Finding Desirable Substitution Box with SASQUATCH

Manas Wadhwa; Indian Institute of Technology, Bhilai, India; Email: manasw@iitbhilai.ac.in
Anubhab Baksi; Nanyang Technological University, Singapore; Email: anubhab001@e.ntu.edu.sg
Kai Hu; Nanyang Technological University, Singapore; Email: kai.hu@ntu.edu.sg
Anupam Chattopadhyay; Nanyang Technological University, Singapore; Email: anupam@ntu.edu.sg
Takanori Isobe; University of Hyogo, Kobe, Japan; Email: takanori.isobe@ai.u-hyogo.ac.jp
Dhiman Saha; Indian Institute of Technology, Bhilai, India; Email: dhiman@iitbhilai.ac.in

***Abstract*—This paper presents "SASQUATCH", an open-source tool, that aids in finding an unknown substitution box (SBox) given its properties. The inspiration of our work can be directly attributed to the DCC 2022 paper by Lu, Mesnager, Cui, Fan and Wang. Taking their work as the foundation (i.e., converting the problem of SBox search to a satisfiability modulo theory instance and then invoking a solver), we extend in multiple directions (including – but not limiting to – coverage of more options, imposing time limit, parallel execution for multiple SBoxes, non-bijective SBox), and package everything within an easy-to-use interface. We also present ASIC benchmarks for some of the SBoxes.**

***Index Terms*—sbox, ddt, lat, dbn, lbn, apn, linear structure**

I. Introduction

The 'substitution box' ('SBox', for short) is an important component in modern cipher construction. In many ciphers, this is the only component that introduces non-linearity. Although in simple terms, it is a look-up table (often a permutation of 2^n; common choice of n being 4 and 8), the security claims/proofs of a cipher are heavily dependent on the cryptographic properties of the SBox used. For this reason, and also because it is an interesting theoretical problem to study; there have been a number of attempts to find a desirable SBox. For instance, the works of [1] or [2, Section 3.3] go into details of finding a good SBox to be used in the context of a cipher construction, the work of [3] deals with classifying SBox according to its certain properties.

In this work, we closely follow the DCC'22 paper by Lu, Mesnager, Cui, Fan and Wang [4]. This paper targets to find an unknown SBox from its properties (this is a hard problem in general, due to the large search space). The authors convert the problem of SBox search to a *Satisfiability Modulo Theory* (SMT) instance, before passing the instance to a solver. The solver which is used in this case is the *Simple Theorem Prover* (STP), which takes an SMT instance in a particular format. In this way, they managed to find some 4- and 5-bit SBoxes from the user-specified properties. No source-code is made public by [4] though, only some code snippets are given within the paper. This motivates us to create a complete and easy-to-use open-source tool.

Contribution & Organization

The background material is covered in Section II. After this, the basic SMT model is described in Section III. At its core, our tool uses SMT conversion and solving, similar to that of [4]. In other words, given the desired properties (which are taken from configuration file in JSON format), our tool converts the search problem to an instance of SMT problem. Thereafter an SMT solver (in this case, an STP solver) is used. The source-code of our tool can be accessed as an open-source project online[1]. Thereafter, we present some ASIC benchmarks in Section VII where we show the reduced costs from what are currently reported. The newly found SBoxes (and impossibility results) are abridged in Section VI. Finally, the paper concludes in Section VIII.

II. Background/Prerequisites

A. SBox and Its Cryptographic Properties

Bijective SBoxes are used almost exclusively; Non-bijective SBoxes are used sporadically, such as in the legacy cipher `DES` (6×4) or `GAGE`/`INGAGE` [5]. However, search for both bijective and non-bijective SBox is an interesting area of research [3].

Differential Distribution Table Let S: $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be an SBox. For any $\alpha \in \mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^n$ DDT or $\delta_S(\alpha,\beta)$ is defined as : $\delta_S(\alpha,\beta) = \#\{x \in \mathbb{F}_2^n \mid S(x) \oplus S(x \oplus \alpha) = \beta\}$

[1]Located at https://github.com/mnswdhw/Sasquatch-public.

Differential Uniformity [6] The differential uniformity of an SBox S: $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ is defined as : $\mathcal{U}(S) \triangleq \max_{\alpha \in \mathbb{F}_2^n \setminus \{0\}, \beta \in \mathbb{F}_2^m} \delta_S(\alpha, \beta)$ The $\mathcal{U}(S)$ of a SBox is ≥ 2. When $\mathcal{U}(S)$ is 2, we call the SBoxalmost perfect non-linear (APN).

Differential Branch Number The differential branch number (DBN) of an SBox is defined as: $\min \{\mathrm{HW}(\alpha) + \mathrm{HW}(\beta) \mid \delta_S(\alpha, \beta) \neq 0, 0 \leq \alpha, \beta < 2^n\}$

Linear Approximation Table Let S: $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be an SBox.For any $\alpha \in \mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^n$ LAT or $\lambda_S(\alpha, \beta)$ is defined as : $\lambda_S(\alpha, \beta) = \#\{x \in \mathbb{F}_2^n \mid \alpha \cdot x \oplus \beta \cdot S(x) = 0\} - 2^{n-1} = \frac{1}{2} \sum_{x \in \mathbb{F}_2^n} (-1)^{\alpha \cdot x \oplus \beta \cdot S(x)}$

The Walsh Fourier Transform is defined as : $\mathcal{W}_S(\alpha, \beta) \triangleq \sum_{x \in \mathbb{F}_2^n} (-1)^{\alpha \cdot x \oplus \beta \cdot S(x)}$

Linearity [6] The Linearity of an SBox S: $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ is defined as :

$\mathcal{L}(S) = \max_{\alpha \in \mathbb{F}_2^n, \beta \in \mathbb{F}_2^m \setminus \{0\}} |\mathcal{W}_S(\alpha, \beta)|$

For bijective SBoxes, $\mathcal{L}(S) \geq 2^{(n+1)/2}$. When the linearity of an SBox S reaches the lower bound, this S is called almost bent (AB) function. It is advised to keep the linearity as low as possible for better resistance against linear cryptanalysis. An AB function has the maximum resistance against both differential and linear cryptanalysis, as an AB function is also APN function.

Linear Branch Number (LBN) The linear branch number (LBN) of an SBox is defined as: $\min \{\mathrm{HW}(\alpha) + \mathrm{HW}(\beta) \mid \lambda_S(\alpha, \beta) \neq 0, 0 \leq \alpha, \beta < 2^n\}$

B. SMT Problem

In computer science, a Boolean satisfiability problem (SAT) [7] is the problem of determining if there exists an interpretation that satisfies a given Boolean formula, i.e., it asks whether the variables involved in a given Boolean formula can be consistently replaced by `True` or `False`. If this is the case, the formula is called *satisfiable*, otherwise *unsatisfiable*. In certain application, we also consider arithmetic operations, for instance, the arithmetic sum of Boolean variables, which leads to the satisfiability modulo theory (SMT) problem. In an SMT [8] problem, some functions and predicate symbols have additional interpretations for the decision formula, which makes it become a much richer language than SAT. Solving SAT and SMT problems, there are many public available solvers. In this paper, we translate our problem that determining a division trail of a non-binary matrix is valid to an SMT problem, and deploy the STP [9] and Cryptominisat-5 [10] as our solvers.

SMT has certain similarity with the 0-1 integer programming problem or mixed integer linear programming (MILP), while the underlying ideas of solving them differ significantly. For the MILP, linear programming solvers first regard the problem as a general linear programming problem in real numbers, then by branch-and-cut strategy, they carefully rule out the illegal branches and then limit the solution to 0-1 integers. SMT solvers try to translate the problem to SAT, then solve it in a binary field. Due to the different methodologies of solvers, their performances depend heavily on the background and the structure of the underlying problem.

C. SBox Search Strategy in Other Works

In general, a new SBox is either presented in a new cipher construction (such as, APN function [11], [12] or linear structure [13, Chapters 7, 8]) or a paper specializing in finding new SBoxes (such as, [3], [14]).

III. SMT Model for SBox Search

In [4], Lu et al. introduce a general SMT/SAT model of searching for SBoxes with desirable cryptographic properties such as the differential branch number (DBN), linear branch number (LBN) or differential uniform and so on. Here we adapted their paper to search for SBoxes with a proper number of LS, DBN and LBN. Following [4], we also take the STP [9] and Cryptominisat-5 [10] as our solvers. Note that other solvers do exist, e.g., Z3[2] is used in [15], [16].

STP and Cryptominisat-5 are already popular in cryptographic research community, where these have been used; for instance, to search for differential and linear characteristics [17], impossible differentials [18] and division properties [19], [20]. The Sage[3] binding for Cryptominisat is used in other areas in cryptography [21], [22]. Since Cryptominisat-5 is called by STP automatically, we only need to concentrate on the STP itself. The grammar of STP can be found in its website[4], we also introduce some basic statements here for the sake of clarity.

The SMT solver STP takes CVC language as one of its input language and it focuses on the bit vectors. There are two main variable types in STP.

- `BITVECTOR`(n): Declares a bit vector variable of length n;
- `BITVECTOR`(n) of `BITVECTOR`(m): Declare an array with n-bit index and the value of each element is an m-bit vector, and we denote it as `ARRAY` (n, m) in this document.

[2]Available at https://github.com/Z3Prover/z3.

[3]https://doc.sagemath.org/html/en/reference/sat/sage/sat/solvers/cryptominisat.html.

[4]Available at https://stp.github.io.

ARRAY is the key component which can be used to model an SBox or its differential distribution table (DDT). STP supports almost all the word-wise and bit-wise functions between vectors and arithmetic operations. We focus on bit-wise operations such as XOR and arithmetic operations such as PLUS in our models to simulate the generation of the DDT from an SBox. STP also allows condition statements IF-ELSE-THEN. With these operations, we can model any constraints in order to accumulate the difference pairs.

The core of Lu et al.'s [4] model is to construct constraints on the conversion between an SBox and its differential distribution table (DDT) or linear approximation table (LAT) based on the STP grammars. Here we briefly recall these constraints.

Constraints for SBox and DDT

For an SBox: $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ [5], each entry of the corresponding DDT which is a two-dimensional table with 2^n rows and 2^n columns that can be calculated as

$$\mathrm{DDT}(\Delta_i, \Delta_o) = \#\{x \in \mathbb{F}_2^n : S(x) \oplus S(x \oplus \Delta_i) = \Delta_o\},$$

where Δ_i and Δ_o are called the input and output difference, respectively. The SBox can be represented by an ARRAY variable while the DDT can be represented by 2^{2n} BITVECTOR variables. The SBox should be a map, then we add the first 2^{2n} constraints on SBox as

$$S(x) \neq S(y), \text{ if } x \neq y, \forall\ x, y \in \mathbb{F}_2^n.$$

To link an SBox S with its DDT, we use the STP language to simulate the process of generating the DDT from an SBox. For any input/output difference pair (Δ_i, Δ_o), we introduce 2^n binary (one-bit BITVECTOR variable) dummy variables $\mathcal{X}(\Delta_i, \Delta_o, x)$ for $x \in \mathbb{F}_2^n$ indicating whether $S(x) \oplus S(x \oplus \Delta_i) = \Delta_o$, concretely,

$$\mathcal{X}(\Delta_i, \Delta_o, x) = \begin{cases} 1, & \text{if } S(x) \oplus S(x \oplus \Delta_i) = \Delta_o \\ 0, & \text{if } S(x) \oplus S(x \oplus \Delta_i) \neq \Delta_o \end{cases}.$$

This can be implemented by the IF-ELSE-THEN statement of STP. Additionally, due to the symmetry of the XOR operation, we also require

$$\mathcal{X}(\Delta_i, \Delta_o, x) = \mathcal{X}(\Delta_i, \Delta_o, x \oplus \Delta_i). \tag{1}$$

Then for all 2^{2n} possibilities of (Δ_i, Δ_o), we totally need 2^{3n} binary dummy variables. The entry of $\mathrm{DDT}(\Delta_i, \Delta_o)$ is then calculated by

$$\mathrm{DDT}(\alpha, \beta) = \sum_{x \in \mathbb{F}_2^n} \mathcal{X}(\Delta_i, \Delta_o, x).$$

The summation is easy to implement using the BVPLUS statement.

[5]For simplicity, here we only consider the SBoxes that are permutations.

Constraints for SBox and LAT

The LAT of a SBox is also a two-dimensional table with 2^n rows and 2^n columns, represented by 2^{2n} BITVECTOR variables. Each entry of the LAT is calculated by

$$\mathrm{LAT}(\Gamma_i, \Gamma_o) = \{x \in \mathbb{F}_2^n : \Gamma_i \cdot x = \Gamma_o \cdot S(x)\},$$

where $\Gamma_i, \Gamma_o \in \mathbb{F}_2^n$ are the input and output masks, respectively, $\alpha \cdot \beta$ is the inner product of α and β. Since STP supports the bit operation AND (&) and XOR (BVXOR) , with 2^{3n} binary dummy variables $\mathcal{X}'(\Gamma_i, \Gamma_o, x)$ we are easy to simulate the generation of the LAT akin to the generation of the DDT. Note we do not need Equation (1) to explicitly require the symmetry.

Constraints for DBN and LBN

We have set up the constraints binding the SBox and its DDT and LAT, then we can add more constraints on DDT or LAT to obtain some fine properties we want. The DBN is defined and calculated as

$$\min_{\Delta_i, \Delta_o \in \mathbb{F}_2^n} \{wt(\Delta_i) + wt(\Delta_o) : \mathrm{DDT}(\Delta_i, \Delta_o) \neq 0\}.$$

Similarly, LAT is defined and calculated as,

$$\min_{\Gamma_i, \Gamma_o \in \mathbb{F}_2^n} \left\{wt(\Gamma_i) + wt(\Gamma_o) : \mathrm{LAT}(\Gamma_i, \Gamma_o) \neq 2^{n-1}\right\}.$$

Hence, to force the DBN be a given value, we add the following constraints,

$$\begin{cases} \mathrm{DDT}(\Delta_i, \Delta_o) = 0,\ \forall\ \Delta_i, \Delta_o \in \mathbb{F}_2^n \text{ s.t. } wt(\Delta_i) + wt(\Delta_o) < \mathrm{DBN} \\ \underset{wt(\Delta_i)+wt(\Delta_o)=\mathrm{DBN}}{\mathrm{CON}} \neq 0||0||\cdots||0 \end{cases},$$

where $\underset{wt(\Delta_i)+wt(\Delta_o)=\mathrm{DBN}}{\mathrm{CON}}$ stands for the concatenation of all entries of DDT with the DBN being DBN. The first equation lets all entries be zero if the branch number of the corresponding input and output differences are less than the DBN, while the second forces at least one entry with the branch number equaling DBN to meet the definition of the DBN.

For the LBN of the LAT, the process is almost the same except that we focus on the bias. Then the constraints for the LBN are

$$\begin{cases} \mathrm{LAT}(\Gamma_i, \Gamma_o) = 2^{n-1},\ \forall\ \Gamma_i, \Gamma_o \in \mathbb{F}_2^n \text{ s.t. } wt(\Gamma_i) + wt(\Gamma_o) < \mathrm{LBN}, \\ \underset{wt(\Gamma_i)+wt(\Gamma_i)=\mathrm{LBN}}{\mathrm{CON}} \mathrm{LAT}(\Delta_i, \Delta_o) \neq 2^{n-1}||2^{n-1}||\cdots||2^{n-1} \end{cases}.$$

Constraints for number of LS

The constraints for the number of linear structure is not included in the original model of [4]. To construct the constraints forcing the DDT have a given number of LS, we need 2^{2n} binary dummy variables $\mathcal{S}(\Delta_i, \Delta_o)$ satisfying

$$\mathcal{S}(\Delta_i, \Delta_o) = \begin{cases} 1, & \text{if DDT}(\Delta_i, \Delta_o) = 2^n \\ 0, & \text{if DDT}(\Delta_i, \Delta_o) < 2^n \end{cases}.$$

Then the number of LS denoted by NLS can be constrained by $\sum_{\Delta_i, \Delta_o \in \mathbb{F}_2^n} \mathcal{S}(\Delta_i, \Delta_o)$.

IV. SASQUATCH DESCRIPTION

Using this tool, we can search multiple SBoxes concurrently or in sequence. This section will elaborate on the cryptographic properties that are currently supported as well as other crucial features.

A. Global Parameters

- **Global Timeout:** Specify the time in seconds, it will wait for finding the SBoxes. If this parameter is specified then it will override the individual timeout option of particular SBox.
- **Sequential:** Boolean True/False, to control parallel processing of SBoxes.
- **Output Directory:** User specified results directory.
- **Debug:** Boolean True/False. If True, Store debug files to assist the developer.

B. SBox Specific Parameters

- **Size:** Size of the SBox.
- **Fixed Point:** If false, then SBox will not have any fixed point.
- **Differential Uniformity:** Specify the required differential uniformity value of the SBox and the sign $==, \geq$ or $\leq$. == means the SBox will have the specified value ($\geq$ will make the SBox will have differential uniformity at least the specified value, $\leq$ will make the SBox will have differential uniformity at most the specified value).
- **Frequency DU:** Specify the number of differential uniformity values required in the DDT table.
- **Linearity:** Specify the required linearity value and the sign $==, \geq$ or $\leq$. The interpretations of these signs are the same as in differential uniformity.
- **Frequency DU:** Specify the number of linearity values required in the LAT table.
- **Bad Input Bad Output DDT:** Here, BIBO means that the hamming weight of both input and output differences (or masks) is exactly one. Specify the number of BIBO patterns in the DDT table.
- **Bad Input Bad Output LAT:** Specify the number of BIBO patterns in the LAT table.
- **Differential Branch Number:** Specify the DBN of the SBox and the associated sign from $==, \geq$ or $\leq$.
- **Linear Branch Number:** Specify the LBN of the SBox and the associated sign from $==, \geq$ or $\leq$.
- **Boomerang Connectivity Table:** Boolean True/False. If True, it will compute the BCT table of the SBox.
- **Involution:** Boolean True/False. If true, the SBox will be equal to it's inverse construction.
- **Look-up:** Specify the entries in the lookup table of the SBox and the corresponding signs $==, \geq$ or $\leq$.
- **Time-out:** Specify the timeout of individual SBox.

CODE 1: CVC formulation for DDT

```
1  //Initialize Variables
2  DDT(a,b) : BITVECTOR(n)  //for all a,b
3  istrue(a,b,x) : BITVECTOR(1) //for all
   ↪  a,b,x
4
5  //Compute DDT
6
7  ASSERT( IF BVXOR(S[x], b) = S[BVXOR( x,
   ↪  a)] THEN
8  IsTrue[a, b, x] = 0bin1 ELSE IsTrue[a, b,
   ↪  x] = 0bin0 ENDIF );
9  ASSERT( DDT[a, b] = BVPLUS( n, IsTrue[a,
   ↪  b, 0], IsTrue[a, b, 1],.... ) );
```

V. CVC CODES

The Simple Theorem Prover (STP) is a constraint solver for the theory of quantifier-free bitvectors that can solve many kinds of problems generated by program analysis tools, theorem provers, automated bug finders, cryptographic algorithms, intelligent fuzzers and model checkers.The file based input formats that STP reads are the: CVC, SMT-LIB1, and SMT-LIB2 formats. STP also offers the functionality to convert from one format to another seamlessly. We need to model the problem of finding the SBox, satisfying the required constraints into a satisfiability problem. For doing this first we need to model the DDT and LAT table using CVC.

Codes 1, 2, 3, 4, 5 and 6 show the basic CVC formulation for DDT, LAT, DU, linearity, DBN and LBN, respectively with inline comments. Any of the CVC codes are for representation purposes only (not directly executable). Note that is used concatenate two binary strings; and there is no inherent comment character, but we use // to write inline comments.

CODE 2: CVC formulation for LAT

```
// Initialize Variables (for a 4 bit S-box)
LAT(a,b) : BITVECTOR(6)  //for all a,b
istrue(a,b,x) : BITVECTOR(1) //for all a,b,x
H: ARRAY BITVECTOR(4) OF BITVECTOR(4) //define hamming weight table
ASSERT( H[0bin0001] = 0bin0001) // assert for all values
W(a,b) : BITVECTOR(7)

//Compute LAT
ASSERT( IF ( (H[BVXOR((a & x),(b & S[x]))] & 0bin0001) = 0bin0000 ) THEN istrue(a,b,x) = 0bin1 ELSE istrue(a,b,x) = 0bin0 ENDIF ); //for all a,b,x

ASSERT( LAT(a,b) = BVSUB(6, BVPLUS(6, 0bin00000@istrue(a,b,x), 0bin00000@istrue(a,b,x1)......, 0bin001000));
W(a,b) = 2* LAT(a,b) //for all a,b
```

CODE 3: CVC formulation for DU

```
//Initialize Variables
#Freq[a,b] : BITVECTOR(<size>)
Freq[a,b] : BITVECTOR(1) // for all a,b

// U(S) is the given DU value
DDT[a,b] <= U(S) // for all a,b
ASSERT( IF DDT[a, b] = U(S) THEN Freq[a, b] = 0bin1 ELSE Freq[a, b] = 0bin0 ENDIF );
ASSERT( #Freq = BVPLUS( n, Freq[a1,b1], Freq[a2,b2],.... ) );
ASSERT(#Freq = <required frequency du>)
```

CODE 4: CVC formulation for linearity

```
% //Initialize Variables
% #Freq[a,b] : BITVECTOR(<size>)
% Freq[a,b] : BITVECTOR(1) // for all a,b

% // L(S) is the given DU value
% abs(W[a,b]) <= L(S) // for all a,b
% ASSERT( IF W[a, b] = L(S) THEN
% Freq[a, b] = 0bin1 ELSE Freq[a, b] = 0bin0 ENDIF );
% ASSERT( #Freq = BVPLUS( n, Freq[a1,b1], Freq[a2,b2],.... ) );
% ASSERT(#Freq = <required frequency lu>)
```

CODE 5: CVC formulation for DBN

```
% //Here `wt` represents Hamming weight
% ASSERT(DDT[a,b] = 0bin0) //for each wt(a) + wt(b) < DBN.
% ASSERT(DDT[a,b] /= 0bin0) //for each wt(a) + wt(b) = DBN.
```

CODE 6: CVC formulation for LBN

```
% //Here `wt` represents Hamming weight
% ASSERT(LAT[a,b] = 0bin0) //for each wt(a) + wt(b) < LBN.
% ASSERT(LAT[a,b] /= 0bin0) //for each wt(a) + wt(b) = LBN.
```

VI. RESULTS (NEW SBOXES AND IMPOSSIBILITY)

A. Impact of Redundant Constraints

The search procedure is deterministic – the same output will always be obtained for the same input constraints. Being partly motivated from [23], we also experiment with some redundant constraints.

Redundant constraints are those constraints that do not affect the optimal solution found by the STP solver, but they influence the solver's run time. To reduce the run time of the solver, the idea is to insert more constraints so as to minimize its search space. This can also have the opposite outcome of increasing the run-time sometimes. In [24], the authors observe that the number of constraints can influence the solution time taken by the MILP solver Gurobi. Inspired by this, we also experiment with the STP solver by putting redundant constraints in calculation of DDT and differential uniformity. Table I captures the impact on run-time in a nutshell.

1) DDT contains no odd value: The DDT of an SBox cannot have odd values as its entries. To prevent the STP solver from asserting the different constraints when a value in DDT is considered odd, we can think of asserting an extra condition (redundant) that no entry of a DDT is odd. We conduct a simple experiment of calculating the DU for different SBoxes under various configurations.

2) Reducing variables: In this experiment, we check the effect of reducing the number of variables controlled by the STP solver for a computation. Under a special case: sign in DU computation is == and frequency_du is null, we can significantly reduce the computation time. This effect is more pronounced when the SBox search is difficult or when the SBox is large ($\geq$ 7-bit). Under the case mentioned above, we can do away with the #Freq and Freq mentioned in Code 3. We make the assertion here, for at least one DDT entry to be equal to the specified value and another assertion that all DDT values be less than or equal to the specified value. This way the only dependence for the computation of DU is on DDT entries.

B. Outcome

We have found a 5-bit APN SBox with this tool. Currently the only known 5-bit APN SBoxes are given

TABLE I: Impact of redundant constraints

(A) DDT odd value

Configuration	Time w/o RC (sec)	Time w RC (sec)
5 bit 16 DU	127.72	212.31
5-bit/8DU	22.8	30.76
4-bit/8DU	2.44	3.47

(B) With and without frequency

Configuration	Time w/o RV (sec)	Time w RV (sec)
4-bit/8 DU	2.64	2.46
5-bit/8DU	21.82	21.46
5-bit/16DU	122.64	41

by [6] and [25]. Our SBox is different from these SBoxes. Although our SBox can be affine equivalent (AE) to [25] (both have the same linearity and distribution of unique values in LAT table), but it is difficult to ascertain because proving two SBoxes to be AE is a difficult problem. Our tool does not seem to scale up for 8-bit SBoxes.

We found the interesting outcomes as listed[6]:

- 4-bit/3LBN/0LS (not affine): 4, 13, 1, 8, 7, 2, 10, 15, 11, 14, 6, 3, 9, 0, 12, 5
- 4-bit/3DBN/3LBN/0LS does not exist
- 5-bit/16LS does not exist
- 5-bit/4DBN: 3, 16, 26, 29, 4, 30, 23, 11, 28, 5, 17, 22, 9, 2, 14, 24, 21, 15, 12, 1, 10, 19, 25, 6, 18, 8, 7, 27, 31, 20, 0, 13
- 6-bit/4DBN: 10, 7, 25, 53, 56, 20, 33, 14, 41, 12, 60, 38, 0, 19, 26, 63, 55, 46, 30, 2, 45, 1, 16, 58, 34, 31, 4, 43, 62, 48, 29, 37, 21, 40, 15, 35, 54, 13, 28, 18, 36, 50, 22, 24, 47, 57, 51, 42, 6, 11, 61, 23, 59, 39, 3, 49, 8, 5, 27, 52, 17, 44, 32, 9
- 7-bit/5DBN/5LBN/0NL: 34, 45, 17, 75, 72, 113, 86, 58, 93, 82, 110, 52, 55, 14, 41, 69, 71, 126, 12, 96, 27, 20, 125, 39, 56, 1, 115, 31, 100, 107, 2, 88, 116, 24, 63, 6, 5, 95, 99, 108, 11, 103, 64, 121, 122, 32, 28, 19, 105, 51, 90, 85, 46, 66, 48, 9, 22, 76, 37, 42, 81, 61, 79, 118, 30, 68, 120, 119, 111, 3, 36, 29, 97, 59, 7, 8, 16, 124, 91, 98, 53, 89, 43, 18, 114, 40, 65, 78, 74, 38, 84, 109, 13, 87, 62, 49, 83, 106, 77, 33, 57, 54, 10, 80, 44, 21, 50, 94, 70, 73, 117, 47, 0, 15, 102, 60, 92, 101, 23, 123, 127, 112, 25, 67, 35, 26, 104, 4
- 7-bit/4DBN/6NL: 42, 39, 95, 59, 123, 41, 100, 106, 65, 88, 84, 15, 25, 30, 43, 1, 18, 11, 124, 34, 97, 4, 89, 82, 94, 60, 27, 8, 54, 19, 69, 57, 45, 93, 3, 114, 117, 10, 58, 13, 121, 0, 111, 74, 23, 67, 76, 120, 31, 80, 119, 105, 90, 56, 28, 17, 98, 55, 21, 48, 109, 86, 33, 38, 36, 14, 61, 92, 7, 24, 9, 115, 79, 66, 99, 112, 113, 107, 50, 77, 70, 96, 12, 22, 108, 62, 118, 85, 20, 73, 122, 5, 103, 53, 78, 47, 49, 83, 101, 40, 63, 35, 52, 6, 68, 127, 87, 102, 91, 72, 46, 81, 126, 2, 16, 71, 32, 29, 125, 110, 37, 116, 75, 64, 51, 44, 104, 26
- 6-bit/3DBN/Inv/18NL: 17, 11, 48, 22, 43, 26, 10, 57, 13, 23, 6, 1, 36, 8, 55, 62, 31, 0, 41, 28, 61, 59, 3, 9, 46, 40, 5, 27, 19, 29, 30, 16, 42, 38, 54, 37, 12, 35, 33, 63, 25, 18, 32, 4, 53, 45, 24, 49, 2, 47, 60, 58, 56, 44, 34, 14, 52, 7, 51, 21, 50, 20, 15, 39
- 6-bit/4DBN/Inv: 63, 16, 56, 12, 28, 46, 18, 25, 38, 57, 10, 50, 3, 13, 32, 60, 1, 54, 6, 19, 23, 35, 41, 20, 43, 7, 29, 36, 4, 26, 51, 47, 14, 45, 39, 21, 27, 55, 8, 34, 58, 22, 49, 24, 61, 33, 5, 31, 53, 42, 11, 30, 62, 48, 17, 37, 2, 9, 40, 59, 15, 44, 52, 0
- 5-bit/4DBN/Inv: 24, 1, 13, 18, 31, 12, 6, 21, 11, 30, 23, 8, 5, 2, 16, 27, 14, 20, 3, 25, 17, 7, 28, 10, 0, 19, 26, 15, 22, 29, 9, 4
- 5-bit/2DU(APN)/4ALU: 16, 23, 28, 21, 30, 11, 19, 29, 24, 1, 0, 8, 14, 12, 31, 25, 5, 4, 2, 15, 27, 7, 17, 20, 10, 9, 22, 6, 26, 13, 3, 18
- 6-bit/4DU: 32, 1, 39, 25, 14, 13, 20, 45, 35, 61, 52, 19, 27, 11, 34, 8, 54, 33, 50, 3, 55, 46, 24, 12, 2, 15, 53, 9, 57, 18, 0, 63, 7, 41, 38, 28, 48, 5, 40, 62, 6, 43, 49, 16, 21, 10, 42, 29, 44, 23, 31, 26, 47, 58, 37, 17, 56, 60, 30, 4, 59, 36, 51, 22
- 5-bit/2DU(APN)/4ALU: 19, 12, 26, 9, 5, 1, 15, 7, 23, 6, 0, 29, 22, 28, 2, 4, 8, 3, 30, 25, 27, 11, 14, 18, 16, 21, 24, 17, 20, 10, 31, 13
- 5-bit/2DU(APN)/4ALU: 0, 29, 25, 2, 21, 31, 15, 19, 30, 14, 27, 9, 16, 28, 20, 10, 13, 11, 5, 22, 8, 6, 26, 17, 23, 1, 4, 3, 7, 18, 12, 24
- 6-bit/2DBN/8DU: 3, 13, 57, 43, 8, 33, 55, 19, 45, 60, 21, 31, 18, 37, 46, 38, 54, 50, 7, 56, 2, 30, 12, 61, 23, 1, 52, 34, 10, 40, 32, 28, 20, 63, 36, 16, 29, 24, 41, 0, 17, 14, 44, 49, 4, 27, 9, 22, 26, 48, 58, 25, 59, 53, 47, 39, 62, 51, 6, 5, 11, 15, 42, 35
- 5-bit/2DBN/8DU/Inv: 22, 26, 19, 29, 14, 12, 10, 13, 28, 24, 6, 25, 5, 7, 4, 30, 23, 21, 20, 2, 18, 17, 0, 16, 9, 11, 1, 31, 8, 3, 15, 27
- 5-bit/8LS: 3, 6, 21, 16, 20, 17, 7, 2, 15, 10, 28, 25, 24, 29, 14, 11, 22, 19, 5, 0, 1, 4, 23, 18, 26, 31, 12, 9, 13, 8, 30, 27

[6]Some more 4-bit SBoxes with LS are available in the code repository.

VII. ASIC Benchmarks

We report updated ASIC costs of SBoxes using a different (coordinate function) representation of the SBoxes. We use the same ASIC library, namely, Mentor LeonardoSpectrum Level 3 (2018a.2) is used for synthesis with the UMC 65 nm Low-Power RVT (Regular VT) Standard Performance Generic Core Cell Library from Faraday. Implementation of all the SBoxes in this article

are done in Verilog. Using the library, we have also calculated the power and latency incurred for the hardware implementation of the SBoxes. The results are shown in Table II.

TABLE II: ASIC cost comparison

SBox	Our cost (GE)	Cost in [24] (GE)
S1	86.25	86.71
S2	142.00	158.59
S3	36.50	44.53

S1: 0, 58, 47, 28, 3, 29, 24, 15, 23, 53, 32, 11, 46, 40, 45, 34, 61, 35, 62, 41, 16, 42, 39, 20, 1, 7, 26, 21, 22, 52, 57, 18, 30, 54, 17, 48, 9, 5, 50, 55, 8, 56, 31, 38, 37, 49, 6, 27, 2, 14, 33, 36, 59, 19, 44, 13, 63, 43, 4, 25, 60, 12, 51, 10

S2: 0, 45, 48, 15, 58, 32, 14, 49, 13, 7, 41, 12, 3, 54, 55, 26, 42, 25, 22, 34, 60, 38, 53, 31, 21, 51, 4, 24, 27, 28, 43, 33, 39, 19, 30, 63, 16, 1, 59, 8, 57, 62, 29, 50, 6, 44, 36, 17, 23, 10, 56, 37, 9, 47, 5, 20, 40, 52, 35, 2, 18, 61, 46, 11

S3: 0, 27, 14, 17, 22, 15, 24, 5, 30, 7, 25, 4, 11, 16, 12, 19, 3, 8, 9, 6, 21, 28, 31, 18, 20, 29, 23, 26, 1, 10, 2, 13

VIII. Conclusion

The main objective in this work is to present an open-source tool named SASQUATCH, and also show several ASIC benchmarks for the SBoxes. Our tool offers an easy-to-use interface to find an SBox given the desirable properties of the SBox (if exists). We have included support for properties like differential uniformity and/or its frequency, the maximum entry in the absolute LAT and/or its frequency, linearity, differential and linear branch number, non-bijective SBox. Additional search options like parallel processing, time limit, information about some of the look-up values etc. are also included in our tool. We find some new SBoxes/impossibility results using SASQUATCH (such as, 5-bit bijective SBox with differential uniformity 2).

We are confident that this tool will help the upcoming cipher designers immensely by allowing them to quickly find a large pool of SBoxes with the properties they want. Although the problem has been studied before (most notably in the DCC'22 paper [4]), our tool offers the maximum set of options/alternative to the best of our finding, it is easily extensible, and can be used without delving into the intricacy of the source codes. In future, one may be interested to cover other features like those needed for side channel countermeasure to an SBox, finding a linear permutation layer, or many-to-one Boolean functions.

References

[1] S. Sarkar, H. Syed, R. Sadhukhan, and D. Mukhopadhyay, "Lightweight design choices for led-like block ciphers," Cryptology ePrint Archive, Paper 2017/1031, 2017, https://eprint.iacr.org/2017/1031. [Online]. Available: https://eprint.iacr.org/2017/1031

[2] R. Avanzi, "The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 4–44, 2017. [Online]. Available: https://doi.org/10.13154/tosc.v2017.i1.4-44

[3] M. Brinkmann and G. Leander, "On the classification of APN functions up to dimension five," *Des. Codes Cryptogr.*, vol. 49, no. 1-3, pp. 273–288, 2008. [Online]. Available: https://doi.org/10.1007/s10623-008-9194-6

[4] Z. Lu, S. Mesnager, T. Cui, Y. Fan, and M. Wang, "An stp-based model toward designing s-boxes with good cryptographic properties," *Designs, Codes and Cryptography*, vol. 90, no. 5, pp. 1179–1202, 2022. [Online]. Available: https://doi.org/10.1007/s10623-022-01034-2

[5] D. Gligoroski, H. Mihajloska, D. Otte, and M. El-Hadedy, "Gage and ingage v1.03," 2019, http://gageingage.org/.

[6] K. Nyberg, "Differentially uniform mappings for cryptography," in *Advances in Cryptology — EUROCRYPT '93*, T. Helleseth, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 55–64.

[7] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[8] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 825–885. [Online]. Available: https://doi.org/10.3233/978-1-58603-929-5-825

[9] V. Ganesh and D. L. Dill, "A decision procedure for bit-vectors and arrays," in *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Springer, 2007, pp. 519–531. [Online]. Available: https://doi.org/10.1007/978-3-540-73368-3_52

[10] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT solvers to cryptographic problems," in *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed., vol. 5584. Springer, 2009, pp. 244–257. [Online]. Available: https://doi.org/10.1007/978-3-642-02777-2_24

[11] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, F. Mendel, B. Mennink, N. Mouha, Q. Wang, and K. Yasuda, "Primates v1.02," 2014, submission to CAESAR. [Online]. Available: http://competitions.cr.yp.to/round2/primatesv102.pdf

[12] B. Bilgin, A. Bogdanov, M. Knezevic, F. Mendel, and Q. Wang, "Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware," in *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, ser. Lecture Notes in Computer Science, G. Bertoni and J. Coron, Eds., vol. 8086. Springer, 2013, pp. 142–158. [Online]. Available: https://doi.org/10.1007/978-3-642-40349-1_9

[13] A. Baksi, *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. Springer, Singapore, 2022, https://doi.org/10.1007/978-981-16-6522-6.

[14] G. Ivanov, N. Nikolov, and S. Nikova, "Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties," *Cryptogr. Commun.*, vol. 8, no. 2, pp. 247–276, 2016. [Online]. Available: https://doi.org/10.1007/s12095-015-0170-5

[15] S. Kumar, V. A. Dasu, A. Baksi, S. Sarkar, D. Jap, J. Breier, and S. Bhasin, "Side channel attack on stream ciphers: A three-step approach to state/key recovery," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 2, p. 166–191, Feb. 2022. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9485

[16] A. Baksi, S. Kumar, and S. Sarkar, "A new approach for side channel analysis on stream ciphers and related constructions," *IEEE Trans. Computers*, vol. 71, no. 10, pp. 2527–2537, 2022. [Online]. Available: https://doi.org/10.1109/TC.2021.3135191

[17] Y. Liu, H. Liang, M. Li, L. Huang, K. Hu, C. Yang, and M. Wang, "STP models of optimal differential and linear trail for s-box based ciphers," Cryptology ePrint Archive, Report 2019/025, 2019, https://eprint.iacr.org/2019/025.

[18] X. Hu, Y. Li, L. Jiao, S. Tian, and M. Wang, "Mind the propagation of states - new automatic search tool for impossible differentials and impossible polytopic transitions," in *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, S. Moriai and H. Wang, Eds., vol. 12491. Springer, 2020, pp. 415–445. [Online]. Available: https://doi.org/10.1007/978-3-030-64837-4_14

[19] K. Hu, Q. Wang, and M. Wang, "Finding bit-based division property for ciphers with complex linear layers," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. 1, pp. 396–424, 2020. [Online]. Available: https://doi.org/10.13154/tosc.v2020.i1.396-424

[20] K. Hu and M. Wang, "Automatic search for a variant of division property using three subsets," in *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, ser. Lecture Notes in Computer Science, M. Matsui, Ed., vol. 11405. Springer, 2019, pp. 412–432. [Online]. Available: https://doi.org/10.1007/978-3-030-12612-4_21

[21] A. Baksi, S. Maitra, and S. Sarkar, "An improved slide attack on Trivium," *Journal IPSI Transaction on Internet Research*, 2015. [Online]. Available: http://vipsi.org/ipsi/journals/journals/papers/tar/2015jan/p1.pdf

[22] S. Maitra, S. Sarkar, A. Baksi, and P. Dey, "Key recovery from state information of sprout: Application to cryptanalysis and fault attack," *IACR Cryptology ePrint Archive*, vol. 2015, p. 236, 2015. [Online]. Available: http://eprint.iacr.org/2015/236

[23] A. Baksi, "New insights on differential and linear bounds using mixed integer linear programming (full version)," Cryptology ePrint Archive, Report 2020/1414, 2020, https://eprint.iacr.org/2020/1414.

[24] S. Sarkar, K. Mandal, and D. Saha, "On the relationship between resilient boolean functions and linear branch number of s-boxes," Cryptology ePrint Archive, Report 2019/1427, 2019, https://eprint.iacr.org/2019/1427.

[25] B. Bilgin, A. Bogdanov, M. Knežević, F. Mendel, and Q. Wang, "Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware," 08 2013, pp. 142–158.