is in white and anything equal to or above 10 is black.



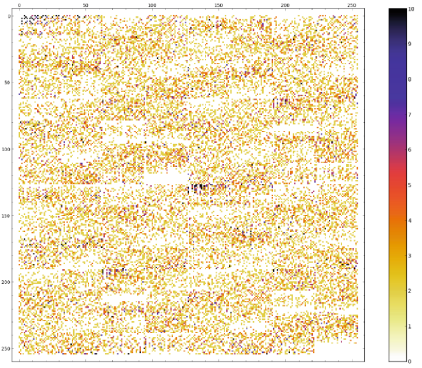
(a) The LAT of the S-Box of Zorro



(b) The LAT of the S-Box S_0 of CLEFIA

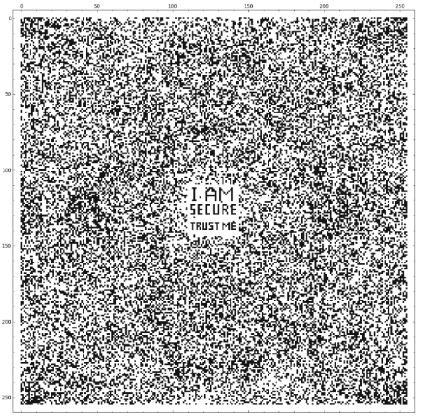


(c) The LAT of the S-Box of SAFER+

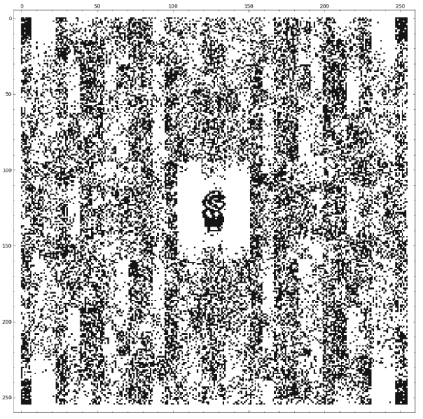


(d) The DDT of a simple S-Box

Fig. 4. The Pollock representation of the LAT or DDT of different S-Boxes. The scales all go from 0 to 10 (anything above 10 is treated as equal to 10).



(a) The DDT of the S-Box in Table 6



(b) The DDT of the S-Box in Table 7

Fig. 5. The DDT of some outputs of Seurat's Steganography: $|d_{i,j}| = 0$ is in white, $|d_{i,j}| \geq 2$ is in black.

C S-Boxes Built from Pictures

The S-Box described in Tables 6 and 7 were built using the method described in Sect. 4.2. Note that these are not bijections. The picture representations of their DDT are given in Figs. 5a and b.

Table 6. An output of the algorithm described in Sect. 4.2.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	7a	b3	b9	b6	53	b1	26	6e	b9	43	86	ec	94	4b	9e	43
1.	5d	83	d5	57	16	4c	44	d5	5d	81	7f	79	b3	8d	e6	f8
2.	0d	59	b3	8d	04	4c	8d	ec	d9	ff	7f	7a	7e	9a	92	61
3.	05	fc	e3	1a	ed	12	1e	52	1a	e5	30	34	ef	e5	97	e5
4.	9e	69	29	d6	29	cd	b8	3a	d2	c4	1b	d1	1c	17	c3	3b
5.	44	ba	bd	19	57	0c	5a	5f	bb	55	b7	4a	5e	3f	a6	fe
6.	7f	c8	7e	65	be	1e	b3	bf	8b	85	83	83	87	12	b2	26
7.	a6	b4	bc	ef	9e	9d	6c	9e	90	5e	68	25	30	97	9f	71
8.	bf	64	65	9a	77	18	da	60	05	97	58	b2	88	d5	25	a1
9.	58	00	db	85	ca	9f	8d	42	db	bc	b2	b6	e7	85	44	78
a.	ac	be	5b	21	45	e9	40	4d	73	5f	af	93	4b	bd	45	42
b.	55	37	e2	c8	c8	20	d1	ee	7e	36	c5	28	32	37	2f	d4
c.	86	21	79	70	08	b6	91	89	e3	e5	10	e5	c6	cf	02	ca
d.	cc	b9	e1	9a	8c	8c	f3	70	ec	13	0f	00	17	7e	57	5c
e.	09	27	27	85	a0	87	3f	53	74	e3	b1	bd	de	b1	8d	61
f.	4b	84	9c	f3	72	04	7e	9c	25	3e	98	9e	43	8d	b2	9d

Table 7. An output of the algorithm described in Sect. 4.2.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	1b	1e	e7	1b	00	1b	4f	e7	07	a8	b7	1c	00	06	1c	1c
1.	30	a9	ab	af	54	50	36	57	65	01	17	7c	53	99	fb	65
2.	86	b5	33	78	c9	80	f5	7f	79	7d	87	7a	4d	14	49	2b
3.	66	d5	c8	54	a9	57	54	ab	aa	98	a8	a8	32	17	d2	cb
4.	d4	e7	73	1b	51	b3	af	50	51	68	ac	6b	d7	52	1b	d5
5.	71	75	8a	97	c8	36	37	33	74	ce	75	4a	77	88	8f	77
6.	1b	ff	e4	b5	ff	1f	1e	fa	b3	4a	b1	4c	fd	fc	4b	01
7.	ca	c8	a0	5b	5e	a1	5b	a6	9d	c8	98	84	cb	31	ca	cb
8.	33	ca	33	cc	7b	83	98	cb	a2	7f	a3	ce	34	33	cb	cd
9.	e7	fd	ff	03	7f	2d	00	b5	05	e5	ff	02	03	06	fc	06
a.	88	8e	74	8b	8c	8e	8c	51	c9	03	88	c9	8a	c9	70	fc
b.	94	2b	d4	29	ae	69	6b	af	b7	91	b7	b7	8b	89	d4	75
c.	d1	c9	98	99	61	ab	aa	61	99	66	12	65	15	2d	2d	33
d.	b3	b3	7c	86	83	7a	7f	78	cf	98	81	30	7e	cf	c9	c9
e.	01	a9	57	ad	e3	80	ad	61	56	53	53	28	56	a8	c8	ae
f.	18	1d	00	06	df	52	52	af	1d	61	e2	60	e2	e6	fa	e2

References

1. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptology **4**(1), 3–72 (1991)
2. Nyberg, K.: Differentially uniform mappings for cryptography. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 55–64. Springer, Heidelberg (1994)
3. Blondeau, C., Nyberg, K.: Perfect nonlinear functions and cryptography. Finite Fields Appl. **32**, 120–147 (2015). Special Issue: Second Decade of FFA
4. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
5. Nyberg, K.: Linear approximation of block ciphers. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 439–444. Springer, Heidelberg (1995)
6. Barreto, P., Rijmen, V.: The whirlpool hashing function. In: First open NESSIE Workshop, Leuven, Belgium, vol. 13, p. 14 (2000)
7. Barreto, P., Rijmen, V.: The khazad legacy-level block cipher. Primitive submitted to NESSIE 97 (2000)
8. Grosso, V., Leurent, G., Standaert, F.-X., Varıcı, K.: LS-Designs: bitslice encryption for efficient masked software implementations. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 18–37. Springer, Heidelberg (2015)
9. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.-X.: Block ciphers that are easier to mask: how far can we go? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer, Heidelberg (2013)
10. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-the Advanced Encryption Standard. Springer, Heidelberg (2002)

11. Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic schemes based on the ASASA structure: black-box, white-box, and public-key (extended abstract). In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 63–84. Springer, Heidelberg (2014)
12. U.S. DEPARTMENT: OF COMMERCE/National Institute of Standards and Technology: Data encryption standard. Publication, Federal Information Processing Standards (1999)
13. National Security Agency, N.S.A.: SKIPJACK and KEA Algorithm Specifications (1998)
14. 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, 404 (2013)
15. Coppersmith, D.: The Data Encryption Standard (DES) and its strength against attacks. IBM J. Res. Dev. **38**(3), 243–250 (1994)
16. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 394–405. Springer, Heidelberg (2001)
17. Patarin, J.: Luby-Rackoff: 7 rounds are enough for formula $2^{n(1-\epsilon)}$ security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 513–529. Springer, Heidelberg (2003)
18. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. J. Cryptology **18**(4), 291–311 (2005)
19. Knudsen, L.R., Robshaw, M., Wagner, D.: Truncated differentials and skipjack. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 165–180. Springer, Heidelberg (1999)
20. Knudsen, L., Wagner, D.: On the structure of skipjack. Discrete Appl. Math. **111**(1), 103–116 (2001)
21. Browning, K., Dillon, J., McQuistan, M., Wolfe, A.: An apn permutation in dimension six. Finite Fields: Theory Appl. **518**, 33–42 (2010)
22. Daemen, J., Rijmen, V.: Probability distributions of correlation and differentials in block ciphers. J. Math. Cryptology JMC **1**(3), 221–242 (2007)
23. O'Connor, L.: Properties of linear approximation tables. In: Preneel, B. (ed.) FSE 1995. LNCS, vol. 1008, pp. 131–136. Springer, Heidelberg (1995)
24. Brickell, E.F., Denning, D.E., Kent, S.T., Maher, D.P., Tuchman, W.: Skipjack review: Interim report (1993)
25. Anonymous: This looks like it might be interesting. sci.crypt (usenet), August 1995. https://groups.google.com/forum/#!msg/sci.crypt/vLtuBDoqPfc/jm6MshFbomgJ
26. Schneier, B.: The S-1 Algorithm. mail to the cypherpunk mailing list (1995). http://cypherpunks.venona.com/date/1995/09/msg00315.html
27. Gilbert, H., Chassé, G.: A statistical attack of the FEAL-8 cryptosystem. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 22–33. Springer, Heidelberg (1991)
28. Tardy-Corfdir, A., Gilbert, H.: A known plaintext attack of FEAL-4 and FEAL-6. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 172–182. Springer, Heidelberg (1992)
29. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 81–91. Springer, Heidelberg (1993)
30. Patarin, J.: Generic attacks on feistel schemes. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 222–238. Springer, Heidelberg (2001)

31. Wernsdorf, R.: The round functions of RIJNDAEL generate the alternating group. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 143–148. Springer, Heidelberg (2002)
32. Blondeau, C., Canteaut, A., Charpin, P.: Differential properties of power functions. Int. J. Inf. Coding Theory **1**(2), 149–170 (2010)
33. Jean-Charles, F., Perret, L.: An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography. J. Symbolic Comput. **44**(12), 1676–1689 (2009)
34. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. **17**(2), 373–386 (1988)
35. Patarin, J.: New results on pseudorandom permutation generators based on the DES scheme. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 301–312. Springer, Heidelberg (1992)
36. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)
37. Een, N., Sörensson, N.: Minisat: A sat solver with conflict-clause minimization. Sat 5 (2005)
38. Biryukov, A., Leurent, G., Perrin, L.: ESC 2015 S-box Reverse-Engineering Challenge. In: Early Symmetric Crypto, ESC 2015, pp. 104–107 (2015)
39. Canteaut, A.: Analyse et Conception de Chiffrements à Clef Secrète. Habilitation à diriger des recherches, Institut National de Recherche en Informatique et Automatique, Rocquencourt, September 2006
40. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit blockcipher CLEFIA (extended abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
41. Massey, J.L.: Safer k-64: A byte-oriented block-ciphering algorithm. In: Anderson, R. (ed.) FSE 1993. LNCS, vol. 809, pp. 1–17. Springer, Heidelberg (1994)
42. Stein, W., et al.: Sage Mathematics Software (Version 5.10). The Sage Development Team (2013). http://www.sagemath.org