Anomalies and Vector Space Search: Tools for S-Box Analysis (Full Version)*

Xavier Bonnetain[1,2], Léo Perrin[1] and Shizhu Tian[1,3,4]

[1] Inria, France
[2] Sorbonne Université, Collège doctoral
[3] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
xavier.bonnetain@inria.fr, leo.perrin@inria.fr, tianshizhu@iie.ac.cn

Abstract. S-boxes are functions with an input so small that the simplest way to specify them is their lookup table (LUT). How can we quantify the distance between the behavior of a given S-box and that of an S-box picked uniformly at random?
To answer this question, we introduce various "anomalies". These real numbers are such that a property with an anomaly equal to a should be found roughly once in a set of 2^a random S-boxes. First, we present statistical anomalies based on the distribution of the coefficients in the difference distribution table, linear approximation table, and for the first time, the boomerang connectivity table.
We then count the number of S-boxes that have block-cipher like structures to estimate the anomaly associated to those. In order to recover these structures, we show that the most general tool for decomposing S-boxes is an algorithm efficiently listing all the vector spaces of a given dimension contained in a given set, and we present such an algorithm.
Finally, we propose general methods to formally quantify the complexity of *any* S-box. It relies on the production of the smallest program evaluating it and on combinatorial arguments.
Combining these approaches, we conclude that all permutations that are *actually* picked uniformly at random always have essentially the same cryptographic properties and the same lack of structure. These conclusions show that multiple claims made by the designers of the latest Russian standards are factually incorrect.

Keywords: S-Box · Reverse-engineering · Vector space search · BCT · Anomaly · Boolean functions · Shannon effect.

1 Introduction

S-boxes are small functions with an input small enough that they can be specified by their lookup tables. If F is an S-box with an n-bit input then it is feasible to describe it using only the sequence $F(0), F(1), ..., F(2^n - 1)$ since, in the vast majority of the cases, $3 \leq n \leq 8$. S-boxes can therefore correspond to arbitrarily complex functions. In practice, such components are the only source of non-linearity of many symmetric primitives. Most prominently, the AES [AES01] uses an 8-bit bijective S-box.

However, because they can be specified using only their lookup tables, it is not necessary for algorithm designers to disclose their design process. They can build an S-box using a secret structure and then hide this structure by only disclosing the lookup table. Such

*This is the full version of the Asiacrypt'19 paper [BPT19a].

an action is considered bad practice as it cannot foster the trust necessary to use the algorithm so specified. And yet, there are several instances of *standardized* algorithms that use secretly designed S-boxes: the DES [DES77], Skipjack [U.S98], and the pair consisting of the hash function Streebog [Fed12] and the block cipher Kuznyechik [Fed15]. The DES and Skipjack were American standards while Streebog and Kuznyechik are Russian ones. Streebog is part of the standard ISO/IEC 10118-3 while Kuznyechik is being considered for inclusion in ISO/IEC 18033-3.

The generation method used by the designers of the S-box shared by Streebog and Kuznyechik then had to be recovered by external cryptographers who, after several attempts [BPU16, PU16], succeeded in [Per19]. The structure presented in this last paper is extremely rare: the probability that a random permutation has a similar one is under 2^{-1601}. Yet, in an internal memo sent to ISO [SM18] before the publication of [Per19], the designers of Kuznyechik stated the following.

> Through thorough search current S-box was obtained [...]
>
> No secret structure was enforced during construction of the S-box. At the same time, it is obvious that for any transformation a lot of representations are possible (see, for example, a lot of AES S-box representations).

In this paper, we prove that none of these statements are correct:[1] it would be necessary to generate an infeasibly large set of S-boxes to obtain one with similar differential, linear, and boomerang properties. At the same time, the structure found had to be inserted deliberately by its designers because the presence of *any* structure this simple in a random permutation is extremely unlikely.

So far, S-box reverse-engineering has dealt with two broad questions. Let $\mathfrak{S}_{2^n}$ be the set of all n-bit permutations and let $F \in \mathfrak{S}_{2^n}$.

1. What is the probability that an S-box picked uniformly in $\mathfrak{S}_{2^n}$ has differential/linear properties at least as good as those of F?

2. How can we recover the structure of F—if it has any?

Answering the first question can also help us better understand the properties of random permutations and thus to better estimate the advantage of an adversary trying to distinguish a (round-reduced) block cipher from a random permutation.

On the other hand, the second one is related to so-called *white-box cryptography*, i.e. to implementation techniques that will hide a secret from an attacker with a total access to the implementation of the algorithm. In practice, in order to try and hide for instance an AES key, the attacker will only be given access to an implementation relying on big lookup tables that hide the details of the computations. Recovering the original structure of these tables can be seen as a particular case of S-box reverse-engineering.

Overall, this second question is more subtle than it may seem: a given function can have multiple different decompositions as evidenced by the multiple results on the Russian S-box [BPU16, PU16, Per19]. We then ask a third natural question whose answer will allow us to both disprove a claim of [SM18], and to estimate if a random permutation can be hoped to be efficiently implemented.

3. Can we expect a random permutation to have a simple description?

1.1 Our Contributions

A Key Concept: Anomalies. We answer the two questions asked above using different variants of a unique approach based on what we call *anomalies*. Intuitively, an anomaly is

[1] With one exception: there are indeed many representations of the AES S-box.

a real number that quantifies how unlikely a property is. For example, there are very few differentially-6 uniform 8-bit permutations,[2] meaning that the anomaly of this property should be high. However, we could argue that what matters in this case is not just the number of differentially-6 uniform permutations but the number of permutations with a differential uniformity *at most* equal to 6. In light of this, we define anomalies as follows.

Definition 1 (Anomaly)**.** Let $F \in \mathfrak{S}_{2^n}$ and let P be a function mapping $\mathfrak{S}_{2^n}$ to a partially ordered set. The *anomaly of* $P(F)$ is defined as $\mathsf{A}(P(F)) = -\log_2(\Pr[P(G) \leq P(F)])$, where the probability is taken over $G \in \mathfrak{S}_{2^n}$. We can equivalently write

$$\mathsf{A}(P(F)) \;=\; -\log_2\left(\frac{\left|\left\{G \in \mathfrak{S}_{2^n}, P(G) \leq P(F)\right\}\right|}{|\mathfrak{S}_{2^n}|}\right) .$$

The *negative anomaly of* $P(F)$ is $\overline{\mathsf{A}}(P(F)) = -\log_2(\Pr[P(G) \geq P(F)])$.

Regardless of P, we always have $2^{-\mathsf{A}(P(F))} + 2^{-\overline{\mathsf{A}}(P(F))} = 1 + \Pr[P(G) = P(F)]$.

In the example given above, P is simply the function returning the differential uniformity of a permutation. The anomaly of the differential uniformity then gets higher as the differential uniformity of F decreases under the median differential uniformity as there are fewer permutations with a low differential uniformity. At the same time, the *negative anomaly* of the differential uniformity increases as the differential uniformity increases above its median value. To put it differently, the anomaly of $P(F)$ quantifies how many S-boxes are at least as good[3] as F in terms of P, and the negative one how many are at least as bad as F. In this paper, we study different anomalies and design new tools that allow their estimation for any S-box.

A property with a high anomaly can be seen as distinguisher in the usual sense, i.e. it is a property that differentiates the object studied from one picked uniformly at random. However, unlike usual distinguishers, we do not care about the amount of data needed to estimate the probabilities corresponding to the anomalies.

Statistical Anomalies. In [BP15] and [Per19], the notions of "differential" and "linear" anomalies were introduced. Definition 1 is indeed a generalization of them. They are based on properties P that correspond to how good the differential and linear properties are. In Section 2, we generalize this analysis to take into account the corresponding negative anomalies, and we introduce the use of the so-called *Boomerang Connectivity Table (BCT)* [CHP+18] for this purpose. To this end, we establish the distribution of the coefficients of the BCT of a random permutation. As an added bonus, this new result allows a better estimation of the advantage of an adversary in a boomerang attack.

Structural Anomalies. Anomalies can also be related to the presence of a structure. For example, for n-bit Boolean functions, the existence of a simple circuit evaluating a function is unlikely:

> "almost all functions" of n arguments have "an almost identical" complexity which is asymptotically equal to the complexity of the most complex function of n arguments.

This statement of Lupanov [Lup73] summarizes the so-called *Shannon effect* [Sha49]. In other words, the existence of a short description is an unlikely event for a Boolean function. Here, we generalize this observation to permutations of $\mathbb{F}_2^n$ and construct anomalies that capture how "structured" an S-box is.

[2] We formally define differential uniformity later. All that is needed in this discussion is that the differential uniformity is an integer which is better when lower.

[3] In this paper, the properties P considered are better when lower.

In Section 3, we present an estimation of the number of permutations that can be constructed using common S-box generation methods (multiplicative inverse, Feistel networks...) and derive the corresponding anomalies. In order to identify these anomalies, it is necessary to recover said structures when they are unknown. We present a simple approach applicable to inversion-based S-boxes that we successfully apply to the 8-bit S-box of the leaked German cipher Chiasmus. In other cases, we show that the detection of structures with a high anomaly can be performed using a vector space search.

Vector Space Search. We provide an efficient algorithm performing this search: given a set $\mathcal{S}$ of elements of $\{0,1\}^n$ and an integer d, this algorithm returns all the vector spaces of dimension d that are fully contained in $\mathcal{S}$. We present it in Section 4. While such an algorithm is needed when looking for a structure in an S-box, we expect it to find applications beyond this area.

Kolmogorov Anomalies. The anomalies we present in Section 3 are related to specific structures that are very common, but they correspond to functions P with a binary output: an S-box has the specific structure considered or it does not. It thus fails to capture the idea behind anomalies which consists in looking at the probability that an event *or a "better" version of it* occurs. To solve this problem, we take inspiration from both a proof of Shannon [Sha49] and the Kolmogorov complexity to define an anomaly that quantifies how simple an implementation of a function is that can be applied regardless of the specifics of the structure considered.

Application. We apply the different methods we present to the S-box of the Russian algorithms, π. We show that its statistical, structural, and Kolmogorov anomalies are either high or extremely high; thus disproving the claims of its designers.

1.2 Mathematical Background

Boolean Functions. Let $\mathbb{F}_2 = \{0,1\}$. In what follows, we consider the following subsets of the set of all functions mapping $\mathbb{F}_2^n$ to itself.

- Recall that the set of all n-bit permutations is denoted $\mathfrak{S}_{2^n}$. It contains $2^n!$ elements. The compositional inverse of $F \in \mathfrak{S}_{2^n}$ is denoted F^{-1}.
- The set of all n-bit linear permutations is denoted $\mathcal{L}_{2^n}$. Its size is such that $|\mathcal{L}_{2^n}| = \prod_{i=0}^{n-1}(2^n - 2^i)$.

For elements of $\mathbb{F}_2^n$, "$+$" denotes the characteristic-2 addition, i.e. the XOR. In cases that might be ambiguous, we use "$\oplus$" to denote this operation.

Let $F \in \mathfrak{S}_{2^n}$ be an S-box. Many of its cryptographic properties can be described using $2^n \times 2^n$ tables: the LAT, DDT and BCT. They are defined below.

The *Linear Approximation Table (LAT)* of F is the table $\mathcal{W}_F$ with coefficients $\mathcal{W}_F(a,b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x + b \cdot F(x)}$ where $x \cdot y = \bigoplus_{i=0}^{n-1} x_i \times y_i$ is the scalar product of two elements $x = (x_0, ..., x_{n-1}), y = (y_0, ..., y_{n-1}) \in \mathbb{F}_2^n$. Its maximum for $b \neq 0$ is the *linearity* of F and is denoted $\ell(F)$. The LAT is used to study linear cryptanalysis [TG92, Mat94]. The set of the coordinates of the coefficients equal to 0 plays a special role, as shown in [CP19]. It is called the *Walsh zeroes* of F and is denoted $\mathcal{Z}_F = \{(a,b) \in (\mathbb{F}_2^n)^2 \mid \mathcal{W}_F(a,b) = 0\} \cup \{(0,0)\}$.

The *Difference Distribution Table (DDT)* of F is the table δ_F with coefficients $\delta_F(a,b) = \#\{x \in \mathbb{F}_2^n, F(x+a) + F(x) = b\}$. Its maximum for $a \neq 0$ is the *differential uniformity* of F and is denoted $u(F)$. The DDT is needed to study differential cryptanalysis [BS91].

Recently, Cid et al. introduced a new tool which they called *Boomerang Connectivity Table (BCT)* [CHP+18]. It is again a $2^n \times 2^n$ table $\mathcal{B}_F$ defined by

$$\mathcal{B}_F(a,b) \;=\; \#\left\{x \in \mathbb{F}_2^n, F^{-1}\left(F(x)+b\right)+F^{-1}\left(F(x+a)+b\right)=a\right\} \;.$$

Its maximum value for $a, b \neq 0$ is the *boomerang uniformity* of F and is denoted β_F. As hinted by its name, the BCT is relevant when studying boomerang attacks [Wag99]. Unlike the DDT and LAT, it is necessary that F is a permutation for the BCT to be well defined.

Statistics. Some of our results rely on both the binomial and Poisson distribution. We denote with Binomial(n,p) the binomial distribution with parameters p and n which correspond respectively to the probability of an event and to the number of trial. It is defined as follows:

$$\Pr\left[X=i\right] \;=\; \text{Binomial}(i;n,p) \;=\; p^i(1-p)^{n-i}\binom{n}{i} \;.$$

It has a mean equal to np and a variance of $np(1-p)$. The Poisson distribution with parameter λ is defined by

$$\Pr\left[X=i\right] \;=\; \text{Poisson}(i;\lambda) \;=\; \frac{e^{-\lambda}\lambda^i}{i!} \;.$$

The mean value and variance of this distribution are both λ. A binomial distribution with small p can be closely approximated by a Poisson distribution with $\lambda = np$.

2 Statistical Properties

Let us consider a permutation F that is picked uniformly at random from $\mathfrak{S}_{2^n}$ and let us consider one of its tables, i.e. its DDT, LAT or BCT. The coefficients in this table may be connected to one another: for example the sum of the coefficients in a row of the DDT have to sum to 2^n. Yet, in practice, the coefficients act like independent and identically distributed random variables. In Section 2.1), we recall what the distributions of the DDT and LAT coefficients are and we establish the distribution of the BCT coefficients.

Then, Section 2.2 presents how the knowledge of these distributions can be used to bound the probability that a random permutation has differential/linear/boomerang properties at least as good as those of the S-box investigated. Additionally, we explain in Section 2.3 how our newly gained knowledge of the distribution of the BCT coefficients allows a better estimation of the advantage of the attacker in a boomerang attack.

2.1 Coefficient Distributions

In [DR07], the authors established and experimentally verified the distribution followed by the DDT and LAT coefficients. The distribution of the LAT coefficients was first established in [O'C95] and then provided a different expression in [DR07]. A more thorough study of the DDT coefficient can be found in [O'C94]. We recall these results in the following two theorems.

Proposition 1 (DDT coefficient distribution [DR07])**.** *The coefficients in the DDT of a random S-Box of* $\mathfrak{S}_{2^n}$ *with* $n \geq 5$ *are independent and identically distributed random variables following a Poisson distribution* Poisson(2^{-1}).

Proposition 2 (LAT coefficient distribution [O'C95, DR07])**.** *The coefficients in the LAT of a random permutation*[4] *of* $\mathfrak{S}_{2^n}$ *are independent and identically distributed random variables with the following probability distribution:*

$$\Pr\left[\mathcal{W}_F(i,j) = 4z\right] = \frac{\binom{2^{n-1}}{2^{n-2}+z}^2}{\binom{2^n}{2^{n-1}}} \ .$$

The situation is the same for the BCT. In order to establish the distribution of the non-trivial coefficients of the BCT of a random permutation, we first recall an alternative definition of the BCT that was introduced in [LQSL19].

Proposition 3 (Alternative BCT definition [LQSL19])**.** *Let* $F \in \mathfrak{S}_{2^n}$ *be a permutation. For any* $a, b \in \mathbb{F}_2^n$*, the entry* $\mathcal{B}_F(a,b)$ *of the BCT of* F *is given by the number of solutions in* $\mathbb{F}_2^n \times \mathbb{F}_2^n$ *of the following system of equations*

$$\begin{cases} F^{-1}(x+b) + F^{-1}(y+b) = a \\ F^{-1}(x) + F^{-1}(y) = a \ . \end{cases} \tag{1}$$

We use this theorem to obtain the distribution of the coefficients in the BCT.

Theorem 1 (BCT coefficient distribution)**.** *If* F *is picked uniformly at random in* $\mathfrak{S}_{2^n}$*, then its coefficients with* $a, b \neq 0$ *can be modeled like independent and identically distributed random variables with the following distribution:*

$$\Pr\left[\mathcal{B}_F(a,b) = c\right] \ = \sum_{2i_1+4i_2=c} P_1(i_1)P_2(i_2) \ ,$$

where P_1 *and* P_2 *are stochastic variable following binomial distributions:*

$$P_1(i) = \text{Binomial}\left(i; 2^{n-1}, \frac{1}{2^n-1}\right) \text{ and } P_2(i) = \text{Binomial}\left(i; 2^{2n-2}-2^{n-1}, \frac{1}{(2^n-1)^2}\right) \ .$$

Proof. For any $x, y \in \mathbb{F}_2^n$ such that $x \neq y$, we define

$$S_{x,y} = \{(x,y), (y,x), (x+b, y+b), (y+b, x+b)\} \ ,$$

which is of cardinality 4 unless $x + y = b$, in which case it only contains 2 elements. These sets are such that a pair (x, y) is a solution of System (1) if and only if all the elements in $S_{x,y}$ are as well. In order to prove this theorem, we will partition the set of all pairs of elements of $\mathbb{F}_2^n$ into such sets $S_{x,y}$.

To this end, we consider the following equivalence relation: $(x, y) \sim (x', y')$ if and only if the multisets $S_{x,y}$ and $S_{x',y'}$ are identical. The corresponding equivalence classes are of size 4 except when $x+y = b$, in which case they contain only 2 elements. There are in total 2^{n-1} classes of size 2. As there are $2^n(2^n-1)$ ordered pairs of elements in $\mathbb{F}_2^n$, we deduce that there are $\left(2^n(2^n-1) - 2 \times 2^{n-1}\right)/4$ classes of cardinality 4, i.e. $2^{2n-2} - 2^{n-1}$.

Then, in order for System (1) to have exactly c solutions, we need that there exists i_1 solutions in classes of size 4 and i_2 in classes of size 2, where $2i_1 + 4i_2 = c$. We deduce that

$$\Pr\left[\mathcal{B}_F(a,b) = c\right] \ = \sum_{2i_1+4i_2=c} P_1(i_1)P_2(i_2) \ ,$$

where $P_1(i_1)$ (respectively $P_2(i_2)$) is the probability that there exists i_1 classes of size 4 (resp. 2) that are solutions of System (1). Let us now prove that the distributions of $P_1(i_1)$ and $P_2(i_2)$ are as stated in the theorem.

[4] The distribution of the coefficients in the LAT of random functions (not permutations) is also provided in [DR07].

Size 2. In this case, it holds that $y = x+b$ so that the lines of System (1) are identical. We assume that $F^{-1}(x)+F^{-1}(x+b) = a$ holds with probability $1/(2^n-1)$ as $F^{-1}(x)+F^{-1}(x+b)$ can take any value in $\mathbb{F}_2^n \backslash \{0\}$. Since there are 2^{n-1} such pairs, $P_1(i_1)$ corresponds to a binomial distribution with 2^{n-1} repetitions of a Bernoulli trial that succeeds with probability $(2^n-1)^{-1}$.

Size 4. The two equations of System (1) are now independent. Using the same reasoning as above, we assume that each line holds with probability $1/(2^n-1)$. Since there are $2^{2n-2}-2^{n-1}$ such pairs, $P_2(i_2)$ corresponds to a binomial distribution with parameters $2^{2n-2}-2^{n-1}$ and $(2^n-1)^{-2}$.

□

2.2 Anomalies in Table Coefficients Distributions

Building upon the general approach presented in [BP15], we can define several anomalies using the distribution of the coefficients in the tables of a permutation $F \in \mathfrak{S}_{2^n}$. We will then be able to estimate the values of the corresponding anomalies using the distributions derived in the previous section.

Maximum Value. For any table, the maximum absolute value of all coefficients is a natural property to use to construct an anomaly as the integers are ordered. Let $\mathsf{max}_T : \mathfrak{S}_{2^n} \to \mathbb{N}$ be the function mapping a permutation $F \in \mathfrak{S}_{2^n}$ to the maximum absolute value of the non-trivial coefficients in a table T. Then we can use the distributions in Propositions 1 and 2 as well as Theorem 1 to estimate the associated anomalies:

$$\mathsf{A}\left(\mathsf{max}_T(F)\right) \;=\; -(2^n-1)^2 \log_2 \left(\sum_{i=0}^{\mathsf{max}_T(F)} p_i \right) \;,$$

where p_i is the probability that $T(a,b) = i$. Indeed, there are only $(2^n-1)^2$ non-trivial coefficients in the DDT, LAT and BCT as the first row and column are fixed in each case. The (negative) anomalies corresponding to the differential uniformity, linearity and boomerang uniformity for $n = 8$ are given in Appendix B in Tables 4a, 4b and 4c respectively.

Maximum Value and Number of Occurrences. In $\mathfrak{S}_{2^8}$, the anomaly of a differential uniformity of 8 is equal to 16.2 but, for a differential uniformity of 6, it is 164.5. In order to have a finer grained estimate of how unlikely the properties of an S-box are, we combine the maximum coefficient in one of its tables with its number of occurrences as was first done in [BP15]. For a $2^n \times 2^n$ table of integers T, let MO be the function such that $\mathsf{MO}(T) = (c, m)$ where c is the maximum absolute value in T and m is its number of occurrences (where the first row and column are ignored). The set $\mathbb{N} \times \mathbb{N}$ in which the output of MO lives can be ordered using the lexicographic ordering, i.e. $(x, y) \leq (x', y')$ if and only if $x < x'$ or $x = x'$ and $y \leq y'$. We then define the differential, linear and boomerang anomalies of F as respectively

$$\mathsf{A}^{\mathsf{d}}(F) = \mathsf{A}\left(\mathsf{MO}(\delta_F)\right)\right), \; \mathsf{A}^{\ell}(F) = \mathsf{A}\left(\mathsf{MO}(\mathcal{W}_F)\right), \text{ and } \mathsf{A}^{\mathsf{b}}(F) = \mathsf{A}\left(\mathsf{MO}(\mathcal{B}_F)\right) \;.$$

This definition of the differential and linear anomalies matches with the one given in [Per19]. The boomerang anomaly was not used before. We also introduce the *negative* differential, linear and boomerang anomalies as the corresponding negative anomalies.

We estimate these anomalies for a table T using the following expression:

$$\mathsf{A}\big(\mathsf{MO}(T) \leq (c,m)\big) = -\log_2 \left(\sum_{k=0}^{m} \binom{(2^n-1)^2}{k} \times p_c^k \times \Big(\sum_{j=0}^{c-1} p_j \Big)^{(2^n-1)^2-k} \right) ,$$

where p_i is the probability that $T(a,b) = |i|$. For the corresponding negative anomaly, we use the following relation:

$$2^{\mathsf{A}(\mathsf{MO}(T)\leq(c,m))} + 2^{\overline{\mathsf{A}}(\mathsf{MO}(T)\leq(c,m))} = 1 + \binom{(2^n-1)^2}{m} p_c^m \Big(\sum_{j=0}^{c-1} p_j \Big)^{(2^n-1)^2-m} .$$

2.3 Tighter Advantage Estimations for Boomerang Attacks

The coefficient distribution we established in Theorem 1 can also be used to compute the expected value of a BCT coefficient. This in turn implies a better understanding of the advantage an adversary has in a boomerang attack.

Theorem 2. *The expected value for each BCT coefficient of a random permutation of $\mathfrak{S}_{2^n}$ converges towards 2 as n increases.*

Proof. Let $F \in \mathfrak{S}_{2^n}$ be picked uniformly at random. The expected value E of $\mathcal{B}_F(a,b)$ is $\sum_{c=0}^{2^n} \Pr\left[\mathcal{B}_F(a,b) = c\right] c$. Using Theorem 1, we express $\Pr\left[\mathcal{B}_F(a,b) = c\right]$ using two binomial distributions P_1 and P_2 so that

$$\begin{aligned} E &= \sum_{c=0}^{2^n} c \times \left(\sum_{2i_1+4i_2=c} P_1(i_1) P_2(i_2) \right) \\ &= \sum_{c=0}^{2^n} \sum_{i_1=0}^{2^{n-1}} \sum_{i_2=0}^{2^{n-2}} (2i_1+4i_2) P_1(i_1) P_2(i_2) \times [2i_1+4i_2 = c] , \end{aligned}$$

where the expression between the brackets is equal to 1 if $2i_1 + 4i_2 = c$, and 0 otherwise. Reordering the sums, we obtain the following expected value:

$$E = \underbrace{\sum_{i_1=0}^{2^{n-1}} \sum_{i_2=0}^{2^{n-2}} (2i_1+4i_2) P_1(i_1) P_2(i_2)}_{E(n)} \underbrace{\sum_{c=0}^{2^n} [2i_1+4i_2 = c]}_{\leq 1} . \tag{2}$$

We then approximate the binomial distributions P_1 and P_2 by Poisson distributions, namely $P_1(i) \approx \mathrm{Poisson}(i; 2^{-1}) = e^{-\frac{1}{2}} 2^{-i}/(i!)$ and $P_1(i) \approx \mathrm{Poisson}(i; 4^{-1}) = e^{-\frac{1}{4}} 4^{-i}/(i!)$. We get

$$\begin{aligned} E(n) &= \sum_{i_1=0}^{2^{n-1}} \sum_{i_2=0}^{2^{n-2}} (2i_1+4i_2) \frac{e^{-\frac{1}{2}} 2^{-i_1}}{i_1!} \frac{e^{-\frac{1}{4}} 4^{-i_2}}{i_2!} \\ &= \sum_{i_1=1}^{2^{n-1}} \frac{e^{-\frac{1}{2}} (\frac{1}{2})^{i_1-1}}{(i_1-1)!} \sum_{i_2=0}^{2^{n-2}} \frac{e^{-\frac{1}{4}} (\frac{1}{4})^{i_2}}{i_2!} + \sum_{i_1=0}^{2^{n-1}} \frac{e^{-\frac{1}{2}} (\frac{1}{2})^{i_1}}{i_1!} \sum_{i_2=1}^{2^{n-2}} \frac{e^{-\frac{1}{4}} (\frac{1}{4})^{i_2-1}}{(i_2-1)!} . \end{aligned}$$

As all sums converge towards 1 as n increases, the limit of $E(n)$ is 2. On the other hand, we remark that $E \leq E(n)$ because of Equation (2), and that

$$E \;\geq\; \sum_{i_1=0}^{2^{n-2}} \sum_{i_2=0}^{2^{n-3}} (2i_1+4i_2) P_1(i_1) P_2(i_2) \underbrace{\sum_{c=0}^{2^n} [2i_1+4i_2 = c]}_{=1} = E(n-1) ,$$

so $E(n-1) \leq E \leq E(n)$. As $E(n)$ converges to 2 as n increases, so does E. □

The expected probability of a boomerang characteristic

$$E_k^{-1}\left(E_k(x) \oplus b\right) \oplus E_k^{-1}\left(E_k(x \oplus a) \oplus b\right) = a$$

is thus 2^{1-n} and not 2^{-n} as we might expect.

2.4 Experimental Results

Verification. To check the validity of our approach to estimate the statistical anomalies, we picked 2^{21} permutations from $\mathfrak{S}_{2^8}$ uniformly at random. We then counted the number N_t of permutations F such that $\lfloor \mathsf{A}(F) \rfloor = t$, and we obtained the following results (only anomalies above 19 are listed):

$$\begin{array}{lll} \mathsf{A}^{\ell}(F): & N_{19} = 1, N_{21} = 1 & \overline{\mathsf{A}}^{\ell}(F): \quad N_{19} = 1 \\ \mathsf{A}^{\mathsf{d}}(F): & \text{See below} & \overline{\mathsf{A}}^{\mathsf{d}}(F): \quad N_{20} = 1 \\ \mathsf{A}^{\mathsf{b}}(F): & N_{19} = 3 & \overline{\mathsf{A}}^{\mathsf{b}}(F): \quad N_{20} = 2\ . \end{array}$$

We deduce that the anomalies other than $\mathsf{A}^{\mathsf{d}}(F)$ behave as we expect: in a set of size 2^t, we can expect to see about 1 permutation with an anomaly of t.

However, for $\mathsf{A}^{\mathsf{d}}(F)$, our results do not quite match the theory. Indeed, we have found too many permutations with a high differential anomaly for it to be a coincidence:

$$\begin{aligned} \mathsf{A}^{\mathsf{d}}(F): \ & N_{19} = 7, N_{20} = 8, N_{21} = 2, N_{22} = 1, N_{23} = 2, \\ & N_{24} = 1, N_{25} = 1, N_{26} = 1, N_{28} = 1\ . \end{aligned}$$

Recall that our estimates of the table-based anomalies rely on the assumption that the coefficients behave like independent random variables. While we experimentally found this assumption to yield accurate models in practice for all tables, it fails to accurately predict the behavior of the maximum value and its number of occurrences in the case of the DDT.

S-boxes from the Literature. We computed the statistical anomalies we defined above for several 8-bit S-boxes from the literature that we obtained from [PW17]. The results are given in Table 1. We also list the number N_V of vector spaces of dimension n contained in $\mathcal{Z}_s$; its importance will appear later in Section 3.

The statistical anomalies of the AES S-box, i.e. of the multiplicative inverse, are unsurprisingly very large. But they are *too* large: an anomaly cannot be higher than $\log_2(|\mathfrak{S}_{2^n}|)$. Our estimates do not hold for objects with properties as extreme as those of the inverse.

We can derive other results from this table. For example, 2-round SPNs have a high negative boomerang anomaly but 3-round ones loose this property. Classical 3-round Feistel networks, as used in ZUC_S0, have a boomerang uniformity which is maximum [BPT19b] so it is not surprising to see that they have a boomerang anomaly so high that we could not compute it. Even though the S-box of Zorro has a modified Feistel structure (it uses a sophisticated bit permutation rather than a branch swap), it still has a high negative boomerang anomaly.

As expected, the S-boxes that were generated using a random procedure have low positive and negative statistical anomalies. The S-box of MD2 was obtained using the digits of π, that of the newDES from the American declaration of independence, and that of Turing from the string "Alan Turing".

The correlation between the different statistical anomalies seems complex. On the one hand, there are S-boxes with very different linear and differential anomalies despite the fact that the square of the LAT coefficients corresponds to the Fourier transform of the DDT (see e.g. Skipjack). As evidenced by the anomalies of the S-boxes of Kalyna,

Table 1: The statistical anomalies and number of vector spaces for some S-boxes from the literature.

Type	Cipher	$\mathsf{A}^{\mathsf{d}}(s)$	$\overline{\mathsf{A}}^{\mathsf{d}}(s)$	$\mathsf{A}^{\ell}(s)$	$\overline{\mathsf{A}}^{\ell}(s)$	$\mathsf{A}^{\mathsf{b}}(s)$	$\overline{\mathsf{A}}^{\mathsf{b}}(s)$	$N_V(s)$
Inverse	AES	7382.13	0.00	3329.43	0.00	9000.05	0.00	2
Logarithm	BelT	74.79	0.00	122.97	0.00	0.98	0.40	2
TKlog	Kuznyechik	80.63	0.00	34.35	0.00	14.18	0.00	3
SPN (2S)	CLEFIA_S0	2.56	0.19	25.62	0.00	0.00	15.60	6
	Enocoro	1.92	0.36	3.26	0.15	0.00	15.60	6
	Twofish_p0	1.36	0.70	3.16	0.17	0.00	33.84	6
	Twofish_p1	1.34	0.72	3.16	0.17	0.00	25.82	6
SPN (3S)	Iceberg	17.15	0.00	3.58	0.10	0.02	3.87	2
	Khazad	16.94	0.00	3.16	0.17	0.98	0.40	2
Feistel	Zorro	2.19	0.27	3.37	0.13	0.00	25.82	2
	ZUC_S0	16.15	0.00	3.16	0.17	0.00	NaN	368
Hill climbing	Kalyna_pi0	104.22	0.00	235.77	0.00	29.67	0.00	2
	Kalyna_pi1	122.64	0.00	268.07	0.00	29.67	0.00	2
	Kalyna_pi2	129.87	0.00	239.28	0.00	5.99	0.00	2
	Kalyna_pi3	122.64	0.00	242.92	0.00	26.44	0.00	2
Random	Turing	0.18	1.94	1.84	0.17	0.98	0.40	2
	MD2	1.36	0.70	0.10	2.41	0.98	0.40	2
	newDES	0.44	0.73	0.32	1.95	0.14	1.86	2
Unknown	Skipjack	0.18	1.94	54.38	0.00	0.98	0.40	2

which were obtained using a hill climbing method optimizing the differential and linear properties [KKO13], these improvements lead to an observable increase of the boomerang anomaly but it can be marginal.

3 Identifying Structures

In this section, we go through the most common S-box structures, and present for each of them the density of the set of such S-boxes (up to affine-equivalence) and the methods that can be used to identify them. In practice, S-boxes operating on at least 6 bits usually fall into two categories: those that are based on the inverse in the finite field $\mathbb{F}_{2^n}$, and those using block cipher structures.

In both cases, the permutations are usually composed with affine permutations. In the context of white-box cryptography, it is common to compose functions with secret affine permutations so as to obfuscate the logic of the operations used. Hence, for both decomposing S-boxes and attacking white-box implementation, it is necessary to be able to remove these affine layers.

While recovering a monomial structure is simple even when it is masked by affine permutations (see Section 3.1 and our results on the S-box of Chiasmus), it is not the case with block cipher structures. In this section, we show how the the recovery of the pattern used in [BPU16] to remove the affine layers of the Russian S-box can be efficiently automatized (Section 3.2), and applied to both SPNs (Section 3.3) and Feistel network (Section 3.4). The core algorithm needed for these attacks is one returning all the vector spaces contained in a set of elements of $\mathbb{F}_2^n$. We will present such an algorithm in Section 4.

These techniques allow us to identify the *structural anomalies* in S-boxes. In order to estimate the anomaly associated with each structure, we upper bound the number of permutation that can be built using each of those that we consider. The corresponding anomalies are summarized in Section 3.5.

3.1 Multiplicative Inverse

Such permutations have a very simple structure: there exists two affine permutations $A : \mathbb{F}_2^n \to \mathbb{F}_{2^n}$ and $B : \mathbb{F}_{2^n} \to \mathbb{F}_2^n$ such that the permutations F can be written $F = B \circ G \circ A$, where G is the permutation of $\mathbb{F}_{2^n}$ defined by $G(x) = x^{2^n-2}$. Their use was introduced in [Nyb94]; the AES [AES01] uses such an S-box.

In practice, the implementation of G requires the use of an encoding of the elements of $\mathbb{F}_{2^n}$ as elements of $\mathbb{F}_2^n$. Usually, it is achieved by mapping $x = (x_0, ..., x_{n-1}) \in \mathbb{F}_2^n$ to $\sum_{i=0}^{n-1} x_i \alpha^i$, where $\alpha \in \mathbb{F}_{2^n}$ is the root of an irreducible polynomial with coefficients in $\mathbb{F}_2$ of degree n. However, this encoding can be seen as being part of A and B.

Density of the set. There is only one function $x \mapsto x^{2^n-2}$. However, there are fewer than $(|\mathcal{L}_{2^n}|2^n)^2$ distinct permutations affine-equivalent to it. Indeed, $(x \times m)^{2^n-2} = x^{2^n-2} \times m^{2^n-2}$, meaning that for a given pair (A, B) of permutations of $\mathcal{L}_{2^n}$ we can define $2^n - 1$ pairs $(A_i, B_i) \in (\mathcal{L}_{2^n})^2$ such that $B_i \circ G \circ A_i = B_j \circ G \circ A_j$ for all i, j. The same reasoning applies to the Frobenius automorphisms because $(x^{2^i})^{2^n-2} = (x^{2^n-2})^{2^i}$. In the end, there are at most

$$\underbrace{|\mathcal{L}_{2^n}|^2}_{L_A \text{ and } L_B} \times \underbrace{2^{2n}}_{c_A \text{ and } c_B} \times \frac{1}{\underbrace{(2^n-1)}_{\text{multiplication}} \times \underbrace{n}_{\text{Frobenius}}} = \frac{2^n}{n} \times (|\mathcal{L}_{2^n}|)^2$$

distinct permutations affine-equivalent to the multiplicative inverse.

How to recognize them? The Chinese cipher SMS4 [Dt08] uses an 8-bit S-box whose structure was not explained. This prompted Liu et al. to try and recover said structure [LJH+07]. They successfully identified it as being affine equivalent to the multiplicative inverse using an *ad hoc* method.

There is a simple test that can be applied to check if a permutation is affine-equivalent to the multiplicative inverse when the input/output size is even.

Lemma 1. *Let $s : x \mapsto x^{2^n-2}$ be a permutation of $\mathbb{F}_{2^n}$ and $F \in \mathfrak{S}_{2^n}$ with n even be such that $F = B \circ s \circ A$ where $A : x \mapsto L_A(x) + c_A$ and $B : x \mapsto L_B(x) + c_B$ are affine permutations. Let $\{(a_i, b_i)\}$ be the set of all coordinates such that $\delta_F(a_i, b_i) = 4$. Then it holds that $b_i = L_B\left(L_A(a_i)^{2^n-2}\right)$ for all i, meaning that $a_i \mapsto b_i$ and s are identical up to translations.*

Proof. We have that $(x + a)^e + x^e = b$ has as many solutions as $(y + 1)^e + y^e = b/a^e$, meaning that all rows of its DDT contain the same coefficients: $\delta_s(a, b) = \delta_s(1, b/a^e)$. In the case of the inverse for n even, $\delta_s(1, c) \in \{0, 2\}$ for all $c \neq 1$ and $\delta_s(1, 1) = 4$. Such a function was called *locally-APN* in [BCC11].

In our case, we have that $\delta_F(a, b) = \delta_s\left(L_A(a), L_B^{-1}(b)\right)$. Using the property we just established with $e = 2^n - 2$, we get $\delta_F(a, b) = \delta_s\left(1, L_B^{-1}(b)/(L_A(a))^{2^n-2}\right)$, where the second coordinate simplifies into $L_B^{-1}(b) \times L_A(a)$. As a consequence, $\delta_F(a, b) = 4$ if and only if $L_B^{-1}(b) = (L_A(a))^{2^n-2}$, which is equivalent to $b = L_B\left(L_A(a)^{2^n-2}\right)$. □

In [Sch14] and [STW13], two separate teams independently recovered the secret block cipher Chiasmus from an encryption tool called GSTOOL. Chiasmus is a German designed 64-bit block cipher which uses two S-boxes S and S^{-1}. Schuster had the intuition that it was built similarly to the AES S-box. He was right. Using Lemma 1 and the linear equivalence algorithm of [BDBP03], we found that the S-box of Chiasmus is also based on a finite field inversion. However, unlike in the AES, it uses *two* affine mappings with non-zero constants. A script generating the S-box of Chiasmus is provided in Appendix E. The S-box itself can be found in a SAGE [Dev17] module [PW17].

We could also have recovered this structure using directly the algorithm of Biryukov et al. [BDBP03] or the more recent one of Dinur [Din18]. However, the above approach and these algorithms share the same shortcoming when it comes to identifying the structure in an unknown S-box $F \in \mathfrak{S}_{2^n}$: if we do not know the exact S-box to which F might be affine-equivalent then they cannot be applied. Even if we know that it might be affine-equivalent to an SPN or a Feistel network, we cannot find the corresponding affine masks.

To solve this problem, we identify patterns in the LAT of the permutations with specific structures that are present regardless of the subfunctions they contain. As a consequence, they can always be detected.

3.2 TU-Decomposition

The TU-decomposition is a general structure that was first introduced in [BPU16] where it was shown that the S-box of the latest Russian standards has such a structure. Later, it was encountered again in the context of the *Big APN problem*, a long standing open question in discrete mathematics. Indeed, the only known solution to this problem is a sporadic 6-bit APN permutation that was found by Dillon et al. [BDMW10] and which was proved in [PUB16] to yield a TU-decomposition. This structure was then further decomposed to obtain the so-called *open butterfly*. As we will show below, some Feistel and SPN structures also share this decomposition. Thus, the tools that can find TU-decomposition can also be used to identify these structures even in the presence of affine masks.

Definition 2 (TU_t-decomposition)**.** Let n and t be integers such that $0 < t < n$. We say that $F \in \mathfrak{S}_{2^n}$ has a TU_t*-decomposition*[5] if there exists:

- a family of 2^{n-t} permutations $T_y \in \mathfrak{S}_{2^t}$ indexed by $y \in \mathbb{F}_2^{n-t}$,
- a family of 2^t permutations $U_x \in \mathfrak{S}_{2^{n-t}}$ indexed by $x \in \mathbb{F}_2^t$, and
- two linear permutations $\mu : \mathbb{F}_2^n \to (\mathbb{F}_2^t \times \mathbb{F}_2^{n-t})$ and $\eta : (\mathbb{F}_2^t \times \mathbb{F}_2^{n-t}) \to \mathbb{F}_2^n$

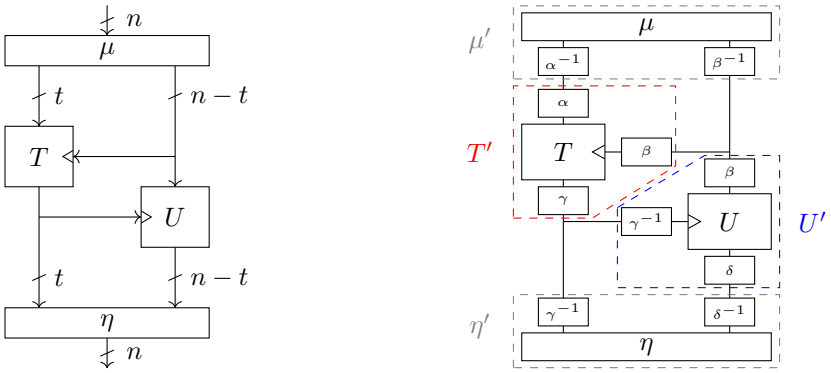
such that $F = \eta \circ G \circ \mu$, where G is the permutation of $\mathbb{F}_2^t \times \mathbb{F}_2^{n-t}$ such that $G(x, y) = \big(T_y(x), U_{T_y(x)}(y)\big)$. This structure is presented in Figure 1a.

In other words, $F \in \mathfrak{S}_{2^n}$ has a TU_t-decomposition if and only if it is affine-equivalent to $G \in \mathfrak{S}_{2^n}$ with the following property: if $G_{\restriction t}$ is the restriction of G to its t bits of highest weight then $x \mapsto G_{\restriction t}(x||a)$ is a permutation for all $a \in \mathbb{F}_2^{n-t}$.

Density of the set. In order to define a permutation with a TU_t-decomposition, we need to choose 2^{n-t} permutations of $\mathfrak{S}_{2^t}$, 2^t permutations of $\mathfrak{S}_{2^{n-t}}$ and two linear permutations operating on n bits. However, several of the permutations generated in this way will be identical. Indeed, we can compose each T_y with a t-bit linear permutation $\alpha \in \mathcal{L}_{2^t}$ to obtain a permutation $T'_y = T_y \circ \alpha$. If we use T'_y and compose μ with α^{-1}, then we obtain the same overall permutation as when T_y and μ are used. More equivalent modifications can be made using linear permutations $\beta \in \mathcal{L}_{2^{n-t}}$, $\gamma \in \mathcal{L}_{2^t}$ and $\delta \in \mathcal{L}_{2^{n-t}}$, as summarized in Figure 1b. Hence, the total number of n-bit permutations with TU_t-decompositions is at most

$$\#\mathsf{TU}_t \;\leq\; \underbrace{|\mathfrak{S}_{2^t}|^{2^{n-t}}}_{T_y} \times \underbrace{|\mathfrak{S}_{2^{n-t}}|^{2^t}}_{U_x} \underbrace{\left(\frac{|\mathcal{L}_{2^n}|}{|\mathcal{L}_{2^t}| \times |\mathcal{L}_{2^{n-t}}|}\right)^2}_{\mu \text{ and } \eta}.$$

[5] This is a simplified version of the TU_t-decomposition compared to [CP19]. Indeed, in that paper, the authors only impose that $T_y \in \mathfrak{S}_{2^{n-t}}$; U_x may have collisions. Since we are only considering bijective S-boxes here, we consider that $U_x \in \mathfrak{S}_{2^t}$.



(a) TU_t-decomposition. (b) Composing its components with linear permutations.

Figure 1: Two functionally equivalent permutations.

This quantity is only a bound as permutations that are self affine-equivalent lead to identical permutations with different μ and η. We used this bound to compute the anomaly associated to the presence of a TU_t-decomposition in a permutation. It is given in Section 2.

How to recognize them? Let $F \in \mathfrak{S}_{2^n}$ be a permutation. As was established in Proposition 6 of [CP19], the presence of a TU_t-decomposition is equivalent to the presence of a specific vector space of zeroes of dimension n in $\mathcal{Z}_F$. Let us first recall the corresponding proposition in the particular case of permutations.

Proposition 4 ([CP19])**.** *Let $F \in \mathfrak{S}_{2^n}$ and let $\mathcal{Z}_F$ be its Walsh zeroes. Then F has a TU_t-decomposition without any affine layers if and only if $\mathcal{Z}_F$ contains the vector space*

$$\left\{(0||a,b||0), a \in \mathbb{F}_2^t, b \in \mathbb{F}_2^{n-t}\right\} \ .$$

The advantage of Proposition 4 is that the pattern described depends only on the presence of a TU_t-decomposition and not on the specifics of the components T and U. Furthermore, recall that if $G = L_2 \circ F \circ L_1$ for some linear permutations L_1 and L_2 then $\mathcal{W}_G(a,b) = \mathcal{W}_F\left((L_1^{-1})^T(a), L_2^T(b)\right)$.

Corollary 1. *Let $F \in \mathfrak{S}_{2^n}$ and let $\mathcal{Z}_F$ be its Walsh zeroes. Then F has a TU_t-decomposition with linear permutations μ and η if and only if*

$$\left\{\left((\mu^{-1})^T(0,a), \eta^T(b,0)\right), a \in \mathbb{F}_2^t, b \in \mathbb{F}_2^{n-t}\right\} \ \subset \ \mathcal{Z}_F \ .$$

It is therefore sufficient to look for all the vector spaces of dimension n contained in $\mathcal{Z}_F$ to see if F has TU_t-decomposition. If we find a vector space that is not the Cartesian product of a subspace of $\{(x,0), x \in \mathbb{F}_2^n\}$ with a subspace of $\{(0,y), y \in \mathbb{F}_2^n\}$ then F does not have a TU_t-decomposition but there exists a linear function L of $\mathbb{F}_2^n$ such that $F + L$ does [CP19]. Regardless, the key tool that allows the search for TU-decomposition is an efficient algorithm returning all the vector spaces of a given dimension that are contained in a set of elements of $\mathbb{F}_2^n$. Indeed, finding such vector spaces will allow us to recover all the values of $(\mu^{-1})^T(0,a)$ and $\eta^T(b,0)$ for $(a,b) \in \mathbb{F}_2^t \times \mathbb{F}_2^{n-t}$, from which we will deduce information about μ and η. We present such an algorithm in Section 4 and we used it as a subroutine of program finding a TU_t-decomposition automatically (see Appendix C).

As observed in [CP19], the number of vector spaces of dimension n in $\mathcal{Z}_F$ is the same as the number of vector spaces of dimension n in the set of the coordinates of the zeroes in the DDT. Thus, we could equivalently present our results in terms of DDT.

3.3 Substitution-Permutation Networks

An n-bit SPN interleaves the parallel application of k possibly distinct m-bit S-boxes with n-bit linear permutations, where $k \times m = n$. We use the common [BS01] notation AS to denote a linear layer followed by an S-box layer. A SAS structure is depicted in Figure 2a.

(a) A two-round SPN (SAS). (b) A 3-round Feistel network.

Figure 2: Two block-cipher-like S-box structures.

Let us estimate the number of r-round SPNs. As the S-box layers are interleaved with linear layers, we need to consider not the size of $\mathfrak{S}_{2^m}$ but instead the number of linear equivalence classes, which is at most $|\mathfrak{S}_{2^m}|/|\mathcal{L}_{2^m}|^2$. The number of permutations with a $A(SA)^r$ structure is then at most

$$\#A(SA)^r \ \leq \ \left(\frac{|\mathfrak{S}_{2^m}|}{|\mathcal{L}_{2^m}|^2}\right)^{rn/m} \times |\mathcal{L}_{2^n}|^{r+1} \ .$$

The corresponding anomalies for some values of n are given in Section 3.5.

How to recognize them? First of all, the algebraic degree of a 2-round SPN is at most equal to $n-2$ [BC13]. Hence, if a permutation is of degree $n-1$, it cannot have such a structure.

In Theorem 3, we will establish the existence of specific vector space of zeroes in the LAT of a 2-round SPN. However, in order to properly state this theorem, we first need to introduce the following notion.

Definition 3 (m-Valid minors)**.** Let k, m and n be integers such that $n = k \times m$. Let $L \in \mathcal{L}_{2^n}$ be a linear permutation. We define it using a k^2 block matrices $L_{i,j}$ of dimension $m \times m$:

$$L = \begin{bmatrix} L_{0,0} & ... & L_{0,k-1} \\ ... & & ... \\ L_{k-1,0} & ... & L_{k-1,k-1} \end{bmatrix} .$$

We call a minor of the matrix L *m-valid* if there exists a pair I, J of subsets of $\{0, ..., k-1\}$ which are of the same size $0 < |I| = |J| < k$ and such that the rank of $L_{I,J} = [L_{i,j}]_{i \in I, j \in J}$ is equal to m.

In other words, an m-valid minor of L is a non-trivial minor of L that is obtained by taking complete m-bit chunks of this matrix, and which has maximum rank.

Theorem 3. *Let $F \in \mathfrak{S}_{2^n}$ be an ASASA structure built using L as its central linear layer and two layers of m-bit S-boxes. For each $I, J \subsetneq \{0, ..., k-1\}$ defining an m-valid minor of L, there exists a vector space of zeroes of dimension n in $\mathcal{Z}_F$.*

Proof. Because of Corollary 1, we restrict ourselves to the SAS structure. If we let the input blocks corresponding to the indices in I take all $2^m|I|$ possible values, then the output blocks with indices in J will also take all $2^{m|J|} = 2^{m|I|}$ possible values. There is thus a corresponding $\mathsf{TU}_{m|I|}$-decomposition and hence a corresponding vector space in $\mathcal{Z}_F$. □

This verification is less efficient than the dedicated cryptanalysis methods presented in [MDFK18]. However, the aim here is not so much to recover the ASASA structure used, it is rather to identify the S-box as having such a structure in the first place. Using the following corollary, we can immediately understand why $N_V = \binom{2\times 2}{2} = 6$ for several S-boxes in Table 1: it is a direct consequence of their 2-round SPN structure and of the strong diffusion of their inner linear layer.

Corollary 2. *Let $F \in \mathfrak{S}_{2^n}$ be the SAS structure built using L as its linear layer and two layers of m-bit S-boxes, where $n = k \times m$. If L is MDS over the alphabet of S-box words, then $\mathcal{Z}_F$ contains at least $\binom{2k}{k}$ vector spaces of dimension n.*

Proof. As L is MDS, all its minors and in particular those corresponding to the definition of m-minors have a maximum rank. There are $\sum_{i=1}^{k} \binom{k}{i} \times \binom{k}{i}$ such m-minors, to which we add the "free" vector space $\{(x, 0), x \in \mathbb{F}_2^n\}$ which is always present: there are at least $\sum_{i=0}^{k} \binom{k}{i}^2 = \binom{2k}{k}$ vector spaces in $\mathcal{Z}_F$. □

3.4 Feistel Networks

The Feistel structure is a classical block cipher construction which is summarized in Figure 2b. The number of permutations that are affine-equivalent to r-round Feistel networks that use permutations as the round functions is at most equal to

$$\underbrace{|\mathfrak{S}_{2^{n/2}}|^r}_{\text{round funcs.}} \times \underbrace{\frac{1}{(2^n)^{\lceil \frac{n}{2} \rceil}}}_{\text{constants}} \times \underbrace{|\mathcal{L}_{2^n}|^2}_{\text{outer layers}} \times \underbrace{\frac{1}{|\mathcal{L}_{2^{n/2}}|^2}}_{\text{branch transforms}} .$$

Indeed, we can apply $n/2$-bit linear permutations L and L' to each branch and, provided that the round functions are modified, we can cancel them out by applying L^{-1} and $(L')^{-1}$ on the output branches. We can also add constants freely to the output of the first $\lceil r/2 \rceil$ round functions, as explained in [BLP16].

How to recognize them? There are efficient function-recovery techniques for up to 5-round Feistel networks [BLP16]. However, as soon as affine masks are added, the corresponding techniques can no longer be applied. Still, as with the SPN structure, Feistel networks with few rounds exhibit specific vector spaces in their Walsh zeroes as was already observed for 4-round Feistel network in [BPU16]. This means that it is possible to detect such structures using the vector spaces in their Walsh zeroes.

Theorem 4 ([BPU16])**.** *Let F be a 4-round Feistel network such that round functions 2 and 3 are permutations. Then $\mathcal{W}_F(x||y, 0||y) = 0$ for all x, y in $\mathbb{F}_2^{n/2}$.*

This observation also holds for a 3-round Feistel. In fact, there are more vector spaces in such a structure.

Theorem 5. *Let F_0, F_1 and F_2 be functions of $\mathbb{F}_2^{n/2}$ such that $F_1 \in \mathfrak{S}_{2^{n/2}}$. Let $F \in \mathbb{F}_2^n$ be the 3-round Feistel network using F_0, F_1 and F_2 as its round functions. Then the set $\mathcal{Z}_F$ contains the following vector spaces of dimension n:*

1. $\{(x, 0), x \in \mathbb{F}_2^n\}$, $\{(0, y), y \in \mathbb{F}_2^n\}$,

2. $\{(x||0, y||0), (x, y) \in \mathbb{F}_2^{n/2} \times \mathbb{F}_2^{n/2}\}$,

3. $\{(x||y, x||0), (x, y) \in \mathbb{F}_2^{n/2} \times \mathbb{F}_2^{n/2}\}$, $\{(x||0, x||y), (x, y) \in \mathbb{F}_2^{n/2} \times \mathbb{F}_2^{n/2}\}$,

4. $\{(x||y, 0||y), (x, y) \in \mathbb{F}_2^{n/2} \times \mathbb{F}_2^{n/2}\}$, $\{(0||y, x||y), (x, y) \in \mathbb{F}_2^{n/2} \times \mathbb{F}_2^{n/2}\}$,

the fourth category being present if F_0 and F_2 are in $\in \mathfrak{S}_{2^{n/2}}$.

The proof of this theorem follows from direct applications of results in [CP19] and of these observations:

- if the 3-round Feistel network implies a specific vector space, it also implies the one with the coordinates swapped because its inverse is also a 3-round Feistel network,
- $(x, y) \mapsto F(x, y) \oplus (x, 0)$ is a permutation if $F_1 \in \mathfrak{S}_{2^{n/2}}$, and
- $(x, y) \mapsto F(x, y) \oplus (0, y)$ has a $\mathsf{TU}_{n/2}$-decomposition if $F_2 \in \mathfrak{S}_{2^{n/2}}$.

The details are provided in Appendix A.

3.5 Structural Anomalies

In light of our results, we can quantify the anomaly associated to the presence of various structures. In this case, the mapping P considered maps $\mathfrak{S}_{2^n}$ to $\{0, 1\}$: a permutation has a specific structure or it does not. The anomaly associated to a given structure is then

$$\mathsf{A}_{\text{structure}} \;=\; -\log_2\left(\frac{\left|\{G \in \mathfrak{S}_{2^n}, G \text{ has the structure}\}\right|}{|\mathfrak{S}_{2^n}|}\right) ,$$

meaning that the set sizes we extracted above allow us to quantify the anomalies associated to the TU_t-decomposition, the SPN structure, the Feistel network and the TKlog (see below for the latter). The corresponding anomalies are summarized in Table 2 for different values of n.

The existence of a TU-decomposition with $t = 1$ for $F \in \mathfrak{S}_{2^n}$ is equivalent to the presence of a component with a linear structure [CP19], i.e. to the existence of $a \in \mathbb{F}_2^n$ such that the Boolean function $x \mapsto a \cdot F(x)$ has a probability 1 differential. Thus, the corresponding row of Table 2 gives the anomaly corresponding to linear structures.

We can also compute the anomaly associated to the TKlog structure [Per19] used in the S-box of Streebog and Kuznyechik [Fed12, Fed15] called $\pi \in \mathfrak{S}_{2^8}$. A TKlog is a $2m$-bit permutation parametrized by an affine function $\kappa : \mathbb{F}_2^m \to \mathbb{F}_{2^{2m}}$ such that $\kappa(x) = \Lambda(x) \oplus \kappa(0)$ for some linear function Λ. This function must be such that $\mathrm{Im}(\Lambda) \cup \mathbb{F}_{2^m}$ spans $\mathbb{F}_{2^{2m}}$. The TKlog also depends on a permutation s of $\mathfrak{S}_{2^m-1}$. It is defined as follows

$$\begin{cases} \pi(0) & = \kappa(0), \\ \pi\left(\alpha^{(2^m+1)j}\right) & = \kappa(2^m - j), \text{ for } 1 \le j < 2^m - 1, \\ \pi\left(\alpha^{i+(2^m+1)j}\right) & = \kappa(2^m - i) + \alpha^{(2^m+1)s(j)}, \text{ for } i < 2^m + 1, j < 2^m - 1 \ , \end{cases} \tag{3}$$

where α is a root of a primitive polynomial p of degree $2m$, so that α^{2^m+1} is a multiplicative generator of $\mathbb{F}_{2^m}^*$. The number of TKlog, is then given by

$$\underbrace{\prod_{i=m}^{2m-1} (2^{2m} - 2^i)}_{\Lambda} \times \underbrace{|\mathfrak{S}_{2^m-1}|}_{s} \times \underbrace{\left(\phi(2^{2m} - 1)/(2m)\right)}_{\#\text{primitive polynomials}} \times \underbrace{2^{2m}}_{\kappa(0)}$$

where ϕ is Euler's totient function. As for the inverse function, the encoding of the elements of $\mathbb{F}_{2^{2m}}$ as binary strings can be considered to be part of the outer affine layers.

Table 2: Upper bounds on the anomalies of the affine-equivalence to some structures. For the TKlog, "AE" corresponds to permutations affine-equivalent to some TKlog and "pure" to TKLog themselves. S/r is the number of S-boxes used in each round, i.e. the number that are applied in parallel.

Structure.	Parameters	$n=6$	$n=8$	$n=12$	$n=16$
$x \mapsto x^{2^n-2}$	–	236.1	1570.6	42981.2	953548.5
TKlog	"pure"	258.7	1601.5	42870.7	952207.7
	AE	184.3	1469.0	42574.2	951683.2
TU-dec.	$t=1$	8.8	95.7	1997.7	32699.7
	$t=n/2$	13.0	201.1	5215.3	91571.2
SPN	ASASA, $S/r=2$	192.7	1435.4	41913.5	947036.0
	ASASASA, $S/r=2$	158.2	1342.3	41316.3	943662.7
Feistel	3-round, $F_i \in \mathfrak{S}_{2^{n/2}}$	205.5	1443.3	41898.2	946980.9
	4-round, $F_i \in \mathfrak{S}_{2^{n/2}}$	220.8	1487.6	42194.2	948664.9

4 Vector Spaces Extraction Algorithms

Let $\mathcal{S}$ be a set of elements of $\mathbb{F}_2^n$. In this section, we describe an algorithm which extracts all the vector spaces of dimension at least d that are completely contained in $\mathcal{S}$. As established in the previous section, the ability to solve this problem will allow us to identify TU-decompositions, some SPNs, and 3,4-round Feistel networks even in the presence of affine encodings. It can also test the CCZ-equivalence [CCZ98] of a function to a permutation, as was done by Dillon et al. [BDMW10] to find the first APN permutation operating on an even number of bits.

Our results can be interpreted using both the ordering relation over the integers and by reasoning over the respective position of the zeroes of the elements in $\mathbb{F}_2^n$. The following lemma links these two views.

Definition 4 (Most Significant Bit)**.** Let $x \in \mathbb{F}_2^n$ and let us write $x = (x[0], ..., x[n-1])$ where $x[0]$ is the least significant bit. We denote $\mathrm{MSB}(x)$ the greatest index i such that $x[i] = 1$.

Lemma 2. *For any $x \in \mathbb{F}_2^n$, it holds that*

$$x < x \oplus a \;\Leftrightarrow\; x[MSB(a)] = 0 \ ,$$

where the order relation is obtained by interpreting x and $x \oplus a$ as the binary representations of integers.

4.1 A Simple Approach and How Ours Improves It

Let us first present a naive approach to solving this problem. At its core, this approach is a tree search that builds the complete vector spaces iteratively.

Starting from a specific element $x \in \mathcal{S}$ and vector space $V_x = \{0, x\}$, we loop over all the elements y such that $y > x$ and check whether $(x \oplus y) \in \mathcal{S}$, in which case we build $V_{x,y} = V_x \cup \{y \oplus v, v \in V_x\}$. We then repeat this process by looking for $z > y$ such that $(z \oplus v) \in \mathcal{S}$ for all $v \in V_{x,y}$. This process can then be iterated until complete bases $(x, y, z, ...)$ of vector spaces are found. Our approach is based on the same principles but it significantly outperforms this naive algorithm by solving its two main shortcomings.

First, the basis of a vector space is not unique. The condition that it be ordered, which is implied by the algorithm sketched above, is not sufficient to ensure uniqueness. This implies that the algorithm will be slowed down by the exploration of the branches that actually correspond to identical spaces, and that a post processing checking for duplicated spaces will be needed. Our algorithm will solve this problem and return exactly one basis for each vector space contained in $\mathcal{S}$. These bases are called *Gauss-Jordan Bases (GJB)* and are introduced in Section 4.2.

Second, at each iteration, we need to consider all $z \in \mathcal{S}$ such that z is strictly larger than the largest vector already in the basis being built. In our approach, we update at each iteration a set that contains all the elements z that could be used to construct a larger basis using a process which we call *vector extraction* (see Section 4.3). Like in the algorithm above, this set only contains elements that are strictly greater than the previous bases elements. However, it is also strictly larger than all the elements in the vector space spanned by this basis and its size is reduced by at least a factor 2 at each iteration. Using vector extractions, we can also skip the test that $(z \oplus v) \in \mathcal{S}$ for all v in the current vector space which will increase the speed of our algorithm.

Besides, in each iteration, we use a heuristic method to consider only a subset of this set of z which is based on the number and positions of its zeroes, the *Bigger MSB Condition*.

In summary, we improve upon the algorithm above in the following ways:

- we construct exactly one basis per vector space contained in $\mathcal{S}$ (using GJB, see Section 4.2),
- we significantly reduce the number of vectors that can be considered in the next iterations (using vector extractions, see Section 4.3), and
- we further decrease the number of vectors that need to be explored at a given iteration using a specific filter (using the Bigger MSB condition, see Section 4.4).

Finally, the vector space extraction algorithm itself is presented in Section 4.5. An algorithm extracting affine spaces which uses the former as a subroutine is presented in Appendix D. We provide an implementation along with this submission, it is described in Appendix C.

In [CDDL06], Canteaut et al. introduced an algorithm which, given an n-bit Boolean function f, lists all the affine spaces of dimension m such that f is constant (or affine) on them. Our algorithm can easily perform the same task. Indeed, f is affine on a subspace U if and only if $\{x||f(x), x \in U\}$ is an affine subspace, meaning that our affine space search algorithms can list all such spaces.

Using our implementation (see Appendix C), we only need about 12 min to reprove their Fact 22 which deals with a 14-bit Boolean function while they claim a runtime of 50 h in this case. Our machine is more recent and thus likely faster than theirs but not by a factor 250: our algorithm is inherently more efficient. It is also far more versatile, as we have established above.

4.2 Gauss-Jordan Bases

These objects are those which our vector space search will actually target. They were described in the context of Boolean functions in [CDDL06].

Definition 5 (GJB [CDDL06])**.** For any vector space V of dimension d, the *Gauss-Jordan Basis (GJB) of* V is the set $\{v_0, ..., v_{d-1}\}$ such that $\langle v_0, ..., v_{d-1}\rangle = V$ which is the smallest such set when sorted in lexicographic order.

For any space V there is *exactly one* GJB. Indeed, we can write down all of its bases, sort the elements in each of them in increasing order and then sort the reordered bases in

lexicographic order. This implies that $v_i < v_{i+1}$ for all i. Some key properties of GJBs are given by the following lemma.

Lemma 3. *GJBs have the following properties.*

1. *If $\{v_0, ..., v_i\}$ is the GJB of $\langle v_0, ..., v_i \rangle$ then $\{v_0, ..., v_{i-1}\}$ is a GJB.*
2. *The basis $\{v_0, ..., v_{d-1}\}$ is a GJB if and only if*

$$\begin{cases} \forall j \in \{0, ..., d-2\},\ MSB(v_j) < MSB(v_{j+1}) \\ \forall i \in \{1, ..., d-1\}, \forall j \in \{0, ..., i-1\}, v_i[MSB(v_j)] = 0\ . \end{cases} \tag{4}$$

3. *If $\{v_0, ..., v_{d-1}\}$ is a GJB then, for all $j \in \{0, ..., d-1\}$, $\langle v_0, ..., v_{d-1} \rangle$ contains exactly 2^j elements x such that $MSB(x) = MSB(v_j)$.*

Proof. We prove each point separately.

Point 1. A basis of $\langle v_0, ..., v_{i-1} \rangle$ lexicographically smaller than $\{v_0, ..., v_i\}$ could be used to build a basis of $\langle v_0, ..., v_i \rangle$, lexicographically smaller than its GJB, which is impossible.

Point 2. We prove each direction of the equivalence separately.

$\Rightarrow$ Suppose that $\{v_0, ..., v_{d-1}\}$ is indeed a GJB. Then $\mathrm{MSB}(v_j) = \mathrm{MSB}(v_{j+1})$ would imply that $\mathrm{MSB}(v_j \oplus v_{j+1}) < \mathrm{MSB}(v_j)$ which, in particular, would imply that $v_j \oplus v_{j+1} < v_j$. This would contradict that $\{v_0, ..., v_{d-1}\}$ is a GJB. Similarly, $\mathrm{MSB}(v_j) > \mathrm{MSB}(v_{j+1})$ would imply $v_j > v_{j+1}$ which is also a contradiction. We deduce that $\mathrm{MSB}(v_j) < \mathrm{MSB}(v_{j+1})$ for any $0 \leq j < d-1$. Suppose now that $v_i[\mathrm{MSB}(v_j)] = 1$ for some $j < i$. We deduce from Lemma 2 that $v_i \geq v_i \oplus v_j$, which is again a contradiction because $\{v_0, ..., v_{d-1}\}$ is minimal. We have thus established that if $\{v_0, ..., v_{d-1}\}$ is a GJB then it must satisfy the conditions in Equation (4).

$\Leftarrow$ Let us now assume that these conditions hold. In this case, we have that $v_i < v_i \oplus \bigoplus_{j \in I} v_j$ for any subset I of $\{0, ..., i-1\}$ because the MSB of $\bigoplus_{j \in I} v_j$ is always strictly smaller than $\mathrm{MSB}(v_i)$ and because of Lemma 2. Thus, adding v_i at the end of $\{v_0, ..., v_{i-1}\}$ yields a GJB of $\langle v_0, ..., v_i \rangle$. A simple induction then gives us the result.

Point 3. Using the first point of this lemma allows us to proceed via a simple induction over the size of the basis. If the basis is simply $\{v_0\}$ then the lemma obviously holds. Then, adding an element v_d to the end of a GJB of size d will add 2^d elements x such that $\mathrm{MSB}(x) = \mathrm{MSB}(v_d)$. □

The last point of Lemma 3 allows a significant speed up of the search for such GJBs. To describe it, we introduce the following concept.

Definition 6 (MSB spectrum)**.** Let $\mathcal{S}$ be a set of elements in $\mathbb{F}_2^n$. The *MSB spectrum of* $\mathcal{S}$ is the sequence $\{\mathrm{N}_i(\mathcal{S})\}_{0 \leq i < n}$ such that

$$\mathrm{N}_i(\mathcal{S}) = \#\left\{x \in \mathcal{S}, \mathrm{MSB}(x) = i\right\}\ .$$

Corollary 3 (MSB conditions)**.** *If a set $\mathcal{S}$ of elements from $\mathbb{F}_2^n$ contains a vector space of dimension d, then there must exist a strictly increasing sequence $\{m_j\}_{0 \leq j \leq d-1}$ of length d such that*

$$N_{m_j}(S) \geq 2^j\ .$$

4.3 Vector Extractions

We now present a class of functions called *extractions* which will play a crucial role in our algorithms. We also prove their most crucial properties.

Definition 7 (Extraction)**.** Let $a \neq 0$ be some element of $\mathbb{F}_2^n$. The *extraction of* a, denoted $\mathcal{X}_a$, is a function mapping a subset $\mathcal{S}$ of $\mathbb{F}_2^n$ to $\mathcal{X}_a(\mathcal{S})$, where $x \in \mathcal{X}_a(\mathcal{S})$ if and only if all of the following conditions are satisfied:

$$x \in \mathcal{S} \ , \quad (x \oplus a) \in \mathcal{S} \ , \quad a < x < (x \oplus a) \ .$$

In particular, $\mathcal{X}_a(\mathcal{S}) \subseteq \mathcal{S}$. Our algorithm will iterate such extractions to construct smaller and smaller sets without loosing any GJBs. This process is motivated by the following theorem.

Theorem 6. *Let $\{v_0, ..., v_{i-1}\}$ be elements of some subset $\mathcal{S}$ of $\mathbb{F}_2^n$ such that $0 \in \mathcal{S}$ and such that $v_{j+1} \in (\mathcal{X}_{v_j} \circ ... \circ \mathcal{X}_{v_0})(\mathcal{S})$ for all $j < i$. Then it holds that $v_i \in (\mathcal{X}_{v_{i-1}} \circ ... \circ \mathcal{X}_{v_0})(\mathcal{S})$ if and only if $\langle v_0, ..., v_i \rangle \subseteq \mathcal{S}$ and $\{v_0, ..., v_i\}$ is the GJB of this vector space.*

Proof. In order to prove the theorem, we proceed by induction over i using the validity of the theorem over bases of size i as our induction hypothesis. At step i, we assume that $v_0, ..., v_i$ are elements of $\mathcal{S}$ and that $v_{j+1} \in (\mathcal{X}_{v_j} \circ ... \circ \mathcal{X}_{v_0})(\mathcal{S})$ for all $j < i$.

Initialization i = 1. By definition of vector extraction, $v_1 \in \mathcal{X}_{v_0}(\mathcal{S})$ if and only if $v_1 \in \mathcal{S}$, and $v_0 \oplus v_1 \in \mathcal{S}$, $v_0 < v_1 < v_0 \oplus v_1$. As we assume $0, v_0 \in \mathcal{S}$, this is equivalent to $\{0, v_0, v_1, v_0 \oplus v_1\} = \langle v_0, v_1 \rangle$ being contained in $\mathcal{S}$ and to $\{v_0, v_1\}$ being a GJB.

Inductive Step i > 1 Let $v_i \in (\mathcal{X}_{v_{i-1}} \circ ... \circ \mathcal{X}_{v_0})(\mathcal{S})$. From the induction hypothesis, we have that $\{v_0, ..., v_{i-1}\}$ is a GJB. Using the second point of Lemma 3, we have that its extension $\{v_0, ..., v_i\}$ is a GJB if and only if $v_i[\mathrm{MSB}(v_j)] = 0$ (which is equivalent to $v_i < v_i \oplus v_j$) for all $0 \leq j < i$ and $\mathrm{MSB}(v_i) > \mathrm{MSB}(v_{i-1})$.

By definition of $\mathcal{X}_{v_j}$, we have that $v_i < v_i \oplus v_j$ for all j such that $0 \leq j < i$, so $\{v_0, ..., v_i\}$ is a GJB if and only if $\mathrm{MSB}(v_i) > \mathrm{MSB}(v_{i-1})$. We have $v_{i-1} < v_i < v_i \oplus v_{i-1}$, which implies in particular $v_{i-1} < v_i \oplus v_{i-1}$, so that $v_i[\mathrm{MSB}(v_{i-1})] = 0$. Thus, $v_i > v_{i-1}$ holds if and only if $\mathrm{MSB}(v_i) > \mathrm{MSB}(v_{i-1})$. □

□

Corollary 4. *If $\{e_0, ..., e_{d-1}\}$ is the GJB of a vector space V such that $V \subseteq \mathcal{S} \subseteq \mathbb{F}_2^n$ then, for all $0 < j \leq d-1$, we have*

$$\langle e_j, e_{j+1}, ..., e_{d-1} \rangle \ \subseteq \ \left(\mathcal{X}_{e_{j-1}} \circ ... \mathcal{X}_{e_1} \circ \mathcal{X}_{e_0}\right)(\mathcal{S}) \ .$$

Evaluating $\mathcal{X}_a$ imposes a priori to look whether $x \oplus a$ belongs in $\mathcal{S}$ for all $x \in \mathcal{S}$ such that $x < x \oplus a$. This verification can be implemented efficiently using a binary search when $\mathcal{S}$ is sorted. We can make it even more efficient using the following lemma.

Lemma 4. *Let $\mathcal{S}$ be a set of elements in $\mathbb{F}_2^n$ and let $a \in \mathcal{S}$. Then we have*

$$\mathcal{X}_a(\mathcal{S}) \ = \ \bigcup_{i=MSB(a)+1}^{n} \mathcal{X}_a\left(\{x \in \mathcal{S}, MSB(x) = i\}\right)$$

4.4 Bigger MSB Condition

The following lemma provides a necessary condition for some $e_0 \in \mathcal{S}$ to be the first element of a GJB of size d.

Lemma 5 (Bigger MSB condition)**.** *If e_0 is the first element in a GJB of size d of elements of a set $\mathcal{S}$ of elements in $\mathbb{F}_2^n$, then $\mathcal{S}'$ defined as*

$$\mathcal{S}' = \{x \in \mathcal{S}, MSB(x) > MSB(e_0)\}$$

must satisfy the MSB condition of Corollary 3 for dimension $d-1$, i.e. there is a strictly increasing sequence $\{m_j\}$ of length $d-1$ such that

$$\#\{x \in \mathcal{S}, MSB(x) = m_j\} > 2^j \ .$$

This lemma provides an efficient filter to know whether x can be the start of a GJB of size d which depends only on the MSB of x, so that it does not need to be evaluated for all $x \in \mathcal{S}$ but only once for each subset of $\mathcal{S}$ with a given MSB.

4.5 Vector Space Extraction Algorithm

Algorithm 1 GJBExtraction algorithm.

```
1:  function GJBExtraction(S, d)
2:      L ← {}
3:      for all a ∈ φ_d(S) do
4:          s_a ← X_a(S)
5:          if |s_a| ≥ 2^{d-1} − 1 then
6:              L' ← GJBExtraction(s_a, max(d − 1, 0))
7:              for all B ∈ L' do
8:                  Add the GJB ({a} ∪ B) to L
9:              end for
10:         end if
11:     end for
12:     return L
13: end function
```

If we let ϕ_d be the identity then we can directly deduce from Theorem 6 and Corollary 4 that GJBExtraction (as described in Algorithm 1) returns the unique GJBs of each and every vector space of dimension at least equal to d that is included in $\mathcal{S}$.

This algorithm can be seen as a tree search. The role of ϕ_d is then to cut branches as early as possible by allowing us to ignore elements that cannot possibly be the first element of a base of size d by implementing the Bigger MSB Condition of Lemma 5:

$$a \in \phi_d(\mathcal{S}) \ \text{ if and only if } \ \exists \{m_j\}_{0 \le j < d}, \begin{cases} m_{j+1} > m_j > \mathrm{MSB}(a) \ , \\ \#\{x \in \mathcal{S}, \mathrm{MSB}(x) = m_j\} > 2^j \ . \end{cases}$$

Note that we only need to try and build such a sequence of increasing m_j once for each value of $\mathrm{MSB}(x)$ for $x \in \mathcal{S}$. It is possible to check for the existence of such a sequence in a time proportional to $|\mathcal{S}|$.

5 How "Structured" is a Random S-box?

In previous sections, we have established the probability that a random S-box has given differential, linear or boomerang properties, and we have quantified the probability that it has some *specific* structures. In this section, we tackle a much more general question:

What is the probability that a random S-box has *any* structure?

In order to answer this question, we first need to define what we mean by *structure* in this case. To this end, we will build upon the concept of *Kolmogorov complexity* of a string to bound the complexity of a function. Like in Section 2, our aim is to measure how far the properties of an S-box are from those that expected of a random S-box. However, we will not rely on statistical arguments but only on the pigeon principle.

We introduce the key concept behind our analysis (the *Kolmogorov anomaly of an S-box*) in Section 5.1. We then use it in Section 5.2 to establish that the number of S-boxes with as high a structure as π is of negligible size.

5.1 The Kolmogorov Anomaly of an S-box

The Kolmogorov complexity of a string is the length of the smallest program generating this string. Since the LUT of an S-box is a string, it would be natural to use its Kolmogorov complexity as an estimation of the complexity of the S-box itself. However, we do not want to capture the complexity of its LUT so much as the complexity of the algorithm used to evaluate the function. Thus, we instead try to estimate the Kolmogorov complexity of the *implementation* of the function.

To derive information about how structured an S-box is, we need to compare the Kolmogorov complexity of its implementation with the size of $\mathfrak{S}_{2^n}$. Indeed, if this length is much smaller than $\log_2(|\mathfrak{S}_{2^n}|)$ then there are very few permutations with as short an implementation. In other words, it is an anomaly. As discussed in the introduction, Shannon used a similar argument to bound the complexity of the circuits implementing Boolean functions in 1949 [Sha49].

Yet, in order to do this comparison, it is necessary that we obtain a meaningful estimate of the Kolmogorov complexity of the implementation.

The choice of the language used to implement the permutation then plays a crucial role. We could simply define a language with a standard library containing a function that evaluates the permutation and obtain a minimal Kolmogorov complexity for the implementation. Nevertheless, this result would not give us any useful information. To solve this problem, we propose to use two sets of languages: variants of the `C` language (portable `C11` and more relaxed K & R style) and compiled programs. As the code for micro-controllers is expected to have a small size, we chose to use the ARM dialect supported by the Cortex-M4 CPU as we had one at hand to test our implementation.

Definition 8 (Kolmogorov Anomaly of a Permutation for a Language)**.** Let $F \in \mathfrak{S}_{2^n}$ be a permutation such that there exists a program of bitlength $\ell_{\mathsf{L}}(\mathsf{L})$ in a given language L returning $F(x)$ when input x. The *Kolmogorov anomaly of F for the language* L is

$$\mathsf{A}^{\mathsf{K}}_{\mathsf{L}}(F) = \log_2(2^n!) - \ell_{\mathsf{L}}(F) - 1 \ .$$

The "-1" comes from the fact that there are at most $1 + 2 + 2^2 + ... + 2^t = 2^{t+1} - 1$ programs with length at most t. The *Kolmogorov anomaly* is then an anomaly in our sense.

5.2 Application to the Russian S-box

Let us estimate the Kolmogorov anomaly for π using first `C` as the language and then actual machine code.

General Approach. In all cases, our approach is based on the TKlog structure of π which we recalled in Equation (3). However, instead of implementing finite field arithmetic, we build a table `s` such that $\mathtt{s}[j] = \alpha^{17s(j)}$. Furthermore, the discrete logarithm that is implicitly used can be implemented in a very compact way. In the case of π, the "logarithm"

of 0 is set to 0 and that of 1 to 255; the other values are as expected. The following code snippet evaluates this function on `x` by setting the value of `l` accordingly.

```
  int l=0, a=2; while((x) && (l++, a != x)) { a=(a << 1) ^ (a >> 7)*0x11d ; }
```

Indeed, if $x = 0$ then this loop is not entered and `l` is indeed set to 0. If not, then we multiply a value `a` by the generator of the multiplicative subfield using its representation as a Galois LFSR until its value if equal to the input `x`. At each iteration, `l` is incremented. We further save space by replacing the Boolean value `a!=x` with `a^x` as both are equal to 0 if and only if `a==x`. We can also write `0x11d` in decimal to save more space, i.e. `285`.

We then set `i=l/17, j=l%17` and need to consider two cases: if `j==0` then the output of π is $\kappa(16 - i)$, otherwise it is $\kappa(16 - i) \oplus \mathtt{s}[j]$. The way in which we evaluate κ changes depending on the language we target.

A Portable C11 Program. Our shortest implementation in portable `C11` follows. In this case, we have implemented κ as $x \mapsto M(x) \oplus \kappa(0)$ where M is a linear function implemented using a macro.

```
#define M(x)(x&8^42)*6^(x&4^2*(x)&6)*9^x&2
typedef unsigned char u;
u p(u x){
  u s[]={1,221,146,79,147,153,11,68,214,215,78,220,152,10,69},a=2,l=0;
  while(x&&(l++,a^x))a=2*a^a/128*285;
  return(l%17?M(16-l%17)^s[l/17]:M(16-l/17))^252;
}
```

It contains 227 characters after useless spaces and new lines are removed. If the C standard does not specify a character source encoding, the characters expected to be supported roughly correspond to the printable ASCII characters. Thus, we counted that 7 bits are required to encode each character. Hence, the total length of this program is $\ell_{\mathsf{C11\ strict}}(\pi) = 1589$ and $\mathsf{A}^{\mathsf{K}}_{\mathsf{C11\ strict}}(\pi) = 94$.

A (Marginally) Less Portable C Program. We can make a much shorter program if we use some features which are not part of the `C` standard but which are used by most compilers in practice.

We consider that the character set used is the ASCII because it allows us to use strings for our `char` array. Moreover, we assume that a `char` is only 8-bit long, which implies that a `signed int` is big enough to contain the range of an `unsigned char`. It allows us to use `int` as argument and return type.

In order to implement κ, we precompute $x \mapsto \kappa(16 - x)$ and store it in a table `k`. In order to initialize both `k` and `s`, we use character strings as they provide are a very compact encoding as long as the values inside are printable ASCII characters other than "`\`". The non-printable character `0xNN` is represented as "`\xNN`".

Hence, our approach is to find two constants to xor to the elements of `k` and `s` such that the amount of printable characters is maximal. As the amplitude of `k` is small, all of them can be printable if we choose well. This is not the case for `s`, the best we can achieve is 8 printable characters amongst the 15. Still, even with these two constraints, many values for the constants remain possible.

We then add the additional constraint that the xor of the two constant shall be $237 = 252 \oplus 17$. We can save space by setting a variable to 17. Finally, we avoid having a character that corresponds to a hexadecimal number following a non-printable character so as to avoid parsing problems. In the end, we choose constants 188 for `k` and 173 for `s`, leading to the strings `@‘rFTDVbpPBvdtfR@` and `\xacp?\xe2>4\xa6\xe9{z\xe3q5\xa7\xe8`. We store both in a unique array `t` and obtain the following implementation.

```
int p(int x){unsigned char
*t="@‘rFTDVbpPBvdtfR@\xacp?\xe2>4\xa6\xe9{z\xe3q5\xa7\xe8",a=2,l=0,b=17;
while(x&&(l++,a^x))a=2*a^a/128*29;return l%b?t[l%b]^t[b+l/b]^b:t[l/b]^188;}
```

After removing the newlines we obtain 173 characters. Each is an ASCII character so that they can all be encoded on a 7-bit word. We deduce that $\ell_{\mathsf{C+ASCII}}(\pi) = 1211$ and $\mathsf{A}^{\mathsf{K}}_{\mathsf{C+ASCII}}(\pi) = 472$.

Type declarations used to be optional in `C89`, and defaulted to `int`. This is no longer in the standard, but many compilers (in particular, `gcc` and `clang`) still support it by default.

Hence, we can further remove the two `int` in the prototype if we do not seek compliance with the `C11` standard. This would further save 8 characters, bringing the total down to 165, i.e. an anomaly of 528.

Binary Code. The previous programs were small in number of characters. However, to reflect the inner complexity (or lack thereof) of the executed program, this approach may not be optimal. In particular, data encoding is denser in binary, type annotations are not in the compiled code and some short successions of C instructions may compile to a lengthy assembly.

We then searched for the shortest *program* we could achieve. We built upon the assembly of a variant of the following C program.

```
int p(int x){
   static unsigned char
    s[]={1,221,146,79,147,153,11,68,214,215,78,220,152,10,69},
    k[]={252,220,206,250,232,248,234,222,204,236,254,202,216,200,218,238,252};
    unsigned char a=2;
    int l=1;
  if(!x) return 252;
  for(;a!=x;l++) a = (a<<1) ^ (a&128 ? 29 : 0);
  unsigned int i=l%17, j=l/17;
  return (i) ? k[i]^s[j] : k[j];
}
```

This program was compiled with `-Os`, both for generic `x86-64` and for the Cortex-M4. The produced assembly part is in Appendix F.2. The Intel program contains 120 bytes, including some padding, while the Cortex-M4 program is 102 bytes long.

We then decided to hand-optimize the Cortex-M4 program. The Intel instruction set is much more complicated, hence we preferred to focus on this simpler language. Our approach is detailed in Appendix F.1 and the assembly code itself is given in Appendix F.2. The shortest program we obtained for the Cortex-M4 is 80 bytes long.

In the end, we have $\ell_{\mathsf{generic\ x86-64}}(\pi) = 120 \times 8 = 960$ bits and $\ell_{\mathsf{Cortex-M4}}(\pi) = 80 \times 8 = 639$, so that $\mathsf{A}^{\mathsf{K}}_{\mathsf{generic\ x86-64}}(\pi) = 723$ and $\mathsf{A}^{\mathsf{K}}_{\mathsf{Cortex-M4}}(\pi) = 1043$.

6 Conclusion

Let us apply our results to π. Although its designers claim to have obtained it by generating permutations uniformly at random and then filtering those according to their cryptographic properties, we found that it has very high anomalies which we summarize in Table 3.

Table 3: Some of the anomalies of π.

Statistical			Structural		Kolmogorov	
Diff.	Linear	Boom.	TU_4	TKlog	C+ASCII	ARM code size
80.6†	34.4	14.2	201.1	1601.5	472	1043

† This anomaly might be overestimated (see Section 2.4).

As we can see, the set of S-boxes with as strong a structure as the TKlog found in π is astonishingly small. In other words, the probability that a random S-box has *any*

structure that is as simple as that of π is negligible. Consequently, the claim of [Per19] that the structure of π was deliberately inserted by its designers is correct. On the other hand, the fact that the designers of π doubled down on their claims of randomness [YH19] instead of acknowledging their use of a structure in light of [Per19] is sufficient for us to urge practitioners *not to use Streebog or Kuznyechik*.

We finally list some open problems that we have identified while working on this paper.

Open Problem 1. How can we better estimate the differential anomaly?

Open Problem 2. Why are there so many vector spaces in $\mathcal{Z}_F$ when F is a 3-round Feistel network of $\mathfrak{S}_{2^8}$?

7 Acknowledgement

We thank Jérémy Jean for shepherding the ASIACRYPT'19 version of this paper [BPT19a]. We also thank Florian Wartelle for fruitful discussions about vector space search, and Anne Canteaut for proofreading a first draft of this paper. The work of Xavier Bonnetain receives funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 714294 – acronym QUASYModo). The work of Shizhu Tian is supported by the National Science Foundation of China (No. 61772517, 61772516).

References

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[BC13] Christina Boura and Anne Canteaut. On the influence of the algebraic degree of f^{-1} on the algebraic degree of $g \circ f$. *IEEE Transactions on Information Theory*, 59(1):691–702, Jan 2013.

[BCC11] Céline Blondeau, Anne Canteaut, and Pascale Charpin. Differential properties of $x \mapsto x^{2^t-1}$. *IEEE Transactions on Information Theory*, 57(12):8127–8137, 2011.

[BDBP03] Alex Biryukov, Christophe De Canniére, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 33–50. Springer, Heidelberg, May 2003.

[BDMW10] K. A. Browning, J.F. Dillon, M. T. McQuistan, and A. J. Wolfe. An APN permutation in dimension six. In *Post-proceedings of the 9-th International Conference on Finite Fields and Their Applications*, volume 518, pages 33–42. American Mathematical Society, 2010.

[BLP16] Alex Biryukov, Gaëtan Leurent, and Léo Perrin. Cryptanalysis of Feistel networks with secret round functions. In Orr Dunkelman and Liam Keliher, editors, *SAC 2015*, volume 9566 of *LNCS*, pages 102–121. Springer, Heidelberg, August 2016.

[BP15] Alex Biryukov and Léo Perrin. On reverse-engineering S-boxes with hidden design criteria or structure. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 116–140. Springer, Heidelberg, August 2015.

[BPT19a] Xavier Bonnetain, Léo Perrin, and Shizhu Tian. Anomalies and vector space search: Tools for S-box analysis. In *Advances in Cryptology – ASIACRYPT 2019*, Cham, 2019. Springer International Publishing.

[BPT19b] Christina Boura, Léo Perrin, and Shizhu Tian. Boomerang uniformity of popular S-box constructions. In *WCC 2019: The Eleventh International Workshop on Coding and Cryptography*, 2019.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-engineering the S-box of streebog, kuznyechik and STRIBOBr1. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 372–402. Springer, Heidelberg, May 2016.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of Feal and N-hash. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 1–16. Springer, Heidelberg, April 1991.

[BS01] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 394–405. Springer, Heidelberg, May 2001.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[CDDL06] Anne Canteaut, Magnus Daum, Hans Dobbertin, and Gregor Leander. Finding nonnormal bent functions. *Discrete Applied Mathematics*, 154(2):202 – 218, 2006. Coding and Cryptography.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, Heidelberg, April / May 2018.

[CP19] Anne Canteaut and Léo Perrin. On CCZ-equivalence, extended-affine equivalence, and function twisting. *Finite Fields and Their Applications*, 56:209–246, 2019.

[DES77] Data encryption standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977.

[Dev17] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.5.1)*, 2017. `http://www.sagemath.org`.

[Din18] Itai Dinur. An improved affine equivalence algorithm for random permutations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 413–442. Springer, Heidelberg, April / May 2018.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *Journal of Mathematical Cryptology*, 1(3):221–242, 2007.

[Dt08] Whitfield Diffie and George Ledin (translators). SMS4 encryption algorithm for wireless networks. Cryptology ePrint Archive, Report 2008/329, 2008. `http://eprint.iacr.org/2008/329`.

[Fed12] Federal Agency on Technical Regulation and Metrology. Information technology – data security: Hash function. English version available at http://wwwold.tc26.ru/en/standard/gost/GOST_R_34_11-2012_eng.pdf, 2012.

[Fed15] Federal Agency on Technical Regulation and Metrology. Information technology – data security: Block ciphers. English version available at http://wwwold.tc26.ru/en/standard/gost/GOST_R_34_12_2015_ENG.pdf, 2015.

[Hel94] Tor Helleseth, editor. *EUROCRYPT'93*, volume 765 of *LNCS*. Springer, Heidelberg, May 1994.

[KKO13] Oleksandr Kazymyrov, Valentyna Kazymyrova, and Roman Oliynykov. A method for generation of high-nonlinear s-boxes based on gradient descent. Cryptology ePrint Archive, Report 2013/578, 2013. http://eprint.iacr.org/2013/578.

[LJH+07] Fen Liu, Wen Ji, Lei Hu, Jintai Ding, Shuwang Lv, Andrei Pyshkin, and Ralf-Philipp Weinmann. Analysis of the SMS4 block cipher. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *ACISP 07*, volume 4586 of *LNCS*, pages 158–170. Springer, Heidelberg, July 2007.

[LQSL19] Kangquan Li, Longjiang Qu, Bing Sun, and Chao Li. New results about the boomerang uniformity of permutation polynomials. *CoRR*, abs/1901.10999, 2019.

[Lup73] O. B. Lupanov. *On Networks Consisting of Functional Elements with Delays*, pages 43–83. Springer US, New York, NY, 1973.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Helleseth [Hel94], pages 386–397.

[MDFK18] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. *Journal of Cryptology*, 31(3):845–884, July 2018.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Helleseth [Hel94], pages 55–64.

[O'C94] Luke O'Connor. On the distribution of characteristics in bijective mappings. In Helleseth [Hel94], pages 360–370.

[O'C95] Luke O'Connor. Properties of linear approximation tables. In Bart Preneel, editor, *FSE'94*, volume 1008 of *LNCS*, pages 131–136. Springer, Heidelberg, December 1995.

[Per19] Léo Perrin. Partitions in the S-box of Streebog and Kuznyechik. *IACR Trans. Symm. Cryptol.*, 2019(1):302–329, 2019.

[PU16] Léo Perrin and Aleksei Udovenko. Exponential s-boxes: a link between the s-boxes of BelT and Kuznyechik/Streebog. *IACR Trans. Symm. Cryptol.*, 2016(2):99–124, 2016. http://tosc.iacr.org/index.php/ToSC/article/view/567.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

[PW17] Léo Perrin and Friedrich Wiemer. S-Boxes used in cryptographic schemes. Available online at https://git.sagemath.org/sage.git/tree/src/sage/crypto/sboxes.py, 2017.

[Sch14] Felix Schuster. Reverse engineering of chiasmus from gstool. Presentation at the HGI-Kolloquium. Slides available at https://prezi.com/ehrz4krw2z0d/hgi-chm/, January 2014.

[Sha49] C. E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, Jan 1949.

[SM18] Vasily Shishkin and Grigory Marshalko. A Memo on Kuznyechik S-Box. ISO/IEC JTC 1/SC 27/WG 2 Officer's Contribution N1804, https://cdn.virgilsecurity.com/assets/docs/memo-on-kuznyechik-s-box.pdf, September 2018.

[STW13] Jan Schejbal, Erik Tews, and Julian Wälde. Reverse engineering of chiasmus from gstool. Presentation at the Chaos Computer Club (CCC)., 2013.

[TG92] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 172–181. Springer, Heidelberg, August 1992.

[U.S98] U.S. Department Of Commerce/National Institute of Standards and Technology. Skipjack and KEA algorithms specifications, v2.0, 1998.

[Wag99] David Wagner. The boomerang attack. In Lars R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 156–170. Springer, Heidelberg, March 1999.

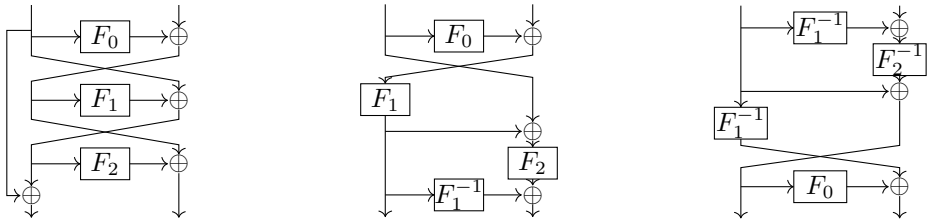
[YH19] Hirotaka Yoshida and Jonathan Hammell. Meeting report for the discussion on Kuznyechik and Streebog. ISO/IEC JTC 1/SC 27/WG 2 Meeting Report N2016, https://cdn.virgilsecurity.com/assets/docs/meeting-report-for-the-discussion-on-kuznyechik-and-streebog.pdf, April 2019.

A Proof of the Vector Spaces for Feistel Networks

Our reasoning will rely on two results from [CP19]. First, if F is a function mapping $\mathbb{F}_2^n$ to itself then $\{(x,0), x \in \mathbb{F}_2^n\}$ has to be in its Walsh zeroes. Second, (Lemma 2 of [CP19]) if the functions F and G are such that

$$\{(x, G(x)), x \in \mathbb{F}_2^n\} = \mathcal{L}\big(\{(x, F(x)), x \in \mathbb{F}_2^n\}\big)$$

then $\mathcal{Z}_G = (\mathcal{L}^T)^{-1}(\mathcal{Z}_F)$.



(a) P. (b) P. (c) P^{-1}.

Figure 3: A permutation P obtained by adding a linear feedforward to a 3-round Feistel network.

Proof of Theorem 5. If $F \in \mathfrak{S}_{2^n}$ is a Feistel network then it is a well-defined permutation, meaning that the spaces in the first category must be in $\mathcal{Z}_F$. More generally, as the inverse of 3-round Feistel network has the same type of structure, if said structure imposes a space in $\mathcal{Z}_F$ then it also imposes the symmetric space where the coordinates are inverted.

The well-known integral distinguisher against 3-round Feistel networks implies the presence of a $\mathsf{TU}_{n/2}$-decomposition composed with a branch swap in its input. Thus, the space of the second category has to be in $\mathcal{Z}_F$.

The first space is in the third category in $\mathcal{Z}_F$ because $F + L$ is a permutation when L is the projection $L(x,y) = (x,0)$ (see Figure 3). Indeed, this space is

$$\left(\begin{bmatrix} I & 0 \\ L & I \end{bmatrix}^T\right)^{-1} \begin{pmatrix} x||y \\ 0||0 \end{pmatrix} = \begin{bmatrix} I & L^T \\ 0 & I \end{bmatrix} \begin{pmatrix} x||y \\ 0||0 \end{pmatrix} = \begin{bmatrix} I & L \\ 0 & I \end{bmatrix} \begin{pmatrix} x||y \\ 0||0 \end{pmatrix} = \begin{pmatrix} x||y \\ x||0 \end{pmatrix}$$

where we used the fact that $L^T = L$. The second space in this category is its symmetric.

Finally, let $F_2 \in \mathfrak{S}_{2^{n/2}}$ and let L' be the projection such that $L'(x,y) = (0,y)$. We have that the right hand side of $F(x,y)$ is equal to $F_0(x) + y + F_2\left(x + F_1\left(y + F_0(x)\right)\right)$, so that the right hand side of $F + L'$ is a permutation of y for any x because $(F + L')(x,y) = F_2\left(F_1\left(y + F_0(x)\right) + x\right) + F_0(x)$. It means that $\mathcal{Z}_{F+L'}$ contains $\{(x||0,0||y), (x,y) \in (\mathbb{F}_2^{n/2})^2\}$, and thus that $\mathcal{Z}_F$ contains

$$\left(\begin{bmatrix} I & 0 \\ L' & I \end{bmatrix}^T\right)^{-1} \begin{pmatrix} x||0 \\ 0||y \end{pmatrix} = \begin{bmatrix} I & (L')^T \\ 0 & I \end{bmatrix} \begin{pmatrix} x||0 \\ 0||y \end{pmatrix} = \begin{bmatrix} I & L' \\ 0 & I \end{bmatrix} \begin{pmatrix} x||0 \\ 0||y \end{pmatrix} = \begin{pmatrix} x||y \\ 0||y \end{pmatrix}$$

for all x, y, where we used the fact that $(L')^T = L'$. We deduce that the first space in the fourth category is in $\mathcal{Z}_F$. The second one is its symmetric. □

B Some Anomalies for 8-bit Permutations

Table 4: The anomalies associated to the maximum absolute value in each table.

$u(F)$	$\mathsf{A}(u(F))$	$\overline{\mathsf{A}}(u(F))$
4	1359.53	0.00
6	164.47	0.00
8	16.15	0.00
10	1.33	0.73
12	0.09	3.99
14	0.01	7.95
16	0.00	12.13
18	0.00	16.46
20	0.00	20.94
22	0.00	25.96

(a) DDT.

$\ell(F)$	$\mathsf{A}(\ell(F))$	$\overline{\mathsf{A}}(\ell(F))$
44	371.61	0.00
48	161.90	0.00
52	66.41	0.00
56	25.62	0.00
60	9.29	0.00
64	3.16	0.17
68	1.01	0.99
72	0.30	2.41
76	0.08	4.14
80	0.02	6.04
84	0.01	8.06
88	0.00	10.19
92	0.00	12.42
96	0.00	14.75
100	0.00	17.19

(b) LAT.

β_F	$\mathsf{A}(\beta_F)$	$\overline{\mathsf{A}}(\beta_F)$
10	442.66	0.00
12	123.52	0.00
14	33.89	0.00
16	8.43	0.00
18	2.05	0.40
20	0.46	1.86
22	0.10	3.87
24	0.02	6.09
26	0.00	8.38
28	0.00	10.75
30	0.00	13.15
32	0.00	15.60
34	0.00	18.09
36	0.00	20.63
38	0.00	23.20

(c) BCT.

C GJB Search Implementation and TU-decomposition

Our implementations of GJBExtraction and CanonicalExtraction are available online at

https://who.rocq.inria.fr/Leo.Perrin/code/tu_code.zip

They are written in C++ and use the standard library (`std::thread`) to handle multi-threading. We also provide a multi-threaded function returning the Walsh zeroes. Python bindings allow the use of these algorithms from higher level SAGE[6] [Dev17] scripts.

Since the core of our library is written in C++, it is necessary to compile it. To this end, we used `cmake`[7] to set up the compilation—it is therefore necessary to install this tool. We also use the `Boost.Python`[8] library to handle the interaction between C++ and Python. On Ubuntu[9], these correspond to the packages `cmake` and `libboost-python-dev` respectively. In order to compile it, use the following commands once `cmake` and `Boost.Python` have been installed.

```
cd <directory containing the unzipped supplementary material>
cd sboxU
cmake .
make
cd ..
```

At that point, you are ready to run the SAGE script `tu_decomposition.sage` which automatically returns the TU_t-decomposition of the Russian π. It recovers and prints the 8-bit binary permutations A and B as well as $T_y \in \mathfrak{S}_{2^4}$ and $U_x \in \mathfrak{S}_{2^4}$ that correspond to the TU_4-decomposition of this component. It then recomputes the lookup table of π using these subfunctions and checks whether it is identical to the original π.

This program also shows that π has only one TU-decomposition, namely the one found by Biryukov et al.

D Looking for Affine Spaces

Using GJBExtraction, we can build a similar algorithm returning all the *affine* spaces in a set of elements of $\mathbb{F}_2^n$. However, for an affine space, the GJB of the underlying vector space is not sufficient to uniquely define it. We also need to describe the offset in such a way that it is uniquely defined. The simplest approach was presented in [CDDL06].

Definition 9 (Canonical representation of an affine space)**.** An affine subspace of dimension d can be represented as $c \oplus \langle e_0, ..., e_{d-1} \rangle$ where $\{e_0, ..., e_{d-1}\}$ is the GJB of its span and where c verifies $c < c \oplus e_j$ for all $j \in \{0, ..., d-1\}$. It is its *canonical representation* and it is unique.

We thus build an algorithm which looks for the canonical representation of each and every affine space contained in a set $\mathcal{S}$ of elements in $\mathbb{F}_2^n$. It uses the following operation whose goal is explained by the next lemma.

Definition 10 (Affine Preprocessing)**.** We call *affine preprocessing for* $c \in \mathbb{F}_2^n$ the function ψ_c mapping sets of elements of $\mathbb{F}_2^n$ to other such sets which is such that $x \in \psi_c(\mathcal{S})$ if and only if $x \oplus c \in \mathcal{S}$ and $c < x \oplus c$ (i.e. $c[\mathrm{MSB}(x)] = 0$).

[6] https://sagemath.org
[7] https://cmake.org/
[8] https://www.boost.org/doc/libs/1_63_0/libs/python/doc/html/index.html
[9] We have only tested our library and the script described below on a machine running Ubuntu.

Lemma 6. *Let $\mathcal{S}$ be a set of elements of $\mathbb{F}_2^n$. Then $\{e_0, ..., e_{d-1}\}$ is the GJB of a space in $\psi_c(\mathcal{S})$ if and only if $c \oplus \{e_0, ..., e_{d-1}\}$ is the canonical representation of an affine space contained in $\mathcal{S}$.*

Proof. This lemma is an equivalence so we prove each of its directions separately.

$\Rightarrow$ Suppose that $\{e_0, ..., e_{d-1}\}$ is the GJB of a vector space $V \subseteq \psi_c(\mathcal{S})$. As this imposes that $x \oplus c \in \mathcal{S}$ for all $x \in V$, we deduce that $c \oplus V \subseteq \mathcal{S}$. Furthermore, as $c < (c \oplus x)$ for all $x \in V$, it holds in particular for all e_j. Thus, $c \oplus \langle e_0, ..., e_{d-1} \rangle$ satisfies all the conditions to be a canonical representation and is such that the corresponding affine space is contained in $\mathcal{S}$.

$\Leftarrow$ Suppose that $c \oplus \langle e_0, ..., e_{d-1} \rangle$ is the canonical representation of an affine space of $\mathcal{S}$. Then, by definition, we have that $c < c \oplus e_j$ for all j. Since all elements of $V = \langle e_0, ..., e_{d-1} \rangle$ share their MSB with one of the vectors in $\{e_0, ..., e_{d-1}\}$, this property holds for all $x \in V$. We deduce that, for all $x \in V$, $c < x \oplus c$. As $c \oplus V \subseteq \mathcal{S}$, we obviously have that $x \oplus c \in \mathcal{S}$ as well. Besides, as $c \oplus \langle e_0, ..., e_{d-1} \rangle$ is a canonical representation, $\{e_0, ..., e_{d-1}\}$ has to be minimal.

We conclude that if $c \oplus \{e_0, ..., e_{d-1}\}$ is the canonical representation of an affine space embedded in $\mathcal{S}$ then $\{e_0, ..., e_{d-1}\}$ is a GJB and $\langle e_0, ..., e_{d-1} \rangle \subseteq \psi_c(\mathcal{S})$.

□

Using Lemma 6, we easily derive that Algorithm 2 returns the unique canonical representation of each and every affine space of dimension at least d contained in a set $\mathcal{S}$ of elements of $\mathbb{F}_2^n$. We use $\mathrm{hw}(x)$ to denote the Hamming weight of x.

Algorithm 2 CanonicalExtraction algorithm.

```
1:  function GJBExtraction(S, d)
2:      L ← {}
3:      for all c ∈ S such that hw(c) + d ≤ n do
4:          s_c ← ψ_c(S)
5:          if |s_c| ≥ 2^d − 1 then
6:              L' ← GJBExtraction(s_c, d))
7:              for all B ∈ L' do
8:                  Add the canonical representation (c ⊕ B) to L
9:              end for
10:         end if
11:     end for
12:     return L
13: end function
```

In practice, we use only offsets c with Hamming weight under $n - d$ as they need at least d zeroes in their binary representation in order to "fit" d different MSBs.

E Generating the S-box of Chiasmus

```
#!/usr/bin/sage
from sage.all import *

A = Matrix(GF(2), 8, 8, [
    1,1,0,1,1,1,1,0, 1,0,0,0,1,1,0,0,
    0,1,0,1,0,1,1,0, 0,0,1,1,0,1,0,1,
    1,1,1,1,1,0,0,0, 0,0,1,1,0,1,1,1,
    0,1,0,1,1,0,0,0, 1,1,1,0,0,0,0,0,
])

B = Matrix(GF(2), 8, 8, [
    1,1,0,1,1,0,0,0, 0,0,0,0,1,0,0,0,
    0,1,0,1,0,1,0,0, 1,1,0,1,0,1,1,0,
    0,0,1,1,0,1,0,0, 0,1,1,1,1,1,0,0,
    0,1,1,1,1,0,0,0, 1,1,0,0,1,0,1,1,
])

def apply_bin_mat(x, mat): # multiplication of an integer by a binary matrix
    n = mat.ncols()
    bin_x = [(x >> i) & 1 for i in reversed(xrange(0, n))]
    x = Matrix(GF(2), n, 1, bin_x)
    y = mat * x
    bin_y = y.T[0][:mat.nrows()]
    return sum(int(bin_y[i]) << (7-i) for i in xrange(0, 8))


def oplus(x, y): # needed because of inconstencies in how XOR is handled in sage
    return int(x).__xor__(int(y))


# generating the LUT of the multiplicative inverse
X = GF(2).polynomial_ring().gen()
F = GF(2**8, name="a", modulus=X**8+X**4+X**3+X**2+1)
mult_inv = [(F.fetch_int(x)**254).integer_representation()
            for x in xrange(0, 256)]

# composing the multiplicative inverse with the affine permutations
c_in, c_out = 0x8f, 0x59
s = [oplus(apply_bin_mat(mult_inv[apply_bin_mat(oplus(x, c_in), A)], B), c_out)
     for x in xrange(0, 256)]

# printing the result
print s
```

F Assembly code

F.1 Optimization Strategy

First, we optimized the discrete logarithm computation. We compute the iterated multiplication in the upper byte of the registers, which allows the left shift to put the uppermost bit of `a` in the carry flag, for free. Then, we suppress the specific case of the logarithm of 0 by initializing `l` at 0 and `a` at 1, and putting the test at the end of the loop. This allows for a simple conditional jump to the last part, instead of a dedicated return. Concretely, the discrete logarithm can be implemented as:

```
   r0 = x;         // Input
   r3 = 0;         // variable l
   if (r3 == 0) jump to END;
   r2 = 1 << 24;  // variable a is initialized to 1 in the upper byte
   r0 = r0 << 24; // Input shifted to the upper byte
LOOP:
   r2 = r2 << 1;  // Shifts and put the upper bit in the carry flag
   if (carry flag set to 1)
     r2 ^= 0x1d000000       // xor 0x1d in the upper byte
   r3 += 1                  // Increment l
   if (r2 != r0)
     jump to LOOP;
END:
```

Second, the lookup can be optimized. There is a quotient and a remainder of the division by 17 to compute, and then two different lookups to do. We first put the table `s` just after `k`, and see a lookup to `s` as a lookup to `k` shifted of 17. Then, we compute the different lookups as in the following pseudocode:

```
   r3 = l;           //Previously computed
   r0 = 17;
   r2 = r3/ro;       // Quotient
   r3 = r3 - r2*r0; // Remainder
   r1 = index of table k;
   if (r3 == 0) jump to L;
   r3 = r1[r3];      // fetches k[ l % 17 ]
   r1 += r0;         // Shifts the index of k to s
L:
   r0 = r1[r2]       // fetches either k[ l/17 ] or s[ l/17 ]
   r0 ^= r3          // is either k[ l/17 ] or s[ l/17 ]^k[ l % 17 ]
```

The assembly is presented in Appendix F.2. It corresponds to a complete 80 bytes long program, with 32 bytes of data and 20 instructions (16 16-bit instructions and 4 32-bit instructions). We can represent it in base64 as
`ACNIsU/wgHIABlIAKL+C8OhSATOCQvjRESCz+/DyAPsSMwOhC7HLXAFEiFxYQHBH/Nz O+uj46t7M7P7K2Mja7vwB3ZJPk5kLRNbXTtyYCkU=`.

Possible improvements. The 32-bit instructions correspond to the initialization of `r2`, the xoring of the modulo in the upper byte, and the division and remainder computation for the lookups. It may be possible to reduce the size of the discrete logarithm computation by finding a different encoding that allows to perform the multiplication with a right shift: in that case, we could have small numbers and a direct carry initialization. We unfortunately did not manage to find a representation that would both allow for a smaller discrete log computation and an easy conversion of the encoding of the input.

F.2 Assembly Code

This section contains the assembly generated by `x86_64-linux-gnu-gcc-7.3.0` and `arm-none-eabi-gcc-6.3.1` from the C code of π, and the shortest hand-optimized Cortex-M4 assembly we managed to find.

The code for generic `x86-64` architecture:

```
        .text
        .globl  p
        .type   p, @function
p:
.LFB0:
        .cfi_startproc
        testl   %edi, %edi
        movl    $252, %eax
        je      .L2
        movl    $1, %eax
        movb    $2, %dl
.L3:
        movzbl  %dl, %ecx
        cmpl    %ecx, %edi
        je      .L11
        sarb    $7, %dl
        addl    %ecx, %ecx
        incl    %eax
        andl    $29, %edx
        xorl    %ecx, %edx
        jmp     .L3
.L11:
        cltd
        movl    $17, %ecx
        idivl   %ecx
        testl   %edx, %edx
        movl    %eax, %eax
        je      .L6
        movl    %edx, %edx
        movb    k.1834(%rdx), %dl
        xorb    s.1833(%rax), %dl
        movzbl  %dl, %eax
        ret
.L6:
        movzbl  k.1834(%rax), %eax
.L2:
        ret
        .cfi_endproc

        .section        .rodata
        .align 8
s.1833:
        .byte   1
        .byte   -35
        .byte   -110
        .byte   79
        .byte   -109
        .byte   -103
        .byte   11
        .byte   68
        .byte   -42
        .byte   -41
        .byte   78
        .byte   -36
        .byte   -104
        .byte   10
        .byte   69
        .align 16
k.1834:

        .byte   -4
        .byte   -36
        .byte   -50
        .byte   -6
        .byte   -24
        .byte   -8
        .byte   -22
        .byte   -34
        .byte   -52
        .byte   -20
        .byte   -2
        .byte   -54
        .byte   -40
        .byte   -56
        .byte   -38
        .byte   -18
        .byte   -4
```

The code for Cortex-M4:

```
        .syntax unified
        .text
        .align  1
        .global p
p:
        cbz     r0, .L7
        movs    r3, #1
        movs    r2, #2
.L3:
        cmp     r2, r0
        beq     .L10
        lsls    r1, r2, #1
        tst     r2, #128
        uxtb    r1, r1
        ite     ne
        movne   r2, #29
        moveq   r2, #0
        eors    r2, r2, r1
        adds    r3, r3, #1
        b       .L3
.L10:
        movs    r2, #17
        sdiv    r2, r3, r2
        add     r1, r2, r2, lsl #4
        subs    r3, r3, r1
        ldr     r1, .L11
        ittte   ne
        addne   r3, r3, r1
        ldrbne  r0, [r1, r2]    @ zero_extendqisi2
        ldrbne  r3, [r3, #15]   @ zero_extendqisi2
        addeq   r2, r2, r1
        ite     ne
        eorne   r0, r0, r3
        ldrbeq  r0, [r2, #15]   @ zero_extendqisi2
        bx      lr
.L7:
        movs    r0, #252
        bx      lr
.L12:
        .align  2
.L11:
        .word   .LANCHOR0
        .size   p, .-p
        .section        .rodata
        .set    .LANCHOR0,. + 0
        .type   s.4078, %object
        .size   s.4078, 15
s.4078:
        .byte   1
        .byte   -35
        .byte   -110
        .byte   79
        .byte   -109
        .byte   -103
        .byte   11
        .byte   68
        .byte   -42
        .byte   -41
        .byte   78
        .byte   -36
        .byte   -104
        .byte   10
        .byte   69
        .type   k.4079, %object
        .size   k.4079, 17
k.4079:
        .byte   -4
        .byte   -36
        .byte   -50
        .byte   -6
        .byte   -24
        .byte   -8
        .byte   -22
        .byte   -34
        .byte   -52
        .byte   -20
        .byte   -2
        .byte   -54
        .byte   -40
        .byte   -56
        .byte   -38
        .byte   -18
        .byte   -4
```

The hand-optimized Cortex-M4 assembly:

```
        .syntax unified
        .text
        .align  4

        .global p
p:
        movs    r3, #0
        cbz     r0, .L2
        mov     r2, #0x1000000
        lsls    r0, #24
.L3:
        lsls    r2, r2, #1
        it      cs
        eorcs   r2, r2, #0x1d000000
        adds    r3, r3, #1
        cmp     r2, r0
        bne     .L3
.L2:
        movs    r0, #17
        udiv    r2, r3, r0
        mls     r3, r0, r2, r3
        adr     r1, .table_k
        cbz     r3, .LL
        ldrb    r3, [r1, r3]
        add     r1, r0
.LL:
        ldrb    r0, [r1, r2]
        eors    r0, r0, r3
        bx      lr
        .align  2

.table_k:
        .byte   -4
        .byte   -36
        .byte   -50
        .byte   -6
        .byte   -24
        .byte   -8
        .byte   -22
        .byte   -34
        .byte   -52
        .byte   -20
        .byte   -2
        .byte   -54
        .byte   -40
        .byte   -56
        .byte   -38
        .byte   -18
        .byte   -4
.table_s:
        .byte   1
        .byte   -35
        .byte   -110
        .byte   79
        .byte   -109
        .byte   -103
        .byte   11
        .byte   68
        .byte   -42
        .byte   -41
        .byte   78
        .byte   -36
        .byte   -104
        .byte   10
        .byte   69
```