IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 0, No. 0, pp. 0–0. DOI:10.46586/tosc.v0.i0.0-0

SoK: Modeling for Large S-boxes Oriented to Differential Probabilities and Linear Correlations (Long Paper)

Ling Sun[1,2,3,4] and Meiqin Wang(✉)[1,3,4]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[2] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, 100878, China
[3] School of Cyber Science and Technology, Shandong University, Qingdao, China
[4] Quan Cheng Shandong Laboratory, Jinan, China
lingsun@sdu.edu.cn, mqwang@sdu.edu.cn

Abstract. Automatic methods for differential and linear characteristic search are well-established at the moment. Typically, the designers of novel ciphers also give preliminary analytical findings for analysing the differential and linear properties using automatic techniques. However, neither MILP-based nor SAT/SMT-based approaches have fully resolved the problem of searching for actual differential and linear characteristics of ciphers with large S-boxes. To tackle the issue, we present three strategies for developing SAT models for 8-bit S-boxes that are geared toward differential probabilities and linear correlations. While these approaches cannot guarantee a minimum model size, the time needed to obtain models is drastically reduced. The newly proposed SAT model for large S-boxes enables us to establish that the upper bound on the differential probability for 14 rounds of SKINNY-128 is 2^{-131}, thereby completing the unsuccessful work of Abdelkhalek *et al.* We also analyse the seven AES-based constructions C1 - C7 designed by Jean and Nikolić and compute the minimum number of active S-boxes necessary to cause an internal collision using the SAT method. For two constructions C3 and C5, the current lower bound on the number of active S-boxes is increased, resulting in a more precise security analysis for these two structures.

Keywords: Automatic cryptanalysis · differential characteristic · SKINNY-128 · PIPO · AES-based construction

1 Introduction

After more than a decade of development, automatic search techniques have become an increasingly potent cryptanalysis tool. Search techniques based on mixed-integer linear programming (MILP) [MWGP11, WW11] and Boolean satisfiability problem or satisfiability modulo theories (SAT/SMT) [MP13] are two of the most popular. From an early focus on differential and linear distinguishers [MWGP11, WW11, SHW+14b, FWG+16, AST+17, MP13, KLT15, LWR16], the application of automatic tools has been gradually expanded to include integral distinguishers [XZBL16, SWW17], polynomials in cube attacks [TIHM17, WHG+19], and distinguishers in meet-in-the-middle attacks [SSD+18, BDG+21], etc.

Automatic methods for differential and linear characteristic search are currently well-established. Typically, the designers of novel ciphers also provide preliminary analytical results for analysing the differential and linear properties via automatic approaches.

However, neither MILP-based nor SAT/SMT-based approaches have yet completely solved the challenge of searching for true differential and linear characteristics of ciphers with large S-boxes.

Abdelkhalek *et al.* [AST+17] investigated the differential distribution table (DDT) of large S-boxes and provided the first method for constructing MILP models for large S-boxes in the development of MILP modelling. The new approach was implemented on `SKINNY-128` [BJK+16] and two `AES`-based constructions in [JN16]. The upper bound on differential probability for 14 rounds of `SKINNY-128` is not completely resolved due to the extremely lengthy runtime of the MILP optimiser. While the minimum number of differential active S-boxes for one `AES`-based construction is raised, the precise lower bound on the number of active S-boxes is not entirely known. Despite the fact that Abdelkhalek *et al.* made it possible to describe large S-boxes, the model for the $*$-DDT, which is a reduced version of the DDT in which all non-zero entries are replaced by one, of the S-box in `AES` consists of 8302 linear inequalities, which is sometimes too complicated for practical applications. To tackle this issue, Boura and Coggia [BC20] proposed two strategies for constructing efficient algorithms for modelling 8-bit S-boxes with the minimum amount of linear inequalities. Notably, the enhanced MILP models for the S-boxes of `AES` and `SKINNY-128` are not used to search for differential characteristics of the two ciphers.

In the SAT/SMT modelling development, Ankele and Kölbl [AK18] stated that their S-box modelling technique was applicable to 8-bit S-boxes; yet, all of the ciphers analysed in the study use 4-bit S-boxes. The SMT model suggested by Liu *et al.* [LLL+19] can manage 8-bit S-boxes. The authors used their model to investigate the security of `AES` and `SKINNY-128` against the impossible differential attack; once again, the differential probability of the two ciphers is not taken into account.

Motivated by resolving the aforementioned unsolved issues pertaining to `SKINNY-128` and `AES`-based constructions, we want to identify a way for efficiently creating SAT models of large S-boxes.

Our contributions. After examining the evolution of MILP and SAT/SMT modelling techniques, we see that the search for actual differential and linear characteristics for ciphers with large S-boxes is not conclusive. We intend to provide effective ways for generating SAT models for large S-boxes.

Three strategies for creating SAT models for large S-boxes oriented to differential probabilities and linear correlations are proposed. Although these methods cannot ensure a minimum model size, the time required to obtain models is greatly decreased. The three tactics are summarised in the following.

1. Since developing SAT models for S-boxes is equal to finding a simplification of a given Boolean function, we attempt to strike a balance between the degree of simplification and the runtime. The first strategy utilises the option of the ESPRESSO logic minimizer [BHH+82, BHMS84], which is an algorithm used to simplify Boolean functions.

2. Noting that the complexity of simplification rises exponentially with the number of input variables of the given Boolean function, we attempt to lower the size of the function that ESPRESSO delivers. The primary concept of the second technique is to divide the description of a big S-box into two steps, so transforming the simplification of large-scale Boolean functions into the simplification of two relatively small-scale functions.

3. In the test, we observe that simplification of a large-scale function is not difficult if the number of terms is not extremely enormous. This discovery motivates the partitioning mechanism of third strategy. Specifically, the target n-bit Boolean function f is subdivided into many n-bit functions f_0, f_1, …, and $f_{\ell-1}$, where ℓ is

an integer, so that the conjunction $\bigwedge_{i=0}^{\ell-1} f_i$ is identical to f. Then, the simplification of f is converted into the simplification of f_0, f_1, …, and $f_{\ell-1}$, hence accelerating the production of the SAT model.

SKINNY-128 is used to evaluate the three strategies. Numerous SAT models are applied to calculate the upper bound of differential probability for SKINNY-128 from 1 to 14 rounds. After analysing all the test results, we believe that the first encoding technique achieves a good compromise between the runtime of ESPRESSO and the execution time of the SAT solver. The test results also reveal that lowering the number of constraints in SAT problems does not necessarily reduce the execution time of the SAT solver, echoing the observation made by Sasaki and Todo [ST17] in the MILP approach. In addition, we empirically demonstrate that reducing the number of variables in SAT problems does not always reduce the execution time of the SAT solver.

The newly presented SAT model for large S-boxes lets us to demonstrate that the upper bound on the differential probability for 14 rounds of SKINNY-128 is 2^{-131}, thereby completing the failed work of Abdelkhalek *et al.* Additionally, the related-key differential properties of both versions of PIPO [KJK+20] are investigated. For PIPO-128, we identify 1792 full-round characteristics with a probability of 2^{-24}, extending the findings of Yadav and Kumar [YK21]. For PIPO-256, we are the first to report full-round differential characteristics.

The encoding approach for large S-boxes is applied to seven AES-based constructions C1 - C7 devised by Jean and Nikolić [JN16]. The SAT method is used to calculate the minimum number of active S-boxes required to induce an internal collision. The test results for all seven constructions, ranging from two to eight stages, are shown in Table 9. For two constructions C3 and C5, the current lower bound on the number of active S-boxes is increased, resulting in a more precise security analysis for these two structures. For C1, C2, C4, and C6, we validate the previously determined lower bound for the number of active S-boxes. In addition, all differential patterns for C1 - C5 that have a minimum number of active S-boxes are supplied.

Outline. The remainder of the paper is structured as follows. In Section 2, notations and knowledge related to the automatic search method are presented. Section 3 is devoted to discussing the MILP modelling work for large S-boxes, while Section 4 discusses the SAT modelling progress for large S-boxes. In Section 5, we propose three efficient methods for building SAT models for large S-boxes. In Section 6, the newly suggested encoding techniques are implemented on SKINNY-128, and their performance is compared. Based on the outcome of the test around SKINNY-128, we recognise that the first modelling technique is superior, and it is subsequently implemented in the studies of PIPO and seven AES-based constructions in Sections 7 and 8, respectively. Section 9 is the conclusion of the paper. Supplementary Material of this paper can be found at https://github.com/SunLing134340/Large-Sbox-Supplementary-Material.

2 Preliminary

In this part, after presenting the notations used throughout the work, we will review two standard ways for simplifying Boolean functions.

2.1 Notations

The *Hamming weight* of a vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{F}_2^n$, indicated by $\mathrm{wt}(\boldsymbol{x})$, is the number of non-zero elements in $\boldsymbol{x}$, i.e., $\mathrm{wt}(\boldsymbol{x}) = \sum_{i=0}^{n-1} x_i$. A vector may be represented in binary representation, decimal representation, and hexadecimal representation; we utilise three fonts to differentiate between these representations: $\mathsf{sans\ serif\ font}$ for binary representation, roman typeface for decimal representation, and `typewriter font` for hexadecimal representation. Consequently, $\mathsf{1011}$, 11, and `0xb` represent the same vector in this study.

The *support* of $\boldsymbol{x}$, indicated by $\mathrm{supp}(\boldsymbol{x})$, is the set of coordinates of elements in $\boldsymbol{x}$ that are non-zero, i.e., $\mathrm{supp}(\boldsymbol{x}) = \{i \in [0, n-1] \mid x_i = \mathsf{1}\}$. Given a Boolean function f over $\mathbb{F}_2^n$, the *support* of f, represented as $\mathrm{supp}(f)$, is the set of vectors in $\mathbb{F}_2^n$, for which f is non-zero, i.e., $\mathrm{supp}(f) = \{\boldsymbol{x} \in \mathbb{F}_2^n \mid f(\boldsymbol{x}) = \mathsf{1}\}$.

2.2 Sum of Products and Product of Sums

For an n-variable Boolean function over $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, a conjunction in which each of the n variables appears once (either in its complemented or uncomplemented form) is known as a *minterm*, i.e., it is a logical expression of n variables using just the complement operator $(\bar{\cdot})$[1] and the conjunction operator $(\wedge)$. For n variables, there are in total 2^n minterms. For $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{F}_2^n$, the minterm $\bigwedge_{i=0}^{n-1} \overline{x_i \oplus u_i}$ is represented by $m_{\boldsymbol{u}}(\boldsymbol{x})$. Note that $\overline{x_i \oplus u_i}$ equals $\overline{x_i}$ if $u_i = \mathsf{0}$, and x_i otherwise. Also note that $m_{\boldsymbol{u}}(\boldsymbol{x}) = \mathsf{0}$ if and only if $\boldsymbol{x} = \boldsymbol{u}$. Based on the notion of minterm, the *canonical sum of products* (SOP) form of f is

$$f(\boldsymbol{x}) = \bigvee_{\boldsymbol{u} \in \mathrm{supp}(f)} m_{\boldsymbol{u}}(\boldsymbol{x}).$$

When the output of f is true, minterms are included in the canonical SOP, which is also referred to as the *sum of minterms*.

A *maxterm* over n Boolean variables is defined as a disjunction $(\vee)$ where each of the n variables occurs once (either in its complemented or uncomplemented form). There are 2^n maxterms over n variables. Denote the maxterm $\bigvee_{i=0}^{n-1} (x_i \oplus u_i)$ for the vector $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ as $M_{\boldsymbol{u}}(\boldsymbol{x})$. The canonical *product of sum* (POS) form of f is

$$f(x) = \bigwedge_{\boldsymbol{u} \in \overline{\mathrm{supp}(f)}} M_{\boldsymbol{u}}(\boldsymbol{x}), \tag{1}$$

where $\overline{\mathrm{supp}(f)} = \{\boldsymbol{x} \in \mathbb{F}_2^n \mid f(\boldsymbol{x}) = \mathsf{0}\}$ represents the complement of the support for the n-bit Boolean function f. It can be seen that $M_{\boldsymbol{u}}(\boldsymbol{x}) = \mathsf{0}$ if and only if $\boldsymbol{x} = \boldsymbol{u}$. Therefore, the creation of the canonical POS of f may be seen as the elimination of any vector over which f is false.

For a given function in canonical SOP or POS form, computing the *minimum SOP or POS form*, which implies the minimum number of disjunctions or conjunctions, is a well-researched issue in the design of digital logic gate circuits. Two conventional simplification techniques will be discussed in Sections 2.4 and 2.5, as they are closely related to our newly suggested encoding methods for large S-boxes in Section 5.

2.3 SAT Problem and POS Form Simplification

Mouha and Preneel [MP13] initiated the use of Boolean satisfiability problem (SAT) and satisfiability modulo theories (SMT) as a supporting tool in symmetric-key cryptography. The fundamental concept is to model the distinguisher searching issue in cryptanalysis as

[1]Note that the complement can only be applied to variables.

an SAT/SMT problem that can be addressed by SAT/SMT solvers such as STP [GD07] and Cryptominisat [SNC09].

All variables taken into account in the SAT problem are binary. A Boolean equation is a *clause* if it is a disjunction ($\vee$) of one or more (possibly negated) Boolean variables; a Boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction ($\wedge$) of one or more clauses. The CNF format is the input format that practically all SAT solvers claim to support. The essential step for the automatic search is thus to express the distinguisher searching issue using the CNF formula.

Since the SAT problem is NP-complete, all available solvers are only able to address problems of practical magnitude. The performance of the SAT solver may be affected if the SAT issue has an excessive number of clauses. As a result, reducing the number of clauses in the CNF format is a common issue when creating SAT problems regarding distinguisher search issues, particularly when constructing SAT models for S-boxes. It is observable that the CNF format of the SAT problem resembles the canonical POS form of a Boolean function. Prior research [AST+17, SWW18] has demonstrated that the POS simplification approach may be used to simplify the CNF format of the SAT problem.

2.4 Quine-McCluskey Algorithm

For a given function, the Quine-McCluskey algorithm is used to calculate the minimum SOP or POS. Quine [Qui52, Qui55] devised the method, while McCluskey [McC56] expanded upon it. The Quine-McCluskey method, whether dealing with SOP or POS, acts on the vector $\boldsymbol{u}$ in the minterm $m_{\boldsymbol{u}}(\boldsymbol{x})$ or the maxterm $M_{\boldsymbol{u}}(\boldsymbol{x})$ and is independent of the operator between variables. In the description that follows, we use the simplification of the SOP form as an illustration and remark that the simplification of the POS form may be accomplished in a similar manner. The Quine-McCluskey algorithm consists of two phases.

Table 1: Finding prime implicants of f_t.

Group	Minterm		Size 2 implicant		Size 4 implicant	
G_0	0000	$m_{\texttt{0x0}}$ (✓)	$000*$	$m_{\texttt{0x0}}, m_{\texttt{0x1}}$ (✓)	🟦 $*00*$	🟦 $m_{\texttt{0x0}}, m_{\texttt{0x1}}, m_{\texttt{0x8}}, m_{\texttt{0x9}}$
	-	-	$00*0$	$m_{\texttt{0x0}}, m_{\texttt{0x2}}$ (✓)	🟦 $*0*0$	🟦 $m_{\texttt{0x0}}, m_{\texttt{0x2}}, m_{\texttt{0x8}}, m_{\texttt{0xa}}$
	-	-	$*000$	$m_{\texttt{0x0}}, m_{\texttt{0x8}}$ (✓)	⬜ $*00*$	⬜ $m_{\texttt{0x0}}, m_{\texttt{0x8}}, m_{\texttt{0x1}}, m_{\texttt{0x9}}$
	-	-	-	-	⬜ $*0*0$	⬜ $m_{\texttt{0x0}}, m_{\texttt{0x8}}, m_{\texttt{0x2}}, m_{\texttt{0xa}}$
G_1	0001	$m_{\texttt{0x1}}$ (✓)	🟦 $0*01$	🟦 $m_{\texttt{0x1}}, m_{\texttt{0x5}}$	🟦 $**10$	🟦 $m_{\texttt{0x2}}, m_{\texttt{0x6}}, m_{\texttt{0xa}}, m_{\texttt{0xe}}$
	0010	$m_{\texttt{0x2}}$ (✓)	$*001$	$m_{\texttt{0x1}}, m_{\texttt{0x9}}$ (✓)	⬜ $**10$	⬜ $m_{\texttt{0x2}}, m_{\texttt{0xa}}, m_{\texttt{0x6}}, m_{\texttt{0xe}}$
	1000	$m_{\texttt{0x8}}$ (✓)	$0*10$	$m_{\texttt{0x2}}, m_{\texttt{0x6}}$ (✓)	-	-
	-	-	$*010$	$m_{\texttt{0x2}}, m_{\texttt{0xa}}$ (✓)	-	-
	-	-	$100*$	$m_{\texttt{0x8}}, m_{\texttt{0x9}}$ (✓)	-	-
	-	-	$10*0$	$m_{\texttt{0x8}}, m_{\texttt{0xa}}$ (✓)	-	-
G_2	0101	$m_{\texttt{0x5}}$ (✓)	🟦 $01*1$	🟦 $m_{\texttt{0x5}}, m_{\texttt{0x7}}$	-	-
	0110	$m_{\texttt{0x6}}$ (✓)	🟦 $011*$	🟦 $m_{\texttt{0x6}}, m_{\texttt{0x7}}$	-	-
	1001	$m_{\texttt{0x9}}$ (✓)	$*110$	$m_{\texttt{0x6}}, m_{\texttt{0xe}}$ (✓)	-	-
	1010	$m_{\texttt{0xa}}$ (✓)	$1*10$	$m_{\texttt{0xa}}, m_{\texttt{0xe}}$ (✓)	-	-
G_3	0111	$m_{\texttt{0x7}}$ (✓)	-	-	-	-
	1110	$m_{\texttt{0xe}}$ (✓)	-	-	-	-

🟦 Prime implicants of f_t. ⬜ Duplicated prime implicants.

Phase 1: finding all prime implicants. We use the vector $\boldsymbol{u}$ to represent the minterm $m_{\boldsymbol{u}}(\boldsymbol{x})$ in this subsection for simplicity. In this phase, we attempt to merge two or more minterms. If two minterms only vary by one bit, that bit may be substituted with an asterisk ($*$) to indicate that it is arbitrary. This allows two minterms to be represented by a single string. As an example, two 4-bit minterms $m_{\texttt{0x1}}$ and $m_{\texttt{0x5}}$ may be combined to get $\texttt{0}{*}\texttt{01}$, which is referred to as an *implicant* of size two since it represents two minterms with no constraint on the second bit. Two implicants of size two may be merged further to yield an implicant of size four. In this process, matching asterisks comes first. Then, two implicants that vary by one bit in the remaining positions may be merged.

To locate all prime implicants, it is necessary to compare and combine all feasible pairings of minterms wherever possible. Grouping minterms according to their Hamming weight decreases the number of comparisons. Suppose the minterms of a given function are split into ϱ groups $\mathsf{G}_0, \mathsf{G}_1, \ldots, \mathsf{G}_{\varrho-1}$, and the Hamming weight of minterms in group G_i is i for $0 \leqslant i \leqslant \varrho - 1$. Consequently, there is no need to compare minterms of nonadjacent groups, since they will always vary in more than one variable. Similarly, comparing minterms within one group is unnecessary as each minterm has the same Hamming weight and any two minterms vary in at least two variables. Therefore, only the minterms of neighbouring groups need to be compared.

A checkmark (✔) is placed next to a minterm after it has been coupled with another minterm to signify that it has been included into an implicant. Thereafter, by merging implicants of size two, implicants of size four are created. Once again, a checkmark is put next to an implicant that may be paired with another implicant. In general, this procedure should be repeated until no more implicant combinations can be made. After that, the unchecked minterms and implicants are known as *prime implicants*. Since each minterm is contained in at least one prime implicant, the original function equals the disjunction of its prime implicants. Despite the fact that the canonical SOP has been reduced to some degree, the outcome is often not the minimum SOP. To do this, we should complete the second phase. Example 1 provides a tiny illustration of this phase.

Phase 2: determining the minimum SOP. First, a *prime implicant chart* is constructed in this phase. Along the side are the newly created prime implicants, and along the top are the minterms of the supplied function. If the prime implicant in the row covers the minterm in the column, the symbol (🔔) is inserted into the table cell. Then, we search for columns containing a single symbol (🔔). If a column contains just one symbol, the minterm can only be covered by a single prime implicant, which is termed as an *essential prime implicant*. The minimum SOP expression must include all essential prime implicants. In certain instances, the essential prime implicants do not cover all minterms, necessitating the use of extra chart reduction processes. Petrick's method [Pet56] should be a more methodical approach, as opposed to trial and error. Using Petrick's approach, all minimum SOP solutions are derived from a prime implicant chart. This methodology is described algorithmically in Algorithm 1.

Remark 1. Boura and Coggia [BC20] offered a novel formulation for the first phase of the Quine-McCluskey algorithm. By substituting the second phase of the Quine-McCluskey algorithm with a minimisation problem in MILP, an improved algorithm for creating MILP models for S-boxes was created. Although the MILP optimiser does not always find the optimal solution, the output provided by the optimiser might be satisfactory. See Section 3.3.1 and [BC20] for more information.

We present a tiny example to easily comprehend the Quine-McCluskey algorithm.

Example 1. The target is a 4-bit Boolean function f_t, and the support is $\mathrm{supp}(f_t) = \{\texttt{0x0}, \texttt{0x1}, \texttt{0x2}, \texttt{0x5}, \texttt{0x6}, \texttt{0x7}, \texttt{0x8}, \texttt{0x9}, \texttt{0xa}, \texttt{0xe}\}$. Therefore, the canonical SOP form of f_t contains ten minterms. As shown in Table 1, the Hamming weight is used to classify the

Algorithm 1 Petrick's method

```
1: Delete the rows for essential prime implicants and their associated columns with the
   symbol (🔔) to reduce the size of the prime implicant chart.
2: Mark the rows of the reduced prime implicant chart as $P_0$, $P_1$, $P_2$, etc.
3: Create a logical function $P$ consisting of the conjunction of disjunctions. Each disjunc-
   tion relates to a column, and the $i$-th disjunction is $(P_{i_0} \vee P_{i_1} \vee \cdots \vee P_{i_k})$, where each
   $P_{i_j}$ represents a row covering the $i$-th column.
4: Multiply out the function $P$ and use the absorption rule $P_i \vee (P_i \wedge P_j) = P_i$ to reduce
   it to a minimum SOP. Each term in the reduced expression corresponds to a set of
   prime implicants including all the minterms in the reduced prime implicant chart.
5: Find each term of the reduced expression of $P$ that has the smallest number of prime
   implicants. For each term containing a minimum number of prime implicants, count
   the number of variables in each prime implicant and calculate the overall number of
   variables.
6: Choose the term or terms that include the minimum total variables and write down
   the conjunction of prime implicants that corresponds.
```

minterms into four groups G_0, G_1, G_2, and G_3. To generate implicants of size two, we begin with group G_0 and compare all of its minterms to those of group G_1. $m_{\texttt{0x0}}$ and $m_{\texttt{0x1}}$ may be joined to generate 000∗ by replacing the final bit with an asterisk. A checkmark (✔) is then put next to the two minterms to indicate that they have been included into an implicant. $m_{\texttt{0x0}}$ and $m_{\texttt{0x2}}$ are then joined to make 00∗0, and $m_{\texttt{0x0}}$ and $m_{\texttt{0x8}}$ form ∗000. After repeating this process for $(\mathsf{G}_1, \mathsf{G}_2)$ and $(\mathsf{G}_2, \mathsf{G}_3)$, we get a list of implicants with a size of two.

Subsequently, implicants of size four are derived by merging implicants of size two, and the result can be found in Table 1. Again, a checkmark (✔) is put next to an implicant when it may be paired with another implicant. In this example, no more combinations of implicants of size four are possible. Note that three implicants with a size of four are duplicates generated by merging the same set of four minterms in different orders. The following analysis will not include duplicates. From Table 1, we observe that f_t has six prime implicants: 0∗01, 01∗1, 011∗, ∗00∗, ∗0∗0, and ∗∗10.

Table 2: Prime implicant chart of f_t.

Prime implicant \ Minterm	$m_{\texttt{0x0}}$	$m_{\texttt{0x1}}$	$m_{\texttt{0x2}}$	$m_{\texttt{0x5}}$	$m_{\texttt{0x6}}$	$m_{\texttt{0x7}}$	$m_{\texttt{0x8}}$	$m_{\texttt{0x9}}$	$m_{\texttt{0xa}}$	$m_{\texttt{0xe}}$
0∗01	-	🔔	-	🔔	-	-	-	-	-	-
01∗1	-	-	-	🔔	-	🔔	-	-	-	-
011∗	-	-	-	-	🔔	🔔	-	-	-	-
∗00∗	🔔	🔔	-	-	-	-	🔔	🔔	-	-
∗0∗0	🔔	-	🔔	-	-	-	🔔	-	🔔	-
∗∗10	-	-	🔔	-	🔔	-	-	-	🔔	🔔

🟩 Essential prime implicant rows.

⬜ Minterm columns covered by essential prime implicants.

With the prime implicants, the prime implicant chart of f_t can be constructed (cf. Table 2). Observe that the columns $m_{\texttt{0x9}}$ and $m_{\texttt{0xe}}$ only contain a single symbol (🔔). ∗00∗ and ∗∗10 are hence two essential prime implicants. Table 3 displays the reduced prime implicant chart for f_t, which is derived by deleting the two essential prime implicant rows and the corresponding columns with the symbol (🔔). The three rows in Table 3 are designated P_0, P_1, and P_2 respectively. According to Algorithm 1, the logical function P is generated as

$$P = (P_0 \vee P_1) \wedge (P_1 \vee P_2) = (P_0 \wedge P_1) \vee P_1 \vee (P_0 \wedge P_2) \vee (P_1 \wedge P_2) = P_1 \vee (P_0 \wedge P_2).$$

Consequently, P_1 (01∗1), which corresponds to the expression $\overline{x_0} \wedge x_1 \wedge x_3$, is the term of P having the fewest prime implicants. Together with the two essential prime implicants ∗00∗ and ∗∗10, the minimum SOP for f_t should be

$$f_t(\boldsymbol{x}) = (\overline{x_1} \wedge \overline{x_2}) \vee (x_2 \wedge \overline{x_3}) \vee (\overline{x_0} \wedge x_1 \wedge x_3)$$

Table 3: Reduced prime implicant chart of f_t.

Prime implicant \ Minterm		$m_{\texttt{0x5}}$	$m_{\texttt{0x7}}$
P_0	0∗01	🔔	-
P_1	01∗1	🔔	🔔
P_2	011∗	-	🔔

2.5 ESPRESSO Logic Minimizer

The Quine-McCluskey method makes the creation of all prime implicants (Phase 1) more efficient. The number of prime implicants of an n-bit logic function, however, may be proven to equal $\mathcal{O}(3^n/n)$. In addition, the second phase attempts to solve a minimum covering problem that is known to be NP-complete. Since the number of components in the covering issue may be proportional to the exponential of the number of input variables of the logic function, the Quine-McCluskey approach becomes unworkable for even moderately large situations. For instance, Abdelkhalek et al. [AST$^+$17] failed to implement the Quine-McCluskey method to reduce a 16-bit function pertaining to AES [DR02]. Various heuristic approaches to the issue were prompted by the need to simplify large-scale functions.

These heuristic procedures use two routes. One method utilises the structure of the traditional method by first producing all prime implicants. However, rather than creating a minimum cover, a near minimum cover is heuristically determined. This approach still has the potential to produce a very large quantity of prime implicants. The second method attempts to simultaneously detect and choose implicants (who are not necessarily prime) for the cover. A lot of follow-up work centered on the second approach, and the first and most successful was the IBM programme MINI [HCO74]. Later, Brown [Bro81] proposed a method called PRESTO. The core of MINI and PRESTO is expanding each implicant of the supplied function and eliminating implicants that are covered by the expanded implicant. MINI expands each implicant to its maximum size both in the input and output parts of the function. PRESTO expands the input part of each implicant to its maximum size, but then reduces the output parts of the implicants maximally by removing the implicants from as many output spaces as possible. In both methods, implicants are expanded and covered, and thus handling the problem of creation and storage of all minterms.

Nonetheless, this primary technique results in a local minimum that may be far off from the global minimum. In order to prevent this issue, following the first expansion and removal of covered implicants, MINI reduces the remaining implicants to their smallest size while preserving the cover of the function. The implicants are then assessed in pairs for reshaping, with one implicant being enlarged and the other being reduced by the same set of minterms. The expansion process is then resumed, and the whole method is repeated until there is no further decrease in the number of product terms. This metric enables the investigation of wider portions of the optimisation space, with the goal of achieving better results at the cost of increased computation time. PRESTO differs from MINI in the way the expansion process is carried out. MINI generates the complement of the logic function to check is the expansion of an implicant does not change the cover of the function. PRESTO avoids the initial cost of computing the complement, but then the input expansion process requires a check on whether all the minterms covered by the expanded implicant are covered by someother implicant of the cover, which, in general, costs more computation time.

Comparing the different tactics adopted by MINI and PRESTO in a specific context inspired the development of ESPRESSO. The first version ESPRESSO-I was created by Brayton *et al.* at IBM [BHH+82]. ESPRESSO-I can enhance some of the algorithms employed by MINI and PRESTO due to its experimenting with logic modification. On the basis of comparisons and improved algorithms from ESPRESSO-I, ESPRESSO-II [BHMS84] was developed; it essentially follows the sequence of top-level transformations of iterated expansion-reduction pioneered by MINI. ESPRESSO has been implemented as a regular minimisation step for logic functions into practically all modern logic synthesis tools. An algorithmic description for ESPRESSO logic minimizer can be found in Algorithm 2. A modern compiliable re-host of the original source code can be found at `https://github.com/classabbyamp/espresso-logic`.

Algorithm 2 ESPRESSO logic minimizer

```
1: [Complement] Compute the complement of the function.
2: [Expand] Expand each implicant into a prime and remove covered implicants.
3: [Extract] Extract essential prime implements and store them in a set.
4: [Irredundant Cover] Find a minimal (optionally minimum) irredundant cover.
5: [Reduce] Reduce each implicant to a minimum essential implicant.
6: [Iterate] Iterate [Expand], [Irredundant Cover], and [Reduce] until no improvement.
7: [Lastgasp] Try [Reduce], [Expand], and [Irredundant Cover] one last time using a
   different strategy. If successful, continue the iteration.
8: [Makesparse] Include the essential prime implicant back into the cover and make the
   output as sparse as possible.
```

3 MILP Modelling Progress for Large S-boxes

In this part, we review the progress made in the MILP modelling of large S-boxes.

3.1 First Bit-Oriented Model for S-boxes

Sun *et al.* [SHW+14b] proposed the first bit-oriented model in the search with MILP approach. Each bit of the inner state should have a binary variable allocated to it. The simplest model for an S-box is to describe its $*$-DDT, which is a reduced form of the DDT in which all non-zero entries are substituted with 1. In this model, we are solely concerned with the feasibility of differential propagation, and a propagation is deemed feasible if its corresponding item in $*$-DDT equals 1. With this notation, two sets

$$\begin{aligned}\mathcal{F} &= \{\boldsymbol{x}\|\boldsymbol{y} \mid \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation}, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n\}, \\ \mathcal{I} &= \{\boldsymbol{x}\|\boldsymbol{y} \mid \boldsymbol{x} \to \boldsymbol{y} \text{ is an impossible propagation}, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n\}\end{aligned} \quad (2)$$

can be constructed for an n-bit S-box. $*$-DDT can be modelled by employing linear inequalities to either characterise all vectors in $\mathcal{F}$ or exclude all vectors in $\mathcal{I}$. These two ideas correspond to the two approaches proposed in [SHW+14b].

3.1.1 H-representation of the Convex Hull

Note that $\mathcal{F}$ is a subset of $\mathbb{R}^{2n}$, where $\mathbb{R}$ is the real number field, and that its *convex hull* is defined as the smallest convex set that contains $\mathcal{F}$. A convex hull can be represented as the common solutions of a finite number of linear inequalities; this is known as the *H-representation* of a convex hull. Using the `inequality_generator()` function in the `sage.geometry.polyhedron` class of the open-source mathematical software SageMath[2] [The22], it is possible to derive the convex hull. Given that the number of inequalities in the H-representation of a convex hull is typically quite high, Sun *et al.* developed a *greedy* technique to reduce the number of inequalities. The greedy algorithm builds a collection of valid cutting-off inequalities by picking at each step an inequality from the convex hull that maximises the number of impossible differential propagations removed from the current feasible region.

3.1.2 Logical Condition Modelling

The main idea of logical condition modelling is that CNF/POS formulas are directly expressible as inequalities in MILP. This method may be thought of as focusing on the set $\mathcal{I}$ and generating linear inequalities to eliminate impossible differential propagations one by one. Specifically, for every vector $\boldsymbol{a}\|\boldsymbol{b} = a_0\| \cdots \|a_{n-1}\|b_0\| \cdots \|b_{n-1}$ in the set $\mathcal{I}$, an inequality about $2n$ Boolean variables $\boldsymbol{x}\|\boldsymbol{y} = x_0\| \cdots \|x_{n-1}\|y_0\| \cdots \|y_{n-1}$ is generated

$$\sum_{i=0}^{n-1}(-1)^{a_i} \cdot x_i + \sum_{i=0}^{n-1}(-1)^{b_i} \cdot y_i + \sum_{i=0}^{n-1} a_i + \sum_{i=0}^{n-1} b_i \geqslant 1. \tag{3}$$

Only when $\boldsymbol{x}\|\boldsymbol{y} = \boldsymbol{a}\|\boldsymbol{b}$ does the inequality become invalid. Applying this method directly will yield an excessive amount of inequalities. To overcome this issue, Sun *et al.* found that certain S-boxes contain conditional differential features, allowing us to rule out many impossible propagations with a single inequality. If the input difference of the 4-bit S-box of `PRESENT` [BKL+07] is `0x9`, for example, the least significant bit of the output difference must be `0`. Then, eight impossible propagations `1001` $\nrightarrow$ $***1$ are eliminated by the linear inequality $-x_0 + x_1 + x_2 - x_3 - y_3 + 2 \geqslant 0$. However, such conditional differential properties do not present in all circumstances.

When transitioning from $*$-DDT to DDT in a model, additional variables must be included to encode differential probabilities. [SHW+14a] provides an example of how to describe the differential probability of the S-box of `PRESENT`. Observe that all applications in [SHW+14b, SHW+14a] are ciphers with at most 6-bit S-boxes. [SHW+14a] also stated that the MILP model is mostly applicable to lightweight ciphers. In [SGL+17], the authors claim unequivocally that the MILP method is incapable of searching for true differential characteristics of 8-bit S-box ciphers.

3.2 Modelling for Large S-boxes

In response to the limitations of previous MILP models for large S-boxes, Abdelkhalek *et al.* [AST+17] suggested a novel modelling method for large S-boxes oriented to $*$-DDT and differential probabilities. The new method is founded on logical condition modelling introduced in Section 3.1.2, in which the number of linear inequalities is minimised or kept as small as possible.

[2]Note that SageMath is a front-end software. The method comes from the classic literature [GKPS67, Bro83, Zie07, GO04].

3.2.1 Modelling ∗-DDT of Large S-boxes

The technique commences with a description of ∗-DDT. A $2n$-bit Boolean function f is built using the ∗-DDT, and $f(\boldsymbol{x}\|\boldsymbol{y})$ equals one if and only if $\boldsymbol{x}\|\boldsymbol{y} \in \mathcal{F} = \mathbb{F}_2^{2n} \setminus \mathcal{I}$. The canonical POS form of f is

$$f(\boldsymbol{x}\|\boldsymbol{y}) = \bigwedge_{\boldsymbol{a}\|\boldsymbol{b}\in\mathcal{I}} \left(\bigvee_{i=0}^{n-1} (x_i \oplus a_i) \vee \bigvee_{i=0}^{n-1} (y_i \oplus b_i) \right).$$

Observe that $f(\boldsymbol{x}\|\boldsymbol{y})$ is non-zero only when for all $\boldsymbol{a}\|\boldsymbol{b} \in \mathcal{I}$, the following expression about $\boldsymbol{x}\|\boldsymbol{y}$ holds

$$\bigvee_{i=0}^{n-1} (x_i \oplus a_i) \vee \bigvee_{i=0}^{n-1} (y_i \oplus b_i) = 1 \tag{4}$$

Since 1-bit XOR $x_i \oplus a_i$ can be represented as $x_i + a_i - 2 \cdot x_i \cdot a_i$, the equation in Equation (4) is identical to the expression in Equation (3). In contrast to the technique employed in [SHW+14b] to reduce the number of inequalities by monitoring conditional differential characteristics, Abdelkhalek *et al.* asserted that lowering the number of inequalities is akin to simplifying the POS form of f, a well-studied subject. The Quine-McCluskey algorithm can be used to determine the minimum POS, which ensures the minimum number of linear inequalities under logical condition modelling. Since the algorithm becomes inapplicable for even moderately large problems, Abdelkhalek *et al.* were unable to apply it for reducing a 16-bit function corresponding to the ∗-DDT of `AES`. In this instance, the heuristic algorithm ESPRESSO was suggested for simplifying large-scale functions so that the number of linear inequalities is maintained as small as possible.

3.2.2 Modelling DDT of Large S-boxes

Abdelkhalek *et al.* split the entries of a DDT into numerous pieces in order to encode the differential probability and control the scale of the Boolean function, from which the concept of p-DDT is generated.

Definition 1 (p-DDT, [AST+17])**.** For a given S-box and its DDT, the corresponding entry of the p-DDT is one if the probability of entry in the DDT is p; otherwise, it is zero.

By applying the approach to model ∗-DDT, one set of linear inequalities $\mathbb{L}_p$ is obtained for each p-DDT, where

$$\mathbb{L}_p = \left\{ \sum_{i=0}^{n-1} a_i^{\langle p|j\rangle} \cdot x_i + \sum_{i=0}^{n-1} b_i^{\langle p|j\rangle} \cdot y_i \geqslant c^{\langle p|j\rangle}, j = 0, 1, \ldots, \ell_p - 1 \right\},$$

and ℓ_p represents the number of linear inequalities for describing the p-DDT. In addition, a binary variable Q_p is allocated to each p-DDT to determine if the linear inequalities in $\mathbb{L}_p$ should be included in the solution phase. Specifically, the linear inequalities in $\mathbb{L}_p$ are only applicable when $Q_p = 1$, and they are disregarded otherwise. This technique is realised by adding a suitably large integer[3] M and transforming each linear inequality in $\mathbb{L}_p$ into a conditional inequality as

$$\sum_{i=0}^{n-1} a_i^{\langle p|j\rangle} \cdot x_i + \sum_{i=0}^{n-1} b_i^{\langle p|j\rangle} \cdot y_i + M(1 - Q_p) \geqslant c^{\langle p|j\rangle}.$$

[3] $M = 2n$ is enough.

When $Q_p = 0$, $\boldsymbol{x}\|\boldsymbol{y}$ can take any value; otherwise, the conditional constraint reverts to the original linear inequality. To ensure that the automatic searching model imposes no more than one set of linear inequalities for a certain p-DDT, no more than one Q_p may take on a non-zero value. Consequently, an additional binary variable Q is allocated to each S-box, and $Q = 1$ if and only if the S-box is active. The constraint

$$Q = \sum_p Q_p$$

establishes the restriction for Q_p. Consequently, the base-2 logarithm of the differential probability for a single S-box is calculated to be

$$\sum_p \log_2(p) \cdot Q_p.$$

3.3 Efficient Modelling for Large S-boxes

Despite the fact that Abdelkhalek *et al.* enabled the description of large S-boxes, the model for the $*$-DDT of the S-box in AES consists of 8302 linear inequalities, which is sometimes too cumbersome for practical usage. Boura and Coggia [BC20] offered two methods for developing efficient algorithms for modelling 8-bit S-boxes with the fewest number of linear inequalities. These approaches are conceptually related to logical condition modelling (cf. Section 3.1.2), in which an inequality is generated to eliminate an impossible vector. The contrast lies in the fact that the new approaches aim to remove more impossible vectors, which often possess algebraic structures, using a single inequality.

3.3.1 Generalised Logical Condition Modelling

Given a vector $\boldsymbol{u} \in \mathbb{F}_2^m$, the set $\{\boldsymbol{x} \in \mathbb{F}_2^m \mid \boldsymbol{x} \preceq \boldsymbol{u}\}$ is denoted as $\mathrm{Prec}(\boldsymbol{u})$, where $\boldsymbol{x} \preceq \boldsymbol{u}$ means that $x_i \leqslant u_i$ for all $0 \leqslant i \leqslant m-1$. The first method derives simple inequalities to eliminate vector spaces of the type $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$ using the following proposition.

Proposition 1 ([BC20])**.** *For two vectors $\boldsymbol{a}$ and $\boldsymbol{u}$ in $\mathbb{F}_2^m$ satisfying $\mathrm{supp}(\boldsymbol{a}) \cap \mathrm{supp}(\boldsymbol{u}) = \emptyset$, denote E as the set $\{0, 1, \ldots, m-1\} \backslash (\mathrm{supp}(\boldsymbol{a}) \cup \mathrm{supp}(\boldsymbol{u}))$. Then, for all $\boldsymbol{x} \in \mathbb{F}_2^m$,*

$$-\sum_{i \in \mathrm{supp}(\boldsymbol{a})} x_i + \sum_{i \in E} x_i + \sum_{i=0}^{m-1} a_i \geqslant 1 \text{ if and only if } \boldsymbol{x} \notin \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}).$$

Algorithm 2 in [BC20] determines, given the set of impossible propagations $\mathcal{I}$ (cf. Equation (2)), all subsets of the type $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$ that are not subsets of others. For each $\boldsymbol{a} \in \mathcal{I}$, spaces $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \subset \mathcal{I}$ are generated by incrementally increasing the weight of $\boldsymbol{u}$, where $\boldsymbol{u}$ should satisfy $\boldsymbol{a} \oplus \boldsymbol{u} \in \mathcal{I}$ and $\mathrm{supp}(\boldsymbol{a}) \cap \mathrm{supp}(\boldsymbol{u}) = \emptyset$, and determining, for all $\boldsymbol{v} \preceq \boldsymbol{u}$ and $\mathrm{wt}(\boldsymbol{v}) = \mathrm{wt}(\boldsymbol{u}) - 1$, if $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{v})$ has previously been designated as a subset of $\mathcal{I}$. This algorithm, according to Boura and Coggia, is closely connected to the first step of the Quine-McCluskey algorithm. By applying this algorithm, an initial set of inequalities is constructed, with a total of 70336 for the $*$-DDT of AES. To select a representative set from the initial inequalities, Boura and Coggia formulated a minimisation problem in MILP according to the method described in [ST17] (cf. Section 3.4.1) and solved it using the Groubi optimiser [Gur22]. Although the optimiser did not achieve the minimum for the S-box of AES even after more than twenty days, the solution supplied by the optimiser is much superior than that returned by ESPRESSO in [AST+17]. The number of inequities falls from 8302 to 7461.

3.3.2 Modelling S-boxes with Inequalities Issued from Balls

The second method creates inequalities to exclude points from a ball $\mathcal{B}(r, \boldsymbol{c})$ whose definition is shown below.

Definition 2 ([BC20])**.** A ball of $\mathbb{F}_2^m$ of radius r centred at $\boldsymbol{c} \in \mathbb{F}_2^m$ is the set of all points whose Hamming distance from the center $\boldsymbol{c}$ is no more than r, i.e., $\mathcal{B}(r, \boldsymbol{c}) = \{\boldsymbol{x} \in \mathbb{F}_2^m \mid \mathrm{wt}(\boldsymbol{x} \oplus \boldsymbol{c}) \leqslant r\}$. Furthermore, the sphere $\mathcal{S}(r, \boldsymbol{c})$ of the ball $\mathcal{B}(r, \boldsymbol{c})$ is defined as the set of points $\boldsymbol{x} \in \mathcal{B}(r, \boldsymbol{c})$ with $\mathrm{wt}(\boldsymbol{x} \oplus \boldsymbol{c}) = r$.

With the following proposition, all points in the ball $\mathcal{B}(r, \boldsymbol{c})$ may be eliminated.

Proposition 2 ([BC20])**.** *For $\boldsymbol{c} \in \mathbb{F}_2^m$,*

$$\sum_{i=0}^{m-1} (1 - c_i) \cdot x_i + c_i \cdot (1 - x_i) \geqslant r + 1 \textit{ if and only if } \boldsymbol{x} \notin \mathcal{B}(r, \boldsymbol{c}).$$

Since the set of possible propagations $\mathcal{F}$ (cf. Equation (2)) is not sparse for S-boxes with a low differential uniformity, the number of balls consisting purely of impossible propagations is normally rather modest, therefore deleting complete balls is frequently ineffective. In this instance, it is feasible to create *distorted balls* in which any possible propagations at the sphere are eliminated. The following proposition provides instructions for making distorted balls.

Proposition 3 ([BC20])**.** *Let $\mathcal{B}(r, \boldsymbol{c}) \subset \mathbb{F}_2^m$ be a ball of radius r from which we remove the set of points $\mathcal{Q} = (\boldsymbol{c} \oplus \mathrm{Prec}(\boldsymbol{q})) \cap \mathcal{S}(r, \boldsymbol{c})$ for some $\boldsymbol{q} \in \mathbb{F}_2^m$. The vectors in $\mathcal{Q}$ represent possible propagations towards the sphere of the ball. We define $\boldsymbol{a} \in \mathbb{Q}^m$, where $\mathbb{Q}$ is the rational number field, so that $a_i = \frac{r+1}{r}$ if $q_i = 1$ and $a_i = 1$ otherwise. Then,*

$$\sum_{i=0}^{m-1} a_i \left[(1 - c_i) \cdot x_i + c_i \cdot (1 - x_i)\right] \geqslant r + 1 \textit{ if and only if } \boldsymbol{x} \notin \mathcal{B}(r, \boldsymbol{c}) \setminus \mathcal{Q}.$$

Boura and Coggia proposed a way to remove all points from three distorted balls with a radius of one using a single inequality in order to obtain better candidates for the initial set of inequalities; this innovative notion is used in Algorithm 3 of [BC20]. After constructing an initial set of inequalities with Algorithm 3, a minimisation problem in MILP is created which is then resolved by the Groubi optimiser. The number of inequalities reaches 2882, despite the fact that the optimisation for the S-box of `AES` did not terminate after a few days. In terms of the amount of inequalities, the proposed solution is considerably superior to the ESPRESSO-simplified version [AST+17]. Notably, despite the creation of improved MILP models for the S-boxes of `AES` and `SKINNY-128`, these models are not employed to analyse the differential characteristics of the two ciphers. The authors used the new models to investigate the resistances of the two ciphers against the impossible differential attack [BBS99].

3.4 Modelling by Minimum Number of Inequalities

Whether reducing the number of inequalities for the S-box can reduce the runtime to solve the MILP problem is a debate from the pioneer work [SHW+14b, SHW+14a]. As a result, modelling S-boxes with the fewest inequalities is a hot topic for research in the area of automatic search. Numerous studies have been conducted to investigate this matter.

3.4.1 Reduction Algorithm for Small S-boxes Based on MILP

The research of Sasaki and Todo [ST17] is able to overcome the limitations of greedy algorithm in terms of locating the fewest inequalities. This technique is derived from the H-representation supplied by SageMath. Assume that the H-representation comprises $\hbar$ linear inequalities designated L_0, L_1, …, $\mathsf{L}_{\hbar-1}$. Selecting inequalities from the H-representation is transformed into a MILP problem by the reduction algorithm.

For each inequality L_i, the reduction method introduces one binary variable ℓ_i, $i = 0, 1, \dots \hbar - 1$. $\ell_i = 1$ indicates that the inequality L_i is employed in the reduced system, whereas $\ell_i = 0$ implies that L_i is not used. Then, as in Equation (2), the set $\mathcal{I}$ of impossible propagation is constructed. Assuming $\mathcal{I}$ consists of Λ vectors labelled $\boldsymbol{I}_0$, $\boldsymbol{I}_1$, …, $\boldsymbol{I}_{\Lambda-1}$. For each vector $\boldsymbol{I}_j$ in $\mathcal{I}$, the inequalities that can exclude $\boldsymbol{I}_j$ are computed, and the indices of the inequalities that satisfy this requirement are arranged in the set $\text{Index}_j \subset \{0, 1, \dots, \hbar - 1\}$. To guarantee that each impossible propagation is eliminated by at least one inequality, the following Λ constraints

$$\sum_{i \in \text{Index}_j} \ell_i \geqslant 1, \text{ for all } 0 \leqslant j \leqslant \Lambda - 1$$

must be imposed to the MILP problem. By setting the objective function of the MILP problem to minimise the value of $\sum_{i=0}^{\hbar-1} \ell_i$, the least amount of inequalities is assured.

Sasaki and Todo demonstrated empirically, via the reduction algorithm, that minimising the number of inequalities does not reduce the execution time of the MILP optimiser.

3.4.2 Finding Smallest MILP Models

Although minimising the number of inequalities does not always reduce the runtime of the MILP optimiser, many subsequent studies continue to seek a breakthrough on the number of inequalities. The task of discovering the smallest MILP models might be of theoretical importance in its own right. For further detail, interested readers might see [LS22, Udo21].

4 SAT/SMT Modelling Progress for S-boxes

Mouha and Preneel [MP13] introduced the use of SAT/SMT in symmetric-key cryptography for the search of distinguisher. Initially, this technique was used to ARX ciphers [KLT15, LWR16, SHY16]. Two studies utilising the SAT/SMT approach to search for differential characteristics appeared around the same time [AK18, SWW18]. [SWW21] is also associated with the SAT-based search for differential characteristics. However, the focus of the study was on modifying the objective function to achieve acceleration for the SAT method, and the modelling for S-boxes adheres to the method described in [SWW18].

4.1 Logical Condition Modelling

The S-box modelling approach described by Ankele and Kölbl [AK18] is comparable to the logical condition modelling using MILP method (cf. Section 3.1.2). The clause

$$\bigvee_{i=0}^{n-1} (x_i \oplus a_i) \vee \bigvee_{i=0}^{n-1} (y_i \oplus b_i) = 1$$

is built for each vector $\boldsymbol{a} \| \boldsymbol{b} = a_0 \| \cdots \| a_{n-1} \| b_0 \| \cdots \| b_{n-1}$ in $\mathcal{I}$ and then added to the SAT problem. This expression is invalid if and only if $\boldsymbol{x} \| \boldsymbol{y} = \boldsymbol{a} \| \boldsymbol{b}$. Adding all clauses thereby removes all impossible propagations. Although the authors claim that their approach is compatible with 8-bit S-boxes, all of the ciphers analysed in [AK18] employ 4-bit S-boxes.

4.2 Logical Condition Modelling Using Simplification

Sun *et al.* [SWW18] suggested a modelling technique with simplification, which was inspired by the work of Abdelkhalek *et al.* [AST$^+$17] in MILP approach. We first review the model oriented to active S-boxes.

4.2.1 Modelling Oriented to Active S-boxes

The model for linear active S-boxes is a natural extension of the model for differential active S-boxes, which we apply as an example. Apart from $2n$ Boolean variables $\boldsymbol{x} = (x_0, x_1, \dots, x_{n-1})$ and $\boldsymbol{y} = (y_0, y_1, \dots, y_{n-1})$ to represent the input and output differences of an n-bit S-box, there should be an auxiliary Boolean variable w. The value of w for active S-boxes is one and zero for inactive S-boxes under the premise that $\boldsymbol{x} \to \boldsymbol{y}$ is a possible propagation. Limiting the total number of active S-boxes for the differential characteristic is equal to setting a maximum for the total sum of auxiliary variables for all S-boxes in the characteristic. $\Pr(\boldsymbol{x} \to \boldsymbol{y})$ represents the differential probability of propagation $\boldsymbol{x} \to \boldsymbol{y}$. According to this criteria, possible values for $\boldsymbol{x}\|\boldsymbol{y}\|w$ fall into the set

$$\mathcal{F}_{\langle n,n,1\rangle} = \left\{ \boldsymbol{x}\|\boldsymbol{y}\|w \;\middle|\; \begin{array}{l} \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n, w \in \mathbb{F}_2, \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation} \\ w = \begin{cases} 0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 1 \\ 1, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) < 1 \end{cases} \end{array} \right\},$$

where the three components in the subscript of $\mathcal{F}$ are, in order, the number of input bits of the S-box, the number of output bits of the S-box, and the number of auxiliary variables involved. The direct method removes impossible vectors one by one with clauses

$$\bigvee_{i=0}^{n-1} (x_i \oplus \tau_i) \vee \bigvee_{i=0}^{n-1} (y_i \oplus \tau_{n+i}) \vee (w \oplus \tau_{2\cdot n}) = 1, \text{ for all } \boldsymbol{\tau} = (\tau_0, \tau_1, \dots, \tau_{2\cdot n}) \notin \mathcal{F}_{\langle n,n,1\rangle}.$$

The number of clauses in the resulting SAT problem will be extremely high if the scale of the set $\mathbb{F}_2^{2n+1} \backslash \mathcal{F}_{\langle n,n,1\rangle}$ is enormous, which could negatively affect the performance of the SAT solver. Finding a solution to reduce the clauses is thus necessary.

Sun *et al.* [SWW18] stated that the conjunction of $|\mathbb{F}_2^{2n+1} \backslash \mathcal{F}_{\langle n,n,1\rangle}|$ clauses is a canonical POS form of the function

$$f_{\langle n,n,1\rangle}(\boldsymbol{x}\|\boldsymbol{y}\|w) = \begin{cases} 1, & \text{if } \boldsymbol{x}\|\boldsymbol{y}\|w \in \mathcal{F}_{\langle n,n,1\rangle} \\ 0, & \text{otherwise} \end{cases},$$

drawing their inspiration from the idea in [AST$^+$17]. Finding a simpler POS form of $f_{\langle n,n,1\rangle}$ is the equivalent of lowering the amount of clauses, and this problem can be handled using the Quine-McCluskey and ESPRESSO algorithms. Commercial tools, like Logic Friday [Ric11] with ESPRESSO as the underlying program, can handle the simplification for small-scale problems. Disjunctions decoded from the simplified POS form of $f_{\langle n,n,1\rangle}$ is a SAT model focused on differential active S-boxes.

4.2.2 Modelling Oriented to Differential Probabilities and Linear Correlations

We only introduce modelling for differential probabilities because the methods for constructing SAT models for linear correlations and differential probabilities are comparable. Note that additional auxiliary variables are required to encode probability information. Given an n-bit S-box, we first construct a set $\mathcal{P}$ consisting of all probabilities for possible differential propagation $\boldsymbol{x} \to \boldsymbol{y}$. The weight of a possible propagation with a probability of p is $-\log_2(p)$ and can take on non-integer values. For non-integral weights, the set $\mathcal{D} = \{\lceil \log_2(p) \rceil - \log_2(p) \mid \log_2(p) \text{ is not an integer, and } p \in \mathcal{P}\}$ contains all feasible decimal values. Assume there are ν items in the set $\mathcal{D}$, which are denoted by d_0, d_1, …, $d_{\nu-1}$. The plan is to introduce two groups of auxiliary variables that are used to encode the integral and decimal components of the weight, respectively. To be precise, $\max\{\lfloor -\log_2(p)\rfloor \mid p \in \mathcal{P}\} \triangleq \mu$ variables $(u_0, u_1, \ldots, u_{\mu-1}) \triangleq \boldsymbol{u}$ are introduced for the integral portion and ν variables $(v_0, v_1, \ldots, v_{\nu-1}) \triangleq \boldsymbol{v}$ are introduced for the decimal portion. Similarly to the scenario concerning active S-boxes, we subsequently determine the possible values for the vector $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v}$. If $\boldsymbol{x} \to \boldsymbol{y}$ is a possible propagation with a probability of p, then $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v}$ assignments must fulfil the equation $\sum_{i=0}^{\mu-1} u_i + \sum_{i=0}^{\nu-1} d_i \cdot v_i = -\log_2(p)$. An optional set of possible values for $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v}$ is

$$\mathcal{F}_{\langle n,n,\mu+\nu\rangle} = \left\{ \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v} \;\middle|\; \begin{array}{l} \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n, \boldsymbol{u} \in \mathbb{F}_2^\mu, \boldsymbol{v} \in \mathbb{F}_2^\nu \\ \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation with probability being } p \\ \boldsymbol{u} = 0\|0\|\cdots\|0\| \underbrace{1\|1\|\cdots\|1}_{\lfloor -\log_2(p)\rfloor \text{ bits}} \\ \boldsymbol{v} = \begin{cases} 0\|0\|\cdots\|0, & \text{if } -\log_2(p) \text{ is an integer} \\ \boldsymbol{e}_l, & \text{if } \lceil \log_2(p)\rceil - \log_2(p) = d_l \in \mathcal{D} \end{cases} \end{array} \right\},$$

where $\boldsymbol{e}_l$ is a unit vector in $\mathbb{F}_2^\nu$ and the l-th bit is the only non-zero bit. Again, $\mathcal{F}_{\langle n,n,\mu+\nu\rangle}$ can be considered as the support of the function

$$f_{\langle n,n,\mu+\nu\rangle}(\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v}) = \begin{cases} 1, & \text{if } \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v} \in \mathcal{F}_{\langle n,n,\mu+\nu\rangle} \\ 0, & \text{otherwise} \end{cases}.$$

The SAT model for differential probabilities can be derived by simplifying the canonical POS form of $f_{\langle n,n,\mu+\nu\rangle}$.

Example 2. Consider a 4-bit S-box whose DDT contains the values 0, 2, 4, 6, and 16. The corresponding set of probability for all feasible differential propagations is $\{2^{-3}, 2^{-2}, 2^{-1.415}, 1\}$. Thus, we claim three variables $(u_0, u_1, u_2) \triangleq \boldsymbol{u}$ to represent the integral portion of the weight and one variable v_0 to represent the decimal portion of the weight. The optional set of possible values for $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|v_0$ is

$$\left\{ \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|v_0 \;\middle|\; \begin{array}{l} \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^4, \boldsymbol{u} \in \mathbb{F}_2^3, v_0 \in \mathbb{F}_2, \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation} \\ \boldsymbol{u}\|v_0 = \begin{cases} 1\|1\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3} \\ 0\|1\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2} \\ 0\|0\|1\|1, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-1.415} \\ 0\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 1 \end{cases} \end{array} \right\}.$$

The vector in the set confirms the equation $u_0 + u_1 + u_2 + 0.415 \cdot v_0 = -\log_2(p)$ for a possible propagation $\boldsymbol{x} \to \boldsymbol{y}$ with a probability of p.

4.3 SMT Modelling for Large S-boxes

Note that the modelling techniques described in [AK18, SWW18] mostly apply to ciphers with 4-bit S-boxes. Liu *et al.* [LLL+19] were the first to introduce the SMT modelling technique for 8-bit S-boxes. The effectiveness of SMT modelling is dependent on the reasoning statement supplied by the SMT solver STP [GD07].

In addition to two n-bit vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ representing the input and output differences of the S-box, the SMT model requires a 1-bit vector w for the search of valid differential characteristics. The reasoning statement

$$\texttt{ASSERT}\,((\boldsymbol{x} = \boldsymbol{a})\ \texttt{\&\&}\ (\boldsymbol{y} = \boldsymbol{b})\ \texttt{=>}\ (w = 1))$$

is added to the SMT model for each vector $\boldsymbol{a}\|\boldsymbol{b}$ in $\mathcal{F}$ where $\boldsymbol{a} \to \boldsymbol{b}$ represents a possible propagation. This phrase specifies that w is assigned the value 1 if $\boldsymbol{x}\|\boldsymbol{y}$ is a member of the set $\mathcal{F}$. Then, for each vector $\boldsymbol{a}\|\boldsymbol{b}$ in $\mathcal{I}$, the reasoning statement

$$\texttt{ASSERT}\,((\boldsymbol{x} = \boldsymbol{a})\ \texttt{\&\&}\ (\boldsymbol{y} = \boldsymbol{b})\ \texttt{=>}\ (w = 0))$$

is added to the SMT model, resulting in $w = 0$ if $\boldsymbol{x}\|\boldsymbol{y}$ takes values from the set $\mathcal{I}$. The SMT model also asserts that $w = 1$ so that any impossible propagations for the S-box are ignored by the solver.

SMT modelling for S-boxes oriented to differential probabilities and linear correlations is comparable to the previously described fundamental model. The SMT model employs a direct way to eliminate all impossible propagations and does not involve simplification.

5 Fast SAT Models for Large S-boxes

As the reduction of canonical POS is NP-complete, it is typically inefficient to generate reduced sets of clauses for big S-boxes (e.g., 8-bit S-boxes) oriented to differential probabilities and linear correlations. Thus, there has been no SAT model[4] for large S-boxes till now. In this section, three fast SAT modellings for large S-boxes oriented to differential probabilities and linear correlations are proposed. Although these approaches cannot guarantee a minimum number of clauses, the time required to acquire clauses is significantly reduced.

The 8-bit S-box $\mathcal{S}_8$ (cf. Table 5) of `SKINNY-128` [BJK+16] will serve as an illustration for the presentation of the subsequent three methods. The set

$$\{2^{-7}, 2^{-6}, 2^{-5.415}, 2^{-5}, 2^{-4.415}, 2^{-4}, 2^{-3.678}, 2^{-3.415}, 2^{-3.193}, 2^{-3}, 2^{-2.678}, 2^{-2.415}, 2^{-2}, 1\}$$

consists of the probabilities of possible differential propagations $\boldsymbol{x} \to \boldsymbol{y}$ for $\mathcal{S}_8$. Following the method in [SWW18], seven variables $(u_0, u_1, \ldots, u_6) \triangleq \boldsymbol{u}$ are then introduced to describe the integral portion of the weight. To encode the decimal portion of the weight, three variables $(v_0, v_1, v_2) \triangleq \boldsymbol{v}$ are used, with $v_0 = 1$ (resp., $v_1 = 1$, $v_2 = 1$) if and only if the decimal portion of the weight equals 0.678 (resp., 0.415, 0.193). The option for the set of possible values for $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v}$ is

$$\mathcal{F}_{\langle 8,8,10\rangle} = \left\{ \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v} \;\middle|\; \begin{array}{l} \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^8, \boldsymbol{u} \in \mathbb{F}_2^7, \boldsymbol{v} \in \mathbb{F}_2^3, \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation} \\ \boldsymbol{u}\|\boldsymbol{v} = \begin{cases} 1\|1\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-7} \\ 0\|1\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-6} \\ 0\|0\|1\|1\|1\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-5.415} \\ 0\|0\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-5} \\ 0\|0\|0\|1\|1\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-4.415} \\ 0\|0\|0\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-4} \\ 0\|0\|0\|0\|1\|1\|1\|1\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3.678} \\ 0\|0\|0\|0\|1\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3.415} \\ 0\|0\|0\|0\|1\|1\|1\|0\|0\|1, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3.193} \\ 0\|0\|0\|0\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3} \\ 0\|0\|0\|0\|0\|1\|1\|1\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2.678} \\ 0\|0\|0\|0\|0\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2.415} \\ 0\|0\|0\|0\|0\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2} \\ 0\|0\|0\|0\|0\|0\|0\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 1 \end{cases} \end{array} \right\}.$$

[4]Note that it is fairly easy to construct large models (e.g., using the approach described in Section 4.1), but they are not useful.

The vector in the set verifies the equation $\sum_{i=0}^{6} u_i + 0.678 \cdot v_0 + 0.415 \cdot v_1 + 0.193 \cdot v_2 = -\log_2(p)$ for a propagation $\boldsymbol{x} \to \boldsymbol{y}$ that is possible with probability p. $\mathcal{F}_{\langle 8,8,10 \rangle}$ can be viewed as the support for a 26-bit Boolean function $f_{\langle 8,8,10 \rangle}$. We attempted to use ESPRESSO to identify a simplification for the canonical POS form of $f_{\langle 8,8,10 \rangle}$, but after *more than a hundred days*, nothing was returned. Therefore, we wish to discover an alternate method for constructing a SAT model for $\mathcal{S}_8$ that demonstrates differential probability. All of the experiments in this study are done on a 16-core AMD EPYC™ 7302 processor , of which only one core is utilised. Although the subsequent approaches are demonstrated using models for differential probabilities, the creation of SAT models for linear correlation may be accomplished in a similar fashion.

5.1 Trade-off Between Level of Simplification and Time

The initial attempt is centred on ESPRESSO itself. We notice that ESPRESSO provides multiple options and commands for minimisation and creates a trade-off between simplification level and execution time. Listed below are several options that may reduce the runtime.

`-efast` This option stops ESPRESSO after the first [`Expand`] and [`Irredundant Cover`] operations (cf. Algorithm 2) and prevents it from iterating over the solution.

`-eness` With this setting, essential prime implicants will not be identified.

`-enirr` With this option, the result will not necessarily be made irredundant in the last step which removes redundant literals.

`-eonset` This option recalculates the support of the input function prior to minimisation, which is advantageous when the canonical POS form includes a large number of maxterms.

Due to the heuristic nature of these options, the number of clauses in the result may not be the minimal. However, in the vast majority of applications, the number of clauses is acceptable for modern SAT solvers. ESPRESSO is used to conduct the simplification of $f_{\langle 8,8,10 \rangle}$ with the four newly introduced options. The simplification is only accomplished with the option `-eonset`. There are 820 clauses in the output, and the total execution time is 3521.42 seconds. Although the remaining three options claim that certain procedures are skipped and the simplification may be faster, the programmes do not terminate after *fifty days*. Consequently, the first fast SAT model for large S-boxes leverages the `-eonset` option of ESPRESSO. The applicability of this strategy is also examined for the S-box of PIPO [KJK+20], which is a simplification issue for a 25-bit Boolean function. Again, only the `-eonset` option delivers the output after 3769.86 seconds, and the simplified result has 6066 clauses.

5.2 Two-Step Encoding Method

Noting that the complexity of simplification increases exponentially with the number of input variables, we wondered whether the size of the function may be decreased to alleviate the difficulty of developing the SAT model. Consequently, the encoding mechanism for auxiliary variables must be modified.

Initially, we examine the set $\mathcal{F}_{\langle 8,8,10 \rangle}$ and notice that u_5 and u_6 always have the same value. Therefore, it is logical to reduce the number of auxiliary variables for the integral portion to six, i.e., $\boldsymbol{u} \in \mathbb{F}_2^6$, and reconstruct the set of possible values for $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v}$ as

$$\mathcal{F}_{\langle 8,8,9\rangle} = \left\{ \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{u}\|\boldsymbol{v} \;\middle|\; \begin{array}{l} \boldsymbol{x},\boldsymbol{y} \in \mathbb{F}_2^8, \boldsymbol{u} \in \mathbb{F}_2^6, \boldsymbol{v} \in \mathbb{F}_2^3, \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation} \\ \boldsymbol{u}\|\boldsymbol{v} = \begin{cases} 1\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-7} \\ 0\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-6} \\ 0\|0\|1\|1\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-5.415} \\ 0\|0\|1\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-5} \\ 0\|0\|0\|1\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-4.415} \\ 0\|0\|0\|1\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-4} \\ 0\|0\|0\|0\|1\|1\|1\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3.678} \\ 0\|0\|0\|0\|1\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3.415} \\ 0\|0\|0\|0\|1\|1\|0\|0\|1, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3.193} \\ 0\|0\|0\|0\|1\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-3} \\ 0\|0\|0\|0\|0\|1\|1\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2.678} \\ 0\|0\|0\|0\|0\|1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2.415} \\ 0\|0\|0\|0\|0\|1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 2^{-2} \\ 0\|0\|0\|0\|0\|0\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = 1 \end{cases} \end{array} \right\}.$$

As a result, the evaluation of the weight for a feasible propagation is modified using $\sum_{i=0}^{5} u_i + u_5 + 0.678 \cdot v_0 + 0.415 \cdot v_1 + 0.193 \cdot v_2$. The set $\mathcal{F}_{\langle 8,8,9\rangle}$ is then treated as the support for the 25-bit function $f_{\langle 8,8,9\rangle}$, and ESPRESSO is used to simplify it under various options. `-eonset` is the only option that returns the result after 1883.51 seconds; 818 clauses comprise the simplification. Comparatively to the simplification for $f_{\langle 8,8,10\rangle}$, the time required to achieve the solution is decreased although the number of clauses is almost the same.

Then, we evaluate the feasibility of reducing the size of the function received by ESPRESSO further. The main idea is dividing the encoding phase for an n-bit S-box into two steps. In addition to the auxiliary variables $\boldsymbol{u}$ and $\boldsymbol{v}$ introduced regarding the integral and decimal parts of the weight, we claim a set of *chaining variables* $\boldsymbol{z}$.

In the first step, we map the various probabilities to distinct values of the vector $\boldsymbol{z}$. We recall that $\mathcal{P}$ represents the set containing all probabilities for possible differential propagations and denote the elements in $\mathcal{P}$ as $\{p_0, p_1, \dots, p_{|\mathcal{P}|-1}\}$. Thus, the number of required chaining variables is $\lceil \log_2(|\mathcal{P}|) \rceil \triangleq \zeta$. To be specific, we introduce ζ variables $(z_0, z_1, \dots, z_{\zeta-1}) = \boldsymbol{z}$. An option for the set of possible values for the vector $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{z}$ is

$$\mathcal{F}^{(1)}_{\langle n,n,\zeta\rangle} = \left\{ \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{z} \;\middle|\; \begin{array}{l} \boldsymbol{x},\boldsymbol{y} \in \mathbb{F}_2^n, \boldsymbol{z} \in \mathbb{F}_2^\zeta, \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation} \\ \boldsymbol{z} = i \text{ if } \Pr(\boldsymbol{x} \to \boldsymbol{y}) = p_i \text{ for all } 0 \leqslant i \leqslant |\mathcal{P}| - 1 \end{array} \right\}.$$

In the second step, we create a connection between chaining variables $\boldsymbol{z}$ and auxiliary variables $\boldsymbol{u}\|\boldsymbol{v}$ so that $\sum_{i=0}^{\mu-1} u_i + \sum_{i=0}^{\nu-1} d_i \cdot v_i$ can correctly compute the weight of the propagation. An option for the set of possible values for the vector $\boldsymbol{z}\|\boldsymbol{u}\|\boldsymbol{v}$ is

$$\mathcal{F}^{(2)}_{\langle \zeta,\mu+\nu\rangle} = \left\{ \boldsymbol{z}\|\boldsymbol{u}\|\boldsymbol{v} \;\middle|\; \begin{array}{l} \boldsymbol{z} \in \mathbb{F}_2^\zeta, \boldsymbol{u} \in \mathbb{F}_2^\mu, \boldsymbol{v} \in \mathbb{F}_2^\nu, \text{if } \boldsymbol{z} = i \\ \boldsymbol{u} = 0\|0\|\cdots\|0\| \underbrace{1\|1\|\cdots\|1}_{\lfloor -\log_2(p_i) \rfloor \text{ bits}} \\ \boldsymbol{v} = \begin{cases} 0\|0\|\cdots\|0, & \text{if } -\log_2(p_i) \text{ is an integer} \\ \boldsymbol{e}_l, & \text{if } \lceil \log_2(p_i) \rceil - \log_2(p_i) = d_l \in \mathcal{D} \end{cases} \end{array} \right\},$$

where the two components in the subscript of $\mathcal{F}$ are the number of chaining variables and the number of auxiliary variables for the integral and decimal parts of the weight.

Thus, with these two steps, we convert the simplification of a large-scale function into the simplification of two relatively small-scale functions $f^{(1)}_{\langle n,n,\zeta\rangle}$ and $f^{(2)}_{\langle \zeta,\mu+\nu\rangle}$, whose supports are $\mathcal{F}^{(1)}_{\langle n,n,\zeta\rangle}$ and $\mathcal{F}^{(2)}_{\langle \zeta,\mu+\nu\rangle}$, respectively.

We return to the S-box $\mathcal{S}_8$ of `SKINNY-128`. In first step, we introduce four chaining variables $(z_0, z_1, z_2, z_3) \triangleq \boldsymbol{z}$ and create an option for the set of possible values for $\boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{z}$ as

$$\mathcal{F}^{(1)}_{\langle 8,8,4\rangle} = \left\{ \boldsymbol{x}\|\boldsymbol{y}\|\boldsymbol{z} \;\middle|\; \begin{array}{l} \boldsymbol{x},\boldsymbol{y} \in \mathbb{F}_2^8, \boldsymbol{z} \in \mathbb{F}_2^4, \boldsymbol{x} \to \boldsymbol{y} \text{ is a possible propagation} \\ \boldsymbol{z} = \begin{cases} 1\|1\|0\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-7} \\ 1\|1\|0\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-6} \\ 1\|0\|1\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-5.415} \\ 1\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-5} \\ 1\|0\|0\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-4.415} \\ 1\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-4} \\ 0\|1\|1\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-3.678} \\ 0\|1\|1\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-3.415} \\ 0\|1\|0\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-3.193} \\ 0\|1\|0\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-3} \\ 0\|0\|1\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-2.678} \\ 0\|0\|1\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-2.415} \\ 0\|0\|0\|1, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 2^{-2} \\ 0\|0\|0\|0, & \text{if } \Pr(\boldsymbol{x}\to\boldsymbol{y}) = 1 \end{cases} \end{array} \right\}.$$

In the second step, we create the connection between $\boldsymbol{z}$ and $\boldsymbol{u}\|\boldsymbol{v}$. An option for the set of possible values for $\boldsymbol{z}\|\boldsymbol{u}\|\boldsymbol{v}$ is

$$\mathcal{F}^{(2)}_{\langle 4,10\rangle} = \left\{ \boldsymbol{z}\|\boldsymbol{u}\|\boldsymbol{v} \;\middle|\; \begin{array}{l} \boldsymbol{z} \in \mathbb{F}_2^4, \boldsymbol{u} \in \mathbb{F}_2^7, \boldsymbol{v} \in \mathbb{F}_2^3 \\ \boldsymbol{u}\|\boldsymbol{v} = \begin{cases} 1\|1\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \boldsymbol{z} = 1\|1\|0\|1 \\ 0\|1\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \boldsymbol{z} = 1\|1\|0\|0 \\ 0\|0\|1\|1\|1\|1\|1\|0\|1\|0, & \text{if } \boldsymbol{z} = 1\|0\|1\|1 \\ 0\|0\|1\|1\|1\|1\|1\|0\|0\|0, & \text{if } \boldsymbol{z} = 1\|0\|1\|0 \\ 0\|0\|0\|1\|1\|1\|1\|0\|1\|0, & \text{if } \boldsymbol{z} = 1\|0\|0\|1 \\ 0\|0\|0\|1\|1\|1\|1\|0\|0\|0, & \text{if } \boldsymbol{z} = 1\|0\|0\|0 \\ 0\|0\|0\|0\|1\|1\|1\|1\|0\|0, & \text{if } \boldsymbol{z} = 0\|1\|1\|1 \\ 0\|0\|0\|0\|1\|1\|1\|0\|1\|0, & \text{if } \boldsymbol{z} = 0\|1\|1\|0 \\ 0\|0\|0\|0\|1\|1\|1\|0\|0\|1, & \text{if } \boldsymbol{z} = 0\|1\|0\|1 \\ 0\|0\|0\|0\|1\|1\|1\|0\|0\|0, & \text{if } \boldsymbol{z} = 0\|1\|0\|0 \\ 0\|0\|0\|0\|0\|1\|1\|1\|0\|0, & \text{if } \boldsymbol{z} = 0\|0\|1\|1 \\ 0\|0\|0\|0\|0\|1\|1\|0\|1\|0, & \text{if } \boldsymbol{z} = 0\|0\|1\|0 \\ 0\|0\|0\|0\|0\|1\|1\|0\|0\|0, & \text{if } \boldsymbol{z} = 0\|0\|0\|1 \\ 0\|0\|0\|0\|0\|0\|0\|0\|0\|0, & \text{if } \boldsymbol{z} = 0\|0\|0\|0 \end{cases} \end{array} \right\}.$$

$\mathcal{F}^{(1)}_{\langle 8,8,4\rangle}$ can be seen as the support of a 20-bit function $f^{(1)}_{\langle 8,8,4\rangle}$, and $\mathcal{F}^{(2)}_{\langle 4,10\rangle}$ can be viewed as the support of a 14-bit function $f^{(2)}_{\langle 4,10\rangle}$. Simplifying $f^{(1)}_{\langle 8,8,4\rangle}$ and $f^{(2)}_{\langle 4,10\rangle}$ is much easier than reducing $f_{\langle 8,8,10\rangle}$ and $f_{\langle 8,8,9\rangle}$. For small-scale functions, ESPRESSO provides two commands, which are listed in the following, to obtain potentially better result.

`-estrong` This option substitutes [`Lastgasp`] with the alternative strategy [`Supergasp`] (cf. Algorithm 2) , which is more expensive but sometimes yields superior results.

-Dexact This command executes the precise minimisation procedure, which guarantees the smallest possible amount of clauses and heuristically minimises the number of literals. It might potentially be costly.

We apply different options in ESPRESSO to simplify $f^{(1)}_{\langle 8,8,4\rangle}$ and $f^{(2)}_{\langle 4,10\rangle}$, and the result is listed in Table 4. Options, such as `-efast`, `-eness` and `-enirr`, which were previously inapplicable for $f_{\langle 8,8,10\rangle}$ and $f_{\langle 8,8,9\rangle}$, can now be used to simplify $f^{(1)}_{\langle 8,8,4\rangle}$. The time required to simplify $f^{(1)}_{\langle 8,8,4\rangle}$ with the `-eonset` option is significantly shorter than the time required to simplify $f_{\langle 8,8,10\rangle}$ or $f_{\langle 8,8,9\rangle}$. The two-step encoding approach is still highly efficient when the total time is comprised of the simplifications for $f^{(1)}_{\langle 8,8,4\rangle}$ and $f^{(2)}_{\langle 4,10\rangle}$. In addition, the amount of clauses for the two-step method is 787 when both functions are simplified with the `-eonset` option, which is fewer than the 820 (resp., 818) clauses that correspond to the simplified result for $f_{\langle 8,8,10\rangle}$ (resp., $f_{\langle 8,8,9\rangle}$).

Table 4: Results for the simplification of $f^{(1)}_{\langle 8,8,4\rangle}$ and $f^{(2)}_{\langle 4,10\rangle}$.

Option	$f^{(1)}_{\langle 8,8,4\rangle}$		$f^{(2)}_{\langle 4,10\rangle}$	
	The number of clauses	Runtime	The number of clauses	Runtime
`Null`	757	87359.76s	30	1.39s
`-efast`	839	98018.76s	32	1.37s
`-eness`	757	95035.71s	30	1.46s
`-enirr`	757	92729.09s	30	1.4s
`-eonset`	757	98.83s	30	0.14s
`-estrong`	730	101050.85s	28	1.45s
`-Dexact`	-	> 60 days	28	0.16s

`Null`: There is no option used in the implementation of ESPRESSO.

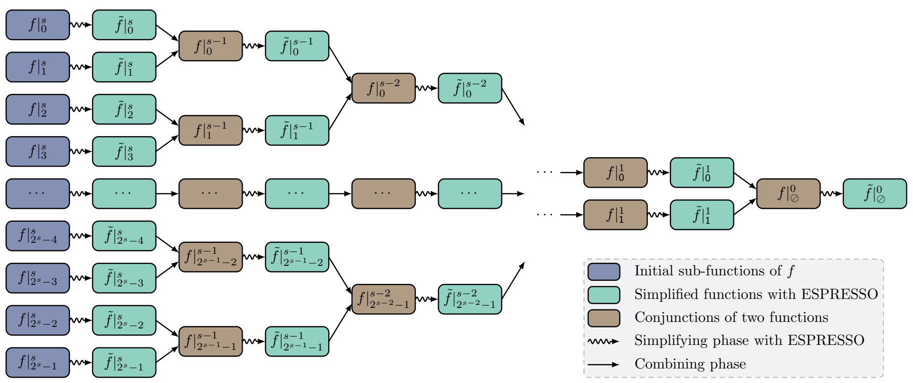

Initial sub-functions of f
Simplified functions with ESPRESSO
Conjunctions of two functions
Simplifying phase with ESPRESSO
Combining phase

Figure 1: Iterative simplification.

5.3 Simplifying by Partitioning Method

In the test regarding ESPRESSO, we find that the simplification of a large-scale function is not difficult if the number of maxterms in the function is not excessively huge. This observation is the basis for the third method.

Definition 3. Given a set $\mathcal{X}$, a family of sets Ψ is a *partition* of $\mathcal{X}$ if and only if the following conditions are met.

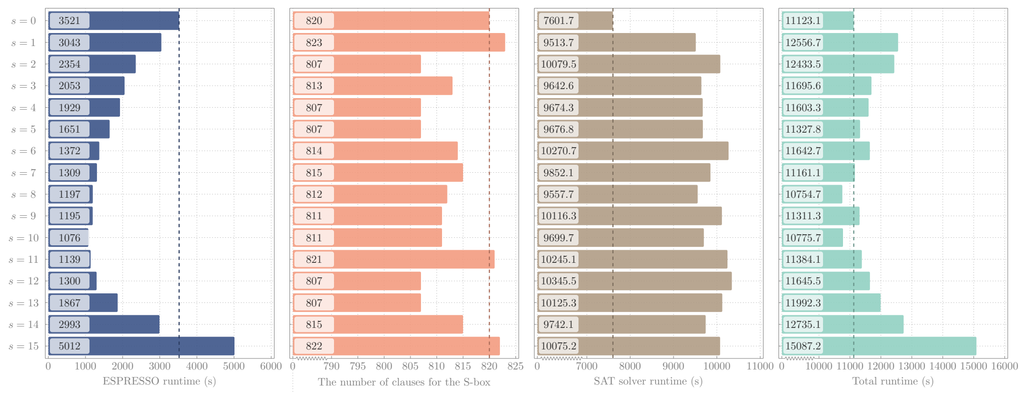
- The family Ψ does not contain the empty set.
- $\mathcal{X}$ is equal to the union of the sets contained in Ψ .
- In Ψ, the intersection of any two different sets is empty set.

Assume that $\Psi = \{\psi_0, \psi_1, \ldots, \psi_{\chi-1}\}$ is a partition of the set $\overline{\mathrm{supp}(f)}$, where $\mathrm{supp}(f)$ is the support of the n-bit function f in Equation (1). For each set ψ_i in Ψ, we define a function f_i with $\overline{\psi_i}$ as its support. With this definition, f can be expressed as

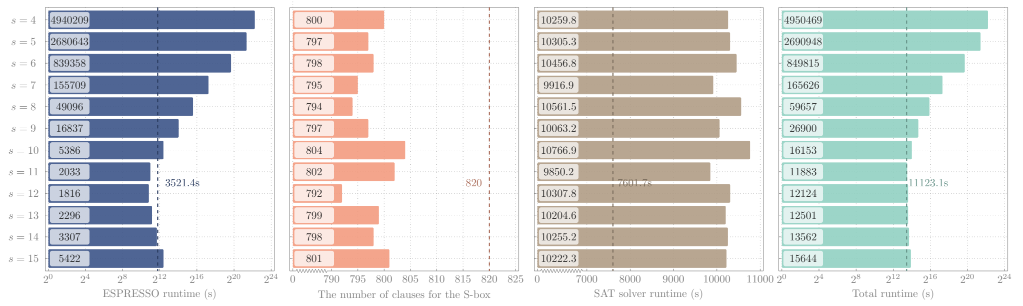
$$f(\boldsymbol{x}) = \bigwedge_{\boldsymbol{u}\in\overline{\mathrm{supp}(f)}} M_{\boldsymbol{u}}(\boldsymbol{x}) = \bigwedge_{i=0}^{\chi-1} \bigwedge_{\boldsymbol{u}\in\psi_i} M_{\boldsymbol{u}}(\boldsymbol{x}) = \bigwedge_{i=0}^{\chi-1} f_i(\boldsymbol{x}).$$

If a simplified representation $\tilde{f}_i$ with fewer disjunctions can be found for each f_i, then the conjunction $\bigwedge_{i=0}^{\chi-1} \tilde{f}_i$ yields a simplified form of f.

Next, we determine the partition of the set $\overline{\mathrm{supp}(f)}$ based on this insight. As demonstrated by the Quine-McCluskey technique, grouping maxterms by the Hamming weight of each term decreases the number of comparisons. Therefore, we hypothesise that the simplification could be made simpler if the maxterms in a given set share as many bit values as possible. To propose a general method of partition, we start with a partition of the set $\mathbb{F}_2^n$. Denote $\mathbb{F}_2^n[s|\mathring{\boldsymbol{x}}]$, where $0 < s \leqslant n$ and $\mathring{\boldsymbol{x}} \in \mathbb{F}_2^s$, a subset of $\mathbb{F}_2^n$, and



(a) Test results regarding the `-eonset` option.

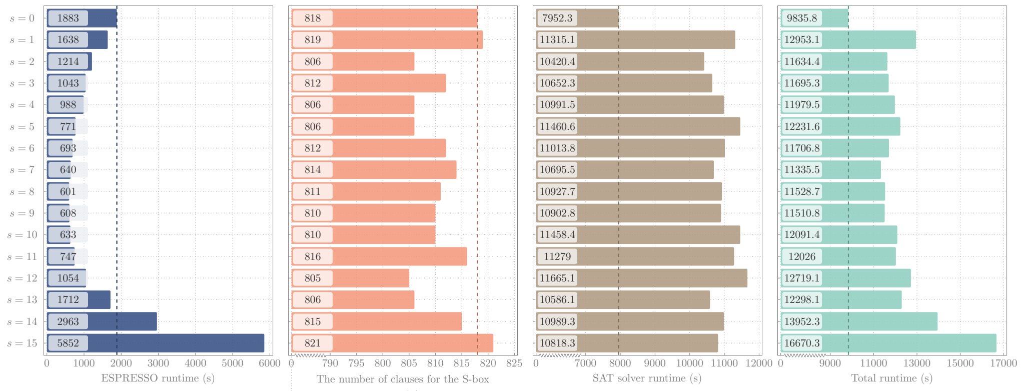


(b) Test results regarding the `-estrong` option.

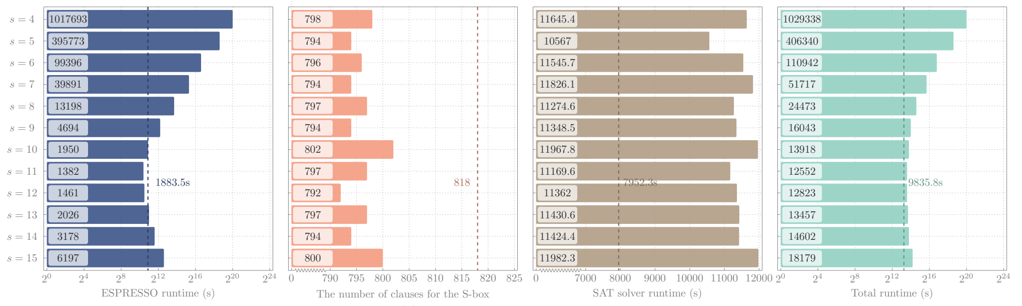
Figure 2: Iterative simplification for the 26-bit Boolean function $f_{\langle 8,8,10\rangle}$ regarding $\mathcal{S}_8$. $s = 0$ indicates that the partitioning method is not employed. Total runtime is the sum of ESPRESSO and SAT solver execution times. Note that for $s < 4$, the simplification with the `-estrong` option does not return the result after more than sixty days, making it infeasible.

the first s-bit value of the vectors $\boldsymbol{x}$ in $\mathbb{F}_2^n[s|\mathring{\boldsymbol{x}}]$ is equal to $\mathring{\boldsymbol{x}}$, i.e., $\boldsymbol{x} \gg (n-s) = \mathring{\boldsymbol{x}}$. The family of sets $\left\{\mathbb{F}_2^n[s|\mathring{\boldsymbol{x}}] \mid \mathring{\boldsymbol{x}} \in \mathbb{F}_2^s\right\}$ constitutes a partition of $\mathbb{F}_2^n$ with 2^s sets, and we use $\Psi^n_{\langle s\rangle}$ to denote this partition. Naturally, the partition $\Psi^n_{\langle s\rangle}$ restricted on the set $\Psi^n_{\langle s\rangle} \cap \overline{\mathrm{supp}(f)} = \left\{\mathbb{F}_2^n[s|\mathring{\boldsymbol{x}}] \cap \overline{\mathrm{supp}(f)} \,\middle|\, \mathring{\boldsymbol{x}} \in \mathbb{F}_2^s\right\}$ turns into a partition of $\overline{\mathrm{supp}(f)}$; for simplicity, the partition is written as $\Psi^n_{\langle s\rangle}\big|_f$, and the set $\mathbb{F}_2^n[s|\mathring{\boldsymbol{x}}] \cap \overline{\mathrm{supp}(f)}$ in the partition is denoted as $\Psi^n_{\langle s\rangle}\big|_f[\mathring{\boldsymbol{x}}]$.

According to the preceding analysis, the partition $\Psi^n_{\langle s\rangle}\big|_f$ permits the decomposition of the function f into 2^s sub-functions with support being $\overline{\Psi^n_{\langle s\rangle}\big|_f[\mathring{\boldsymbol{x}}]}$, which are abbreviated as $f|^s_{\mathring{\boldsymbol{x}}}$ for convenience, where $\mathring{\boldsymbol{x}} \in \mathbb{F}_2^s$. The combination of simplified expressions $\bigwedge_{\mathring{\boldsymbol{x}} \in \mathbb{F}_2^s} \tilde{f}|^s_{\mathring{\boldsymbol{x}}}$ for sub-functions is a simplification for f. Nonetheless, during the testing step, we notice that the number of clauses in the simplified form generated by this method is typically quite high. In order to further optimise the simplification for f, we employ an iterative approach.



(a) Test results regarding the `-eonset` option.



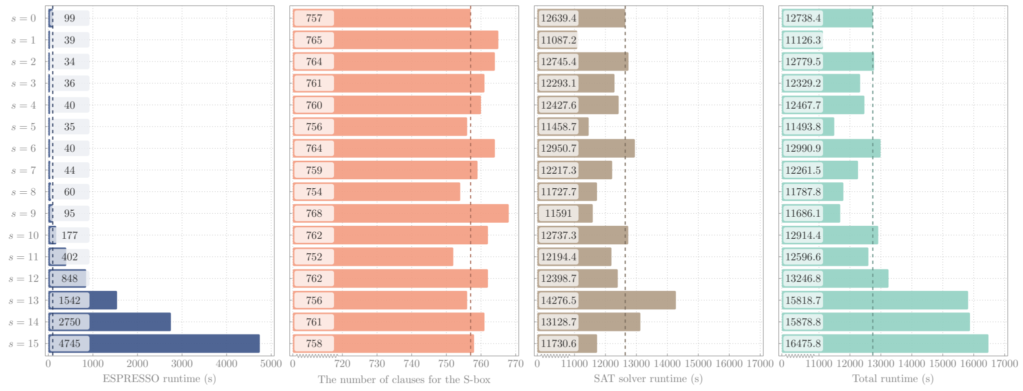
(b) Test results regarding the `-estrong` option.

Figure 3: Iterative simplification for the 25-bit Boolean function $f_{\langle 8,8,9\rangle}$ regarding $\mathcal{S}_8$. Note that for $s < 4$, the simplification with the `-estrong` option does not return the result after more than sixty days, making it infeasible.

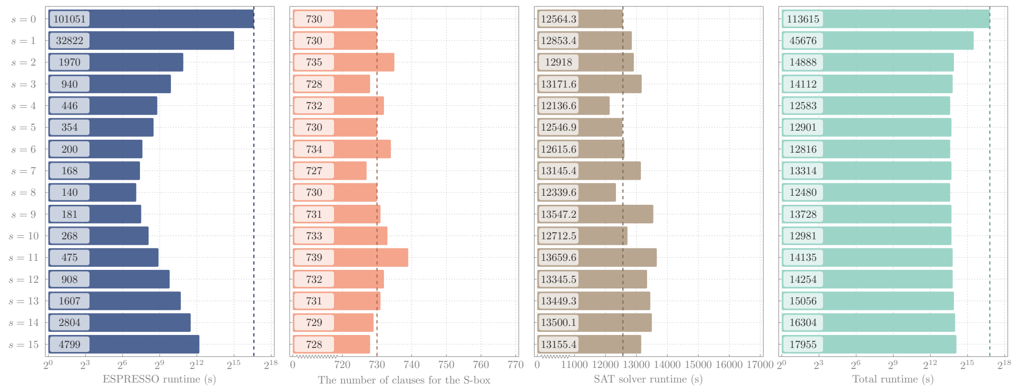
Figure 1 displays the core idea. After simplifying 2^s functions $f|^s_{\mathring{\boldsymbol{x}}}$, the simplified functions $\tilde{f}|^s_{\mathring{\boldsymbol{x}}}$ are combined in pairs. Specifically, if two vectors $\mathring{\boldsymbol{x}}_1$ and $\mathring{\boldsymbol{x}}_2$ differ in their last bit, the clauses of two functions $\tilde{f}|^s_{\mathring{\boldsymbol{x}}_1}$ and $\tilde{f}|^s_{\mathring{\boldsymbol{x}}_2}$ are concatenated into one function $f|^{s-1}_{\mathring{\boldsymbol{x}}_1 \gg 1}$. This yields a total of 2^{s-1} functions $f|^{s-1}_{\mathring{\boldsymbol{x}}}$ with $\mathring{\boldsymbol{x}} \in \mathbb{F}_2^{s-1}$. These functions are again simplified using ESPRESSO, and the results are merged in pairs to get 2^{s-2} functions $f|^{s-2}_{\mathring{\boldsymbol{x}}}$ with $\mathring{\boldsymbol{x}} \in \mathbb{F}_2^{s-2}$. This process is done until the clauses of two functions $\tilde{f}|^1_0$ and $\tilde{f}|^1_1$ are merged into a single function $f|^0_\oslash$. By using ESPRESSO to simplify the function $f|^0_\oslash$,

$\tilde{f}|^0_\emptyset$ is generated, which is a simplification of the original function f.

It can be imaged that the level of simplification of the final output and the runtime are affected by the number of components in the initial partition $\Psi^n_{\langle s\rangle}\big|_f$. To discover the feasibility of the iterative approach, we implement tests for the 26-bit Boolean function $f_{\langle 8,8,10\rangle}$ regarding $\mathcal{S}_8$. In the test, the value of s is traversed from 1 to 15, and the outcome of simplification using the `-eonset` option is depicted in Figure 2(a). From Figure 2(a), we find that the runtime shows a decrease followed by an increase with the increasing of s. When s is set to 10, the time required to obtain the simplification is 1076s, which is significantly shorter than the time required without the partition approach. Additionally, some values of s make it possible to achieve a simplification with fewer clauses. For instance, when s is set to 2, 4, 5, 12, or 13, 807 clauses are returned. Since the number of maxterms in $\tilde{f}|^s_{\boldsymbol{x}_1}$ is significantly fewer than that of f, the `-estrong` option can be used to simplify $f_{\langle 8,8,10\rangle}$ for certain values of s. As illustrated in Figure 2(b), it is possible to reduce the number of clauses even if the runtime may be exceptionally long. Notably, when s is set to 12, the result of employing the `-estrong` option is reduced to 792 clauses.



(a) Test results regarding the `-eonset` option.



(b) Test results regarding the `-estrong` option.

Figure 4: Iterative simplification for the 20-bit Boolean function $f^{(1)}_{\langle 8,8,4\rangle}$ regarding $\mathcal{S}_8$.

To illustrate that simplification by partition is a widely used technique, it is also applied to the 25-bit Boolean function $f_{\langle 8,8,9\rangle}$ and the 20-bit Boolean function $f^{(1)}_{\langle 8,8,4\rangle}$ with respect to `SKINNY-128`. As seen in Figure 3, the partition approach reduces the time required to get simplified clauses with the `-eonset` option, and when s is set to 12, the number of clauses reaches 805, which is fewer than the number of clauses that would be generated without partitioning. In addition, when $s \geqslant 4$, the `-estrong` option can be used, and the runtime is less than 1883.51 seconds when s is 11 or 12. In general, the number of clauses returned by the `-estrong` option is always smaller than the number returned by the `-eonset` option. The number of clauses reaches 792 when s equals 12. Figure 4 illustrates the outcome of simplifying $f^{(1)}_{\langle 8,8,4\rangle}$. Using the partition technique for simplification with the `-eonset` option might marginally reduce the time required to obtain the simplified clauses, as the runtime without partitioning is already quite quick. A benefit is that the number of clauses can be somewhat decreased. For simplifications utilising the `-estrong` option, the time required to achieve a simplified result is drastically decreased when compared to the simplification that do not employ the partition approach. When s is set to 7, the execution time is lowered from 101051 seconds to 168 seconds, and the number of clauses is decreased from 730 to 727. These examples demonstrate that the partition approach is effective not just for simplifying large-scale functions, but also for simplifying small-scale functions with the `-estrong` option.

6 Tight Probability Bound for 14 Rounds of `SKINNY-128`

In this section, we will first review the specifications of `SKINNY-128` and then present the probability bound for 14-round encryption. In the latter portion of this section, we take `SKINNY-128` as an example to analyse the runtime for solving SAT problems constructed using different encoding methods in Section 5.

6.1 Specification of `SKINNY-128`

`SKINNY-128` is one variant of the `SKINNY` family of lightweight block ciphers [BJK+16]. The block size is $n = 128$ bits, and during the encryption phase, the internal state is viewed as a 4×4 square array of bytes. `SKINNY-128` is based on the TWEAKEY framework in [JNP14] and accepts tweakey input. The user can select which portion of the tweakey input serves as the key and/or tweak material. There are three variations of `SKINNY-128` based on tweakey size t, with tweakey sizes of 128, 256, and 384, respectively. The tweakey state is seen as a collection of t/n 4×4 square arrays of bytes.

Initialisation After receiving a plaintext $m = m_0\|m_1\|\cdots\|m_{15}$, where $m_i \in \mathbb{F}_2^8$, $0 \leqslant i \leqslant 15$, the internal state is created by setting the internal state IS to

$$IS = \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}.$$

$IS_{i,j}$ represents the internal state cell placed in the i-th row and j-th column. The 4×4 square array of cells can be viewed as a vector by concatenating the rows, and the cell of the internal state located at the l-th position in the vector is denoted with a single subscript IS_l, i.e., $IS_{i,j} = IS_{4i+j}$.

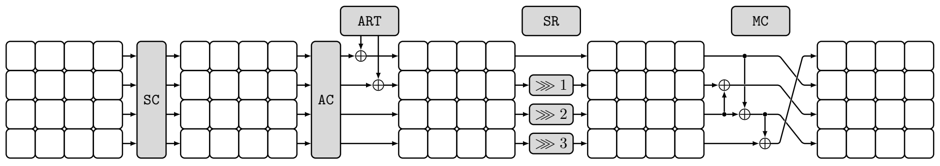


Figure 5: Round function of `SKINNY-128`.

As illustrated in Figure 5, a single encryption round consists of the following five operations: `SubCells` (`SC`), `AddConstants` (`AC`), `AddRoundTweakey` (`ART`), `ShiftRows` (`SR`), and `MixColumns` (`MC`). The number of rounds r varies according to the block and tweakey sizes. The minimal number of rounds required for `SKINNY-128` is 40 rounds.

Table 5: $\mathcal{S}_8$ of `SKINNY-128`. $\boldsymbol{x} \| \boldsymbol{y}$ represents the 8-bit input, where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^4$.

$\mathcal{S}_8(\boldsymbol{x}\|\boldsymbol{y})$		$\boldsymbol{y}$															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\boldsymbol{x}$	0	65	4c	6a	42	4b	63	43	6b	55	75	5a	7a	53	73	5b	7b
	1	35	8c	3a	81	89	33	80	3b	95	25	98	2a	90	23	99	2b
	2	e5	cc	e8	c1	c9	e0	c0	e9	d5	f5	d8	f8	d0	f0	d9	f9
	3	a5	1c	a8	12	1b	a0	13	a9	05	b5	0a	b8	03	b0	0b	b9
	4	32	88	3c	85	8d	34	84	3d	91	22	9c	2c	94	24	9d	2d
	5	62	4a	6c	45	4d	64	44	6d	52	72	5c	7c	54	74	5d	7d
	6	a1	1a	ac	15	1d	a4	14	ad	02	b1	0c	bc	04	b4	0d	bd
	7	e1	c8	ec	c5	cd	e4	c4	ed	d1	f1	dc	fc	d4	f4	dd	fd
	8	36	8e	38	82	8b	30	83	39	96	26	9a	28	93	20	9b	29
	9	66	4e	68	41	49	60	40	69	56	76	58	78	50	70	59	79
	a	a6	1e	aa	11	19	a3	10	ab	06	b6	08	ba	00	b3	09	bb
	b	e6	ce	ea	c2	cb	e3	c3	eb	d6	f6	da	fa	d3	f3	db	fb
	c	31	8a	3e	86	8f	37	87	3f	92	21	9e	2e	97	27	9f	2f
	d	61	48	6e	46	4f	67	47	6f	51	71	5e	7e	57	77	5f	7f
	e	a2	18	ae	16	1f	a7	17	af	01	b2	0e	be	07	b7	0f	bf
	f	e2	ca	ee	c6	cf	e7	c7	ef	d2	f2	de	fe	d7	f7	df	ff

`SubCells` (`SC`) The 8-bit S-box $\mathcal{S}_8$ (cf. Table 5) is applied to each byte of the state.

`AddConstants` (`AC`) A 6-bit LFSR, whose state is indicated by $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$, is used to generate round constants. The function for updating is

$$rc_4 \| rc_3 \| rc_2 \| rc_1 \| rc_0 \| rc_5 \oplus rc_4 \oplus 1 \leftarrow rc_5 \| rc_4 \| rc_3 \| rc_2 \| rc_1 \| rc_0.$$

The six bits of the LFSR are initialised to zero, and the LFSR is updated before being used in the current round. The bits of the LFSR are organised in a 4×4 array

$$\begin{bmatrix} c_0 & 0 & 0 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where $c_0 = 0\|0\|0\|0\|rc_3\|rc_2\|rc_1\|rc_0$, $c_1 = 0\|0\|0\|0\|0\|0\|rc_5\|rc_4$, and $c_2 =$ `0x02`. The round constants are mixed with the internal state via bitwise XOR operations, while array placement is respected.

`AddRoundTweakey` (`ART`) The first and second rows of each tweakey array are extracted and XORed with the internal state. Since the value of tweakey has no effect on the search for differential characteristics, we do not describe its construction.

`ShiftRows` (`SR`) The second, third, and fourth rows are rotated to the right by one, two, and three bytes, respectively.

`MixColumns` (`MC`) Multiplying each column of the internal state by the following binary matrix

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

6.2 Tight Probability Bound for 14-Round of SKINNY-128

Due to the large size of the state, the designers of SKINNY-128 gave only lower bounds for the number of differential active S-boxes. Given that the number of active S-boxes for 14 rounds is 61, the probability for 14-round differential characteristics has an upper bound of 2^{-122}. Abdelkhalek *et al.* attempted to propose tight upper bounds for the probability of SKINNY-128 utilising a MILP model for large S-boxes. The task was completed up to 13 rounds, and the search on 13 rounds took 16 days, according to [AST+17]. For 14-round of SKINNY-128, they merely demonstrated that no differential characteristic had a probability greater than 2^{-128}.

Table 6: Upper bound on the differential probability p of SKINNY-128.

Round	1	2	3	4	5	6	7
$-\log_2(p)$	2	4	10	16	24	32	52
Round	8	9	10	11	12	13	14
$-\log_2(p)$	72	86	96	104	112	123	131

Using the fast SAT models for big S-boxes, we find in this paper that the upper bound on the probability for 14 rounds of SKINNY-128 is 2^{-131}, thereby finishing the remaining task of Abdelkhalek *et al.* The outcome of searching for the upper bound on the differential probability for up to 14 rounds of SKINNY-128 using the SAT approach is shown in Table 6. The probability bounds for 10-round and 13-round differential characteristics conform to the result in [AST+17]. Figure 6 illustrates a 14-round differential characteristic of SKINNY-128 with probability 2^{-131}.

6.3 Runtime with Different Encoding Methods for S-boxes

The time required to solve SAT problems generated with different S-box encoding techniques varies. The numerous sets of clauses in Section 5 produced by ESPRESSO under various settings and encoding techniques are utilised to determine the upper bound of differential probability for SKINNY-128 from 1 to 14 rounds. The solver Cryptominisat is used to solve all of the SAT problems in this study. In the test, we use the models returned by ESPRESSO without modifying the order of the clauses in the model. We should remind the reader that the order of clauses and/or variables may also influence the execution time of the SAT solver. In addition, the SAT solver may also have a considerable variance in the execution time, and we believe that a comparison with other SAT solvers would be an interesting research project.

As seen in Figure 2, for the 26-bit encoding of $\mathcal{S}_8$, the partition approach can minimise the time required for ESPRESSO to create reduced clauses, but the time required for the SAT solver to solve the SAT problem rises. The good news is that for some values of s, the overall runtime, which consists of the runtimes for ESPRESSO and SAT solver, can be decreased. For example, when s is set to 8, the overall runtime is 10754.7 seconds, however it is 11123.1 seconds when using the clauses provided in Section 5.1.

Figure 3 depicts the test outcomes relevant to the 25-bit encoding of $\mathcal{S}_8$. Similar to the scenario presented in Figure 2, the partition approach shortens the execution time for ESPRESSO but cannot reduce the execution time for the SAT solver. In addition, a comparison of Figure 2 and Figure 3 reveals that lowering the number of variables may not reduce the execution time of SAT solver. If the partition technique is not employed, for instance, it takes 7601.7 seconds to solve a SAT problem with S-boxes containing 820 clauses and 26 variables; whereas it takes 7952.3 seconds to solve a SAT problem with S-boxes containing 818 clauses and 25 variables. Recall that Sasaki and Todo [ST17]

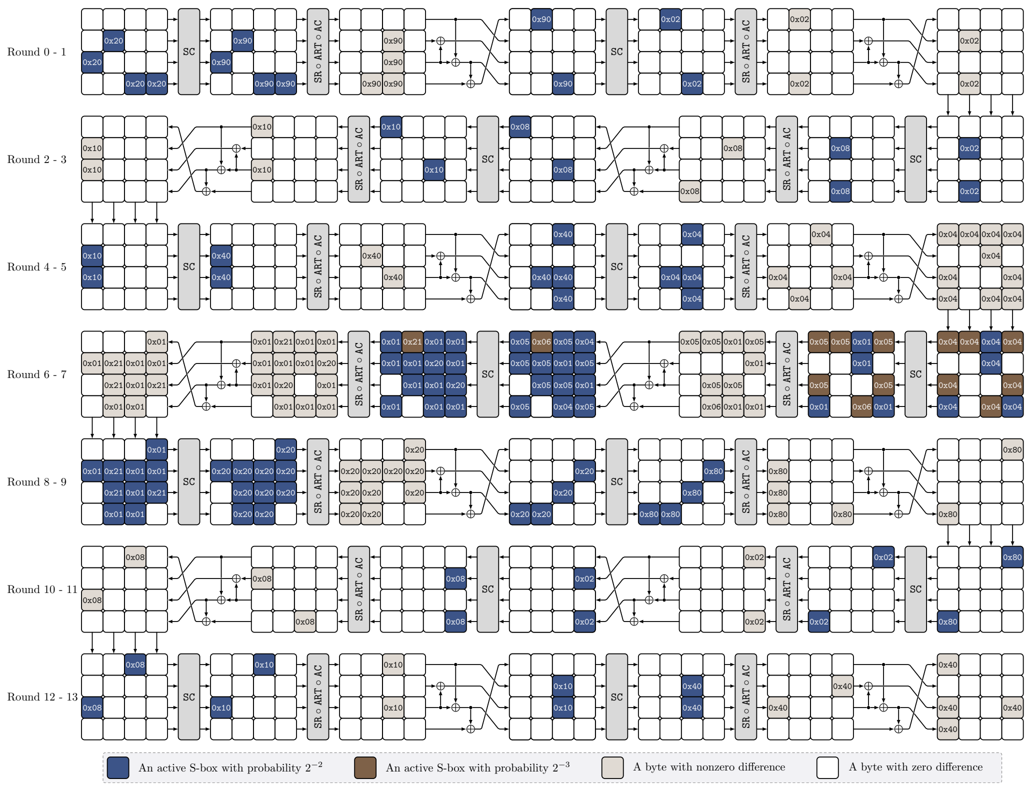

Round 0 - 1
Round 2 - 3
Round 4 - 5
Round 6 - 7
Round 8 - 9
Round 10 - 11
Round 12 - 13
SC
SR ∘ ART ∘ AC
An active S-box with probability 2^{-2}
An active S-box with probability 2^{-3}
A byte with nonzero difference
A byte with zero difference

Figure 6: 14-round differential characteristic of `SKINNY-128` with probability 2^{-131}.

shown that, for 4-bit S-boxes, lowering the number of inequalities in MILP models does not necessarily reduce the runtime of the MILP optimiser. Figure 2 and Figure 3 illustrate that, for 8-bit S-boxes, reducing the number of clauses in SAT problems does not always decrease the execution time of SAT solver. Moreover, a comparison of Figure 2 and Figure 3 demonstrates that lowering the number of variables in SAT problems may not reduce the execution time of SAT solver.

The sets of clauses (see Table 4) created by the two-step encoding approach are also incorporated into SAT problems so that the runtime for searching for 1-round to 14-round upper bound on differential probability for `SKINNY-128` may be evaluated. The test result is given in Figure 7. Figure 7 demonstrates that for two sets of clauses with the same amount of clauses, the runtime for the SAT solver to solve their respective SAT problems may change. When comparing the results of Figure 7 to those of Figure 2 and Figure 3, it is evident that the time required for ESPRESSO to locate simplified clauses has been drastically decreased. If the ESPRESSO option `-eonset` is fixed, the runtime for the simplification phase of the two-step technique is 99.0 seconds, whereas the runtimes for the 26-bit and 25-bit encodings are 3521.4 and 1883.5 seconds, respectively. The bad news is that the total runtime increases; the minimum total runtime for the two-step technique is 12659.4 seconds, whereas the minimum total runtimes for the 26-bit and 25-bit encodings are 11123.1 seconds and 9835.8 seconds, respectively.

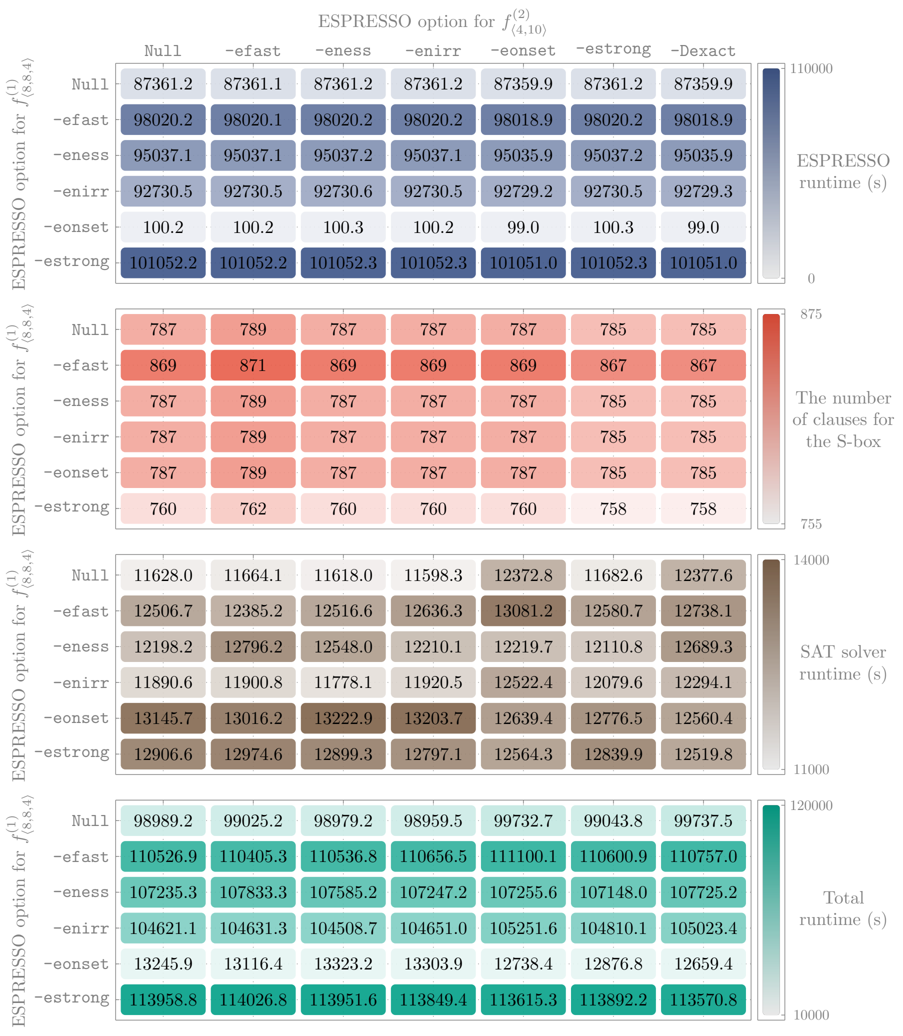
Figure 4 depicts the cumulative impact of the two-step technique and partition method on the execution time of SAT solver. As shown in Figure 4(a), incorporating the partition into the two-step technique may have a beneficial impact on the execution time of SAT solver when ESPRESSO is implemented with the `-eonset` option. For instance, when s is set to 1, the runtime of the SAT solver is 11087.2 seconds, which is shorter than the

12639.4 seconds required by the technique without the partition method. In addition, the overall execution time for $s = 1$ is 11126.3 seconds, which is faster than the total execution time when the partition technique is not used. Comparing the findings in Figure 4(a) and Figure 4(b) demonstrates the previously indicated fact, namely that lowering the number of clauses in SAT problems does not necessarily reduce the execution time of the SAT solver.

After analysing all the test results, we conclude that the encoding strategy described in Section 5.1, which relies on the `-eonset` option given by ESPRESSO, is a good solution for balancing the runtime for ESPRESSO and the execution time for the SAT solver. In the remaining applications for PIPO and AES-based constructions, we employ the procedure described in Section 5.1 to create S-box clauses.

7 Related-Key Differential Properties of PIPO

In this section, the modelling approach for large S-boxes is applied to PIPO, and the related-key differential properties of the two variants of PIPO are studied.



ESPRESSO option for $f^{(2)}_{\langle 4,10\rangle}$ (columns); ESPRESSO option for $f^{(1)}_{\langle 8,8,4\rangle}$ (rows)

ESPRESSO runtime (s) (0 – 110000)

	Null	-efast	-eness	-enirr	-eonset	-estrong	-Dexact
Null	87361.2	87361.1	87361.2	87361.2	87359.9	87361.2	87359.9
-efast	98020.2	98020.1	98020.2	98020.2	98018.9	98020.2	98018.9
-eness	95037.1	95037.1	95037.2	95037.1	95035.9	95037.2	95035.9
-enirr	92730.5	92730.5	92730.6	92730.5	92729.2	92730.5	92729.3
-eonset	100.2	100.2	100.3	100.2	99.0	100.3	99.0
-estrong	101052.2	101052.2	101052.3	101052.3	101051.0	101052.3	101051.0

The number of clauses for the S-box (755 – 875)

	Null	-efast	-eness	-enirr	-eonset	-estrong	-Dexact
Null	787	789	787	787	787	785	785
-efast	869	871	869	869	869	867	867
-eness	787	789	787	787	787	785	785
-enirr	787	789	787	787	787	785	785
-eonset	787	789	787	787	787	785	785
-estrong	760	762	760	760	760	758	758

SAT solver runtime (s) (11000 – 14000)

	Null	-efast	-eness	-enirr	-eonset	-estrong	-Dexact
Null	11628.0	11664.1	11618.0	11598.3	12372.8	11682.6	12377.6
-efast	12506.7	12385.2	12516.6	12636.3	13081.2	12580.7	12738.1
-eness	12198.2	12796.2	12548.0	12210.1	12219.7	12110.8	12689.3
-enirr	11890.6	11900.8	11778.1	11920.5	12522.4	12079.6	12294.1
-eonset	13145.7	13016.2	13222.9	13203.7	12639.4	12776.5	12560.4
-estrong	12906.6	12974.6	12899.3	12797.1	12564.3	12839.9	12519.8

Total runtime (s) (10000 – 120000)

	Null	-efast	-eness	-enirr	-eonset	-estrong	-Dexact
Null	98989.2	99025.2	98979.2	98959.5	99732.7	99043.8	99737.5
-efast	110526.9	110405.3	110536.8	110656.5	111100.1	110600.9	110757.0
-eness	107235.3	107833.3	107585.2	107247.2	107255.6	107148.0	107725.2
-enirr	104621.1	104631.3	104508.7	104651.0	105251.6	104810.1	105023.4
-eonset	13245.9	13116.4	13323.2	13303.9	12738.4	12876.8	12659.4
-estrong	113958.8	114026.8	113951.6	113849.4	113615.3	113892.2	113570.8

Figure 7: Test results for two-step encoding method regarding $\mathcal{S}_8$.

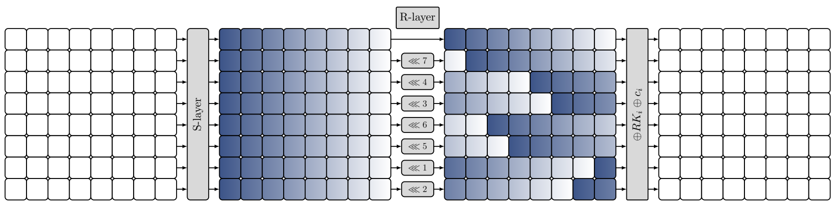


Figure 8: Round function of PIPO.

7.1 Description of PIPO

PIPO is a lightweight block cipher proposed at ICISC 2020 by Kim *et al.* [KJK+20]. It is a 64-bit block cipher with two instances that accept 128-bit and 256-bit keys, and we distinguish between them using the notations PIPO-128 and PIPO-256. The number of rounds r during encryption is dependent on the size of the key: $r = 13$ for PIPO-128 and $r = 17$ otherwise.

The internal state is regarded as an 8×8 square array of bits during the encryption and decryption processes. We refer to X as the internal state and $X[i]$ as the i-th row of X for $0 \leqslant i \leqslant 7$. Plaintext $m = m_{63} \| m_{62} \| \cdots \| m_0$ is supplied to the cipher. The internal state is initialised by setting X in the following manner

$$X = \begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \\ X[4] \\ X[5] \\ X[6] \\ X[7] \end{bmatrix} = \begin{bmatrix} m_7 & m_6 & m_5 & m_4 & m_3 & m_2 & m_1 & m_0 \\ m_{15} & m_{14} & m_{13} & m_{12} & m_{11} & m_{10} & m_9 & m_8 \\ m_{23} & m_{22} & m_{21} & m_{20} & m_{19} & m_{18} & m_{17} & m_{16} \\ m_{31} & m_{30} & m_{29} & m_{28} & m_{27} & m_{26} & m_{25} & m_{24} \\ m_{39} & m_{38} & m_{37} & m_{36} & m_{35} & m_{34} & m_{33} & m_{32} \\ m_{47} & m_{46} & m_{45} & m_{44} & m_{43} & m_{42} & m_{41} & m_{40} \\ m_{55} & m_{54} & m_{53} & m_{52} & m_{51} & m_{50} & m_{49} & m_{48} \\ m_{63} & m_{62} & m_{61} & m_{60} & m_{59} & m_{58} & m_{57} & m_{56} \end{bmatrix}.$$

The key schedule is quite simple.

▷ The 128-bit master key K is divided into two 64-bit subkeys K_0 and K_1 as $K = K_1 \| K_0$ for PIPO-128. The whitening and round keys are defined as $RK_i = K_{i \bmod 2}$, where $i = 0, 1, \ldots, 13$.

▷ The 256-bit master key K in PIPO-256 is divided into four 64-bit subkeys K_0, K_1, K_2, and K_3 as $K = K_3 \| K_2 \| K_1 \| K_0$. The setting for the whitening and round keys is $RK_i = K_{i \bmod 4}$, $i = 0, 1, \ldots, 17$.

Before running the round function in both variants, the whitening key RK_0 is XORed with the internal state.

One encryption round in PIPO consists of the following three operations: S-layer, R-layer, and round key and constant XOR. See Figure 8 for a demonstration of the round function.

S-layer An 8-bit S-box denoted as S_8 is applied to each column of the 8×8 square array of bits, with the uppermost bit being the least significant. The definition of S_8 is available in [KJK+20].

R-layer The rows of the array of cipher state bits are rotated in this layer, as depicted in Figure 8.

Round key and constant XOR Internal state is XORed with the i-th round key RK_i and the i-th constant $c_i = i$, where $1 \leqslant i \leqslant r$.

7.2 Related-Key Differential Properties of PIPO-128

We analyse the related-key differential properties of PIPO-128 using the S-box modelling method in Section 5.1 and list the test results in Table 7. In [YK21], Yadav and Kumar showed a 2-round iterative related-key differential characteristic with probability 2^{-4} and construct a full-round characteristic with probability 2^{-24} for PIPO-128. This result is consistent with their findings. Beyond this, we discover that PIPO-128 has 1792 full-round characteristics with a probability of 2^{-24}. Figure 9 provides a visual depiction of these characteristics. There are 224 possible propagations for the DDT of S_8, with a probability of 2^{-4}. In addition, the differential propagation $\alpha_7\|\alpha_6\|\alpha_5\|\alpha_4\|\alpha_3\|\alpha_2\|\alpha_1\|\alpha_0 \to \beta_7\|\beta_6\|\beta_5\|\beta_4\|\beta_3\|\beta_2\|\beta_1\|\beta_0$ depicted in Figure 9 can also be placed in the remaining seven columns. These explain the origin of the 1792($= 224 \times 8$) differential characteristics.

Table 7: Related-key differential properties of PIPO-128.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13
Active S-box	0	0	1	2	2	3	3	4	4	5	5	6	6
$-\log_2(p)$	0	0	4	8	8	12	12	16	16	20	20	24	24

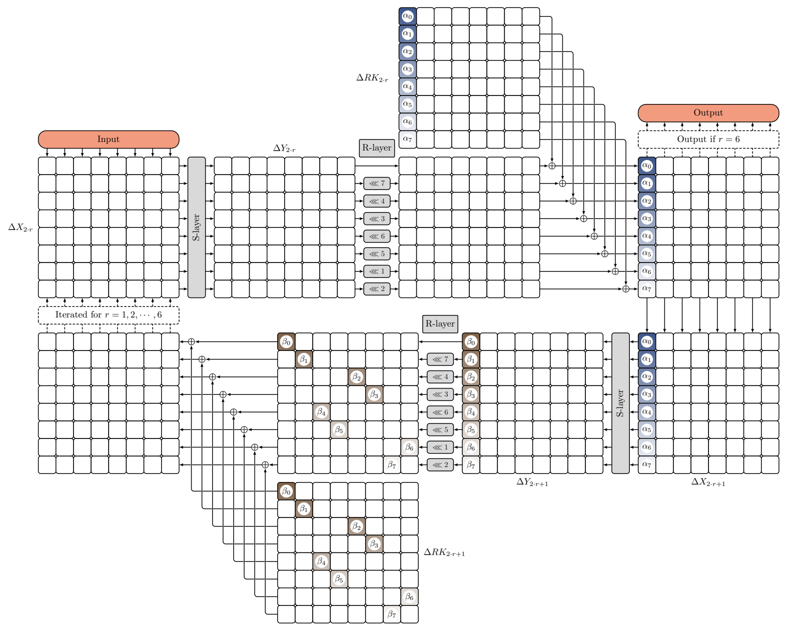

Input
Output
Output if $r = 6$
$\Delta RK_{2\cdot r}$
$\Delta X_{2\cdot r}$
$\Delta Y_{2\cdot r}$
S-layer
R-layer
Iterated for $r = 1, 2, \cdots, 6$
$\Delta Y_{2\cdot r+1}$
$\Delta X_{2\cdot r+1}$
$\Delta RK_{2\cdot r+1}$
α_0 α_1 α_2 α_3 α_4 α_5 α_6 α_7
β_0 β_1 β_2 β_3 β_4 β_5 β_6 β_7
$\lll 7$ $\lll 4$ $\lll 3$ $\lll 6$ $\lll 5$ $\lll 1$ $\lll 2$

Figure 9: Full-round related-key differential characteristic with probability 2^{-24} for PIPO-128. $\alpha_7\|\alpha_6\|\alpha_5\|\alpha_4\|\alpha_3\|\alpha_2\|\alpha_1\|\alpha_0 \to \beta_7\|\beta_6\|\beta_5\|\beta_4\|\beta_3\|\beta_2\|\beta_1\|\beta_0$ represents a possible differential propagation for the S-box with probability 2^{-4}.

7.3 Related-Key Differential Properties of PIPO-256

The new encoding strategy for large S-boxes in Section 5.1 is employed to examine the related-key differential properties of PIPO-256. The results are summarised in Table 8. It can be notice that there exists a full-round differential characteristic with probability 2^{-16}, and a further investigation shows that there are 5376 full-round differential characteristic with probability 2^{-16}. An illustration for the full-round characteristic can be found in Figure 10. Similar to PIPO-128, the differential propagation $\alpha_7\|\alpha_6\|\alpha_5\|\alpha_4\|\alpha_3\|\alpha_2\|\alpha_1\|\alpha_0 \to$

$\beta_7\|\beta_6\|\beta_5\|\beta_4\|\beta_3\|\beta_2\|\beta_1\|\beta_0$ with probability 2^{-4} offers 224 options, and the propagation of Figure 10 may be inserted in the remaining seven columns. In addition, the input location of the 17-round distinguisher can be moved to the $\Delta X_{4\cdot r+1}$ or $\Delta X_{4\cdot r+3}$ positions of Figure 10. Taking all of these facts into account, it is possible to explain the origin of the $5376 (= 224 \times 8 \times 3)$ differential characteristics.

Table 8: Related-key differential properties of PIPO-256.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Active S-box	0	0	0	0	1	1	1	2	2	2	2	3	3	3	3	4	4
$-\log_2(p)$	0	0	0	0	4	4	4	8	8	8	8	12	12	12	12	16	16

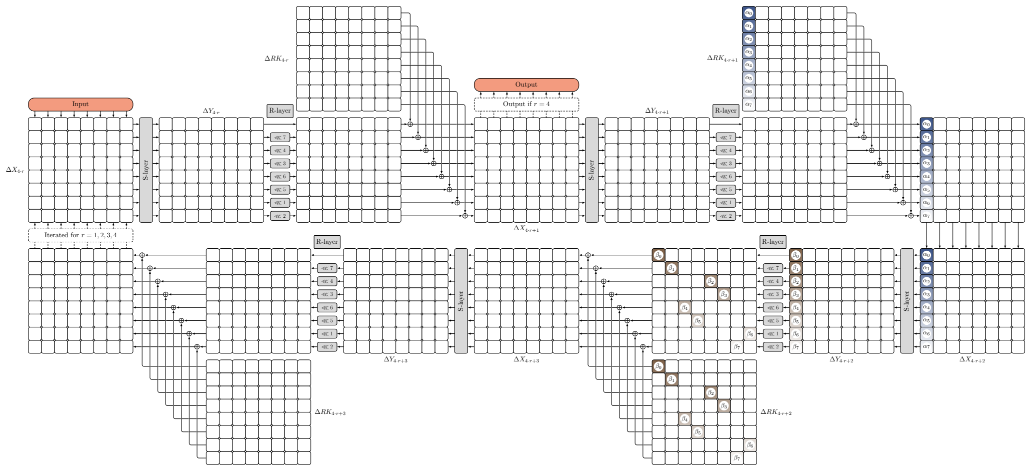


Figure 10: Full-round related-key differential characteristic with probability 2^{-16} for PIPO-256. $\alpha_7\|\alpha_6\|\alpha_5\|\alpha_4\|\alpha_3\|\alpha_2\|\alpha_1\|\alpha_0 \rightarrow \beta_7\|\beta_6\|\beta_5\|\beta_4\|\beta_3\|\beta_2\|\beta_1\|\beta_0$ represents a possible differential propagation for the S-box with probability 2^{-4}.

8 Application to AES-Based Constructions

Jean and Nikolić [JN16] suggested a number of AES-based constructions that can be utilised as building blocks for Message Authentication Codes (MACs) and Authenticated Encryption (AE) schemes. They intended to explore the efficiency boundaries of constructions that may be realised with AES-NI instruction aesenc, which executes one round of AES. The internal states of the designs are made up of many 128-bit words, known as *blocks*, and the *step functions* are based only on aesenc and bitwise XOR operations. The state size, the amount of aesenc calls each step, and the selection of state words to which aesenc is applied assure the high efficiency of the designs.

In addition to the efficiency, Jean and Nikolić were particularly concerned with the security of the designs. Since the most common attacks against MACs and AE are internal collisions based on high probability differential characteristics that begin and end with zero state differences, the security of these constructions is determined by counting the number of active S-boxes required to produce an internal collision. The minimum number of active S-boxes should be 22, given that the key size is 128 bits and the maximum differential probability of the S-box in AES is 2^{-6}. Jean and Nikolić used a MILP-based search to determine the number of active S-boxes and provided seven secure constructions C1 - C7 (cf. Figure 11(a) - Figure 11(g)) with excellent state size and efficiency trade-offs. The lower bounds for the number of active S-boxes for C1 - C7 using the MILP model are listed in Table 9, however the number of step functions necessary to attain the lower bound is not specified. As noted by the authors, the automatic search strategy they employ targets truncated differentials. There may not be a true differential characteristic that corresponds to the search outcomes. In other words, it is possible that the security of these structures was underestimated.

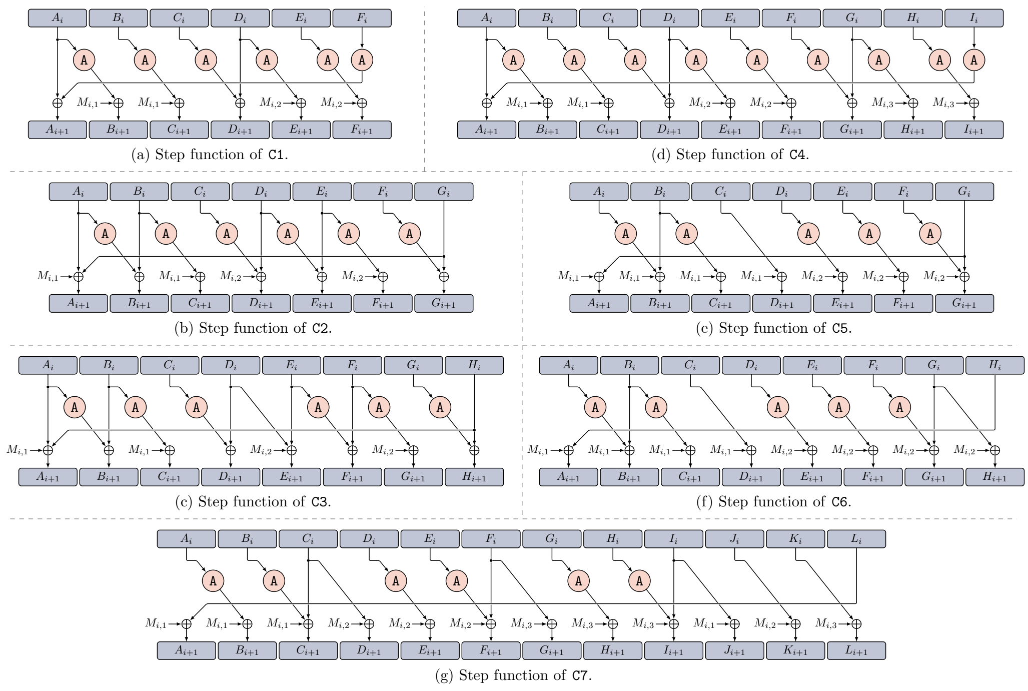


Figure 11: Step functions of C1 - C7. A_i, B_i, …, L_i are input blocks for the i-th step function. A_{i+1}, B_{i+1}, …, L_{i+1} are output blocks for the i-th step function. A represents one-round of AES. $M_{i,1}$, $M_{i,2}$, and $M_{i,3}$ are message blocks incorporated at the i-th step.

Abdelkhalek *et al.* [AST+17] were aware of this issue and analysed the security of two arbitrarily selected constructions, i.e., C5 and C1, using the MILP model for large S-boxes (cf. Table 9). They demonstrated that it is impossible for C5 to have any 4-step truncated differential characteristics with no more than 23 active S-boxes. Therefore, they concluded that the minimum number of active S-boxes for C5 is 24, an increase from the initial estimate of 22 by the designers. For C1, they confirmed that the minimal number of active S-boxes required to cause a collision is 22, while the best differential characteristic has a probability of 2^{-134}, as opposed to 2^{-132}.

In this study, the encoding approach for large S-boxes proposed in Section 5.1 is applied to the constructions C1 - C7. The SAT method is used to determine the minimum number of active S-boxes necessary to cause an internal collision. All seven constructions, ranging from two to eight steps, are analysed. The test outcomes are shown in Table 9.

8.1 Results on C1

For the construction C1, we first verify that no differential characteristic may lead to an internal collision if the number of steps n_s is fixed at 2. When $n_s \geqslant 3$, the lower bound on the number of active S-boxes reaches 22, in agreement with the analyses in [JN16] and

Table 9: Lower bound on the number of active S-boxes for C1 - C7.

C1		C2		C3		C4		C5		C6		C7		Ref.
n_s	#S	n_s	#S	n_s	#S	n_s	#S	n_s	#S	n_s	#S	n_s	#S	
-	22	-	25	-	34	-	25	-	22	-	23	-	25	[JN16]
3-7	22	-	-	-	-	-	-	4-7	24	-	-	-	-	[AST+17]
2	✗	2	✗	2	✗	2	✗	2	✗	2	✗	2	48	Section 8
3	22	3	50	3	47	3	33	3	40	3	48	3	48	
4	22	4	25	4	47	4	25	4	25	4	> 41	4	> 37	
5	22	5	25	5	36	5	25	5	25	5	23	5	28	
6	22	6	25	6	36	6	25	6	25	6	23	6	28	
7	22	7	25	7	36	7	25	7	25	7	23	7	> 24	
8	22	8	25	8	36	8	25	8	25	8	23	8	> 24	

n_s: The number of step functions. #S: The number of active S-boxes.

-: No information is provided.

✗: There is no differential characteristic with the specified number of step functions.

[AST+17]. According to our analysis of all 3-step differential characteristics with 22 active S-boxes, there are a total of four differential patterns (cf. Supplementary Material), which match to the four patterns described in [AST+17].

In addition to examining the lower bound on the number of active S-boxes, Abdelkhalek *et al.* [AST+17] evaluate the upper bound on the differential probability for 3-step differential characteristics. The probability upper bound is demonstrated by analysing each of the four differential patterns and determining that not all of the active S-boxes in the characteristic attain the optimal probability, say 2^{-6}. We also study the upper bound on the differential probability for C1 using the automatic technique. The SAT modelling approach described in Section 5.1 for large S-boxes oriented to differential probability is implemented. The test results validate the conclusion presented by Abdelkhalek *et al.*, namely that the maximum differential probability for 3-step characteristics that facilitate an internal collision is 2^{-134}. Figure 12 demonstrates a newly identified 3-step differential characteristic with a probability of 2^{-134}.

8.2 Results on C2

For C2, we first determine that there is no differential characteristic that would support an internal collision if the number of steps n_s is set at 2. When n_s is set to 3, the minimum number of active S-boxes is 50. When $n_s \geqslant 4$, the minimum number of active S-boxes for differential characteristics guaranteeing an internal collision reaches 25, as specified in [JN16]. The differential patterns of all 4-step differential characteristics of 25 active S-boxes are analysed, resulting in a total of 16 differential patterns. The 16 differential patterns are supplied in the Supplementary Material.

8.3 Results on C3

For C3, we begin by ensuring that, while the number of steps n_s is set at 2, there are no differential characteristics that might cause an internal collision. The minimum number of active S-boxes for differential characteristics is 47 when n_s is set to 3 or 4. For $n_s \geqslant 5$, we discover that the minimum number of active S-boxes for differential characteristics is 36, and this bound holds for up to eight steps. Given the 34 active S-boxes established by [JN16], our new result increases the lower bound on the number of active S-boxes for C3. There are 32 differential patterns after analysing the differential patterns of all 5-step differential characteristics with 36 active S-boxes. The 32 differential patterns are presented in the Supplementary Material.

8.4 Results on C4

For the C4 construction, we first discover that if the number of steps n_s is set at 2, no differential characteristic can sustain an internal collision. When n_s is increased to 3, the number of active S-boxes for differential characteristics reaches a minimum of 33. If n_s is set to 4, the minimum number of active S-boxes is reduced to 25; this bound is consistent with the one provided by [JN16]. The differential patterns of all 4-step characteristics with 25 active S-boxes are then analysed, and a total of 12 differential patterns are discovered. The 12 differential patterns can be seen in the Supplementary Material.

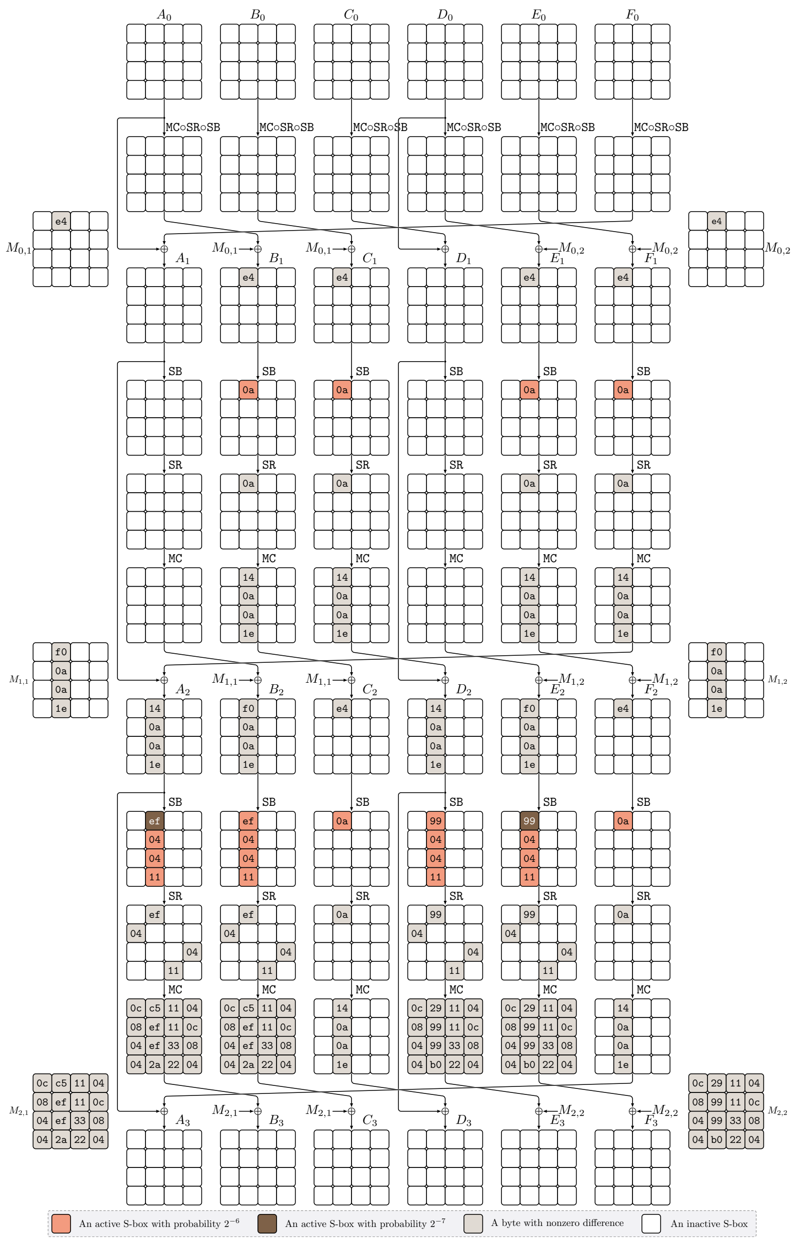


Figure 12: 3-step differential characteristic of probability 2^{-134} for C1.

8.5 Results on `C5`

For the construction `C5`, when the number of step functions n_s is fixed at 2, there is no differential characteristic that may cause an internal collision, and an internal collision is only feasible if $n_s \geqslant 3$. When n_s is set to 3, there must be at least 40 active S-boxes for differential characteristics to cause a collision. The minimum number of active S-boxes is reduced to 25 when $n_s = 4$. Table 9 demonstrates that our results raise the lower bound on the number of active S-boxes for `C5` when compared to those of [AST+17] and [JN16]. We examine the 4-step differential characteristics of 25 active S-boxes and discover that all of these characteristics can be classified into 12 differential patterns. The 12 differential patterns are available in the Supplementary Material.

8.6 Results on `C6`

First, we verify that 2-step differential characteristics cannot lead to an internal collision for `C6`. When the number of steps n_s is set to 3, the minimum number of active S-boxes is determined to be 48. Due to the extremely lengthy runtime of the SAT solver, when n_s is fixed to 4, we only validate that all 4-step differential characteristics have more than 41 active S-boxes, but we cannot guarantee an exact lower bound for the number of active S-boxes. For $n_s \geqslant 5$, the minimum number of active S-boxes is confirmed to be 23; this value corresponds to the one specified in [JN16].

8.7 Results on `C7`

For the `C7` construction, we find that 2-step differential characteristics can lead to an internal collision, and that the minimal number of active S-boxes is 48. When n_s is set to 3, the minimum number of active S-boxes remains unchanged. For 4-step differential characteristics, we only validate that the minimal number of active S-boxes is more than 37, and the concrete bound cannot be determined until the SAT solver is run for an extremely long period of time. The minimum number of active S-boxes is 28 when $n_s = 5$ or 6. Due to the restricted problem-solving capacity of the SAT solver, the lower bound on the number of active S-boxes for $n_s \geqslant 7$ is uncertain. We only validate that the minimum number of active S-boxes exceeds 24, tying the result with [JN16].

9 Conclusion

This study begins with a summary of the MILP and SAT modelling developments for large S-boxes. Then, we provide three techniques for quickly constructing SAT models for large S-boxes oriented to differential probabilities and linear correlations. The newly suggested encoding techniques are initially used to the analysis of `SKINNY-128`. Comparing the runtime of alternative encoding techniques for S-boxes reveals that the first encoding approach achieves a good compromise between the runtime of ESPRESSO and the execution time of the SAT solver. On the other hand, we determine that the upper bound on the differential probability for 14 rounds of `SKINNY-128` is 2^{-131}, completing the remaining work of Abdelkhalek *et al.* The first approach of encoding large S-boxes is also implemented on PIPO and the seven `AES`-based constructions `C1` - `C7`. For two constructions `C3` and `C5`, the current lower bound on the number of active S-boxes is lifted, resulting in a more precise security analysis for these two structures. Additionally, all differential patterns for `C1` - `C5` that achieve a minimum number of active S-boxes are reported.

Acknowledgements.

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. The research leading to these results has received funding from the National Natural Science Foundation of China (Grant No. 62272273, Grant No. 62002201, Grant No. 62032014), the National Key Research and Development Program of China (Grant No. 2018YFA0704702), and the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025).

References

[AK18] Ralph Ankele and Stefan Kölbl. Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, pages 163–190, 2018.

[AST+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) S-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2017(4):99–129, 2017.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[BC20] Christina Boura and Daniel Coggia. Efficient MILP modelings for Sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020.

[BDG+21] Zhenzhen Bao, Xiaoyang Dong, Jian Guo, Zheng Li, Danping Shi, Siwei Sun, and Xiaoyun Wang. Automatic search of meet-in-the-middle preimage attacks on AES-like hashing. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 771–804. Springer, 2021.

[BHH+82] Robert K Brayton, GD Hachtel, LA Hemachandra, AR Newton, and ALM Sangiovanni-Vincentelli. A comparison of logic minimization strategies using espresso: An apl program package for partitioned logic minimization. In *Proceedings of the International Symposium on Circuits and Systems*, pages 42–48, 1982.

[BHMS84] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2 of *The Kluwer International Series in Engineering and Computer Science.* Springer, 1984.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The

SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 450–466, 2007.

[Bro81] Douglas W. Brown. A state-machine synthesizer - SMS. In Robert J. Smith, editor, *Proceedings of the 18th Design Automation Conference, DAC '81, Nashville, Tennessee, USA, June 29 - July 1, 1981*, pages 301–305. ACM/IEEE, 1981.

[Bro83] Arne Brondsted. *An introduction to convex polytopes*, volume 90. Springer Science & Business Media, 1983.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[FWG$^+$16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. MILP-based automatic search algorithms for differential and linear trails for Speck. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 268–288, 2016.

[GD07] Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 519–531. Springer, 2007.

[GKPS67] Branko Grünbaum, Victor Klee, Micha A Perles, and Geoffrey Colin Shephard. *Convex polytopes*, volume 16. Springer, 1967.

[GO04] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition.* Chapman and Hall/CRC, 2004.

[Gur22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[HCO74] Se June Hong, Robert G. Cain, and Daniel L. Ostapko. MINI: A heuristic approach for logic minimization. *IBM J. Res. Dev.*, 18(5):443–458, 1974.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2014.

[KJK$^+$20] Hangi Kim, Yongjin Jeon, Giyoon Kim, Jongsung Kim, Bo-Yeon Sim, Dong-Guk Han, Hwajeong Seo, Seonggyeom Kim, Seokhie Hong, Jaechul Sung, and Deukjo Hong. PIPO: A lightweight block cipher with efficient higher-order masking software implementations. In Deukjo Hong, editor, *Information Security and Cryptology - ICISC 2020 - 23rd International Conference, Seoul, South Korea, December 2-4, 2020, Proceedings*, volume 12593 of *Lecture Notes in Computer Science*, pages 99–122. Springer, 2020.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 161–185, 2015.

[LLL$^+$19] Yu Liu, Huicong Liang, Muzhou Li, Luning Huang, Kai Hu, Chenhe Yang, and Meiqin Wang. STP models of optimal differential and linear trail for S-box based ciphers. *IACR Cryptol. ePrint Arch.*, 2019:25, 2019.

[LS22] Ting Li and Yao Sun. Superball: A new approach for MILP modelings of Boolean functions. *IACR Trans. Symmetric Cryptol.*, 2022(3):341–367, 2022.

[LWR16] Yunwen Liu, Qingju Wang, and Vincent Rijmen. Automatic search of linear trails in ARX with applications to SPECK and Chaskey. In *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 485–499, 2016.

[McC56] Edward J McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Technical report, Cryptology ePrint Archive, Report 2013/328, 2013.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, pages 57–76, 2011.

[Pet56] Stanley R Petrick. A direct determination of the irredundant forms of a Boolean function from the set of prime implicants. *Air Force Cambridge Res. Center Tech. Report*, pages 56–110, 1956.

[Qui52] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952.

[Qui55] Willard V Quine. A way to simplify truth functions. *The American mathematical monthly*, 62(9):627–631, 1955.

[Ric11] S Rickmann. Logic friday (version 1.1.3) [computer software], 2011.

[SGL$^+$17] Siwei Sun, David Gérault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. Analysis of AES, SKINNY, and others with constraint programming. *IACR Trans. Symmetric Cryptol.*, 2017(1):281–306, 2017.

[SHW$^+$14a] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, and Ling Song. Automatic enumeration of (related-key) differential and linear characteristics with predefined properties and its applications. *IACR Cryptol. ePrint Arch.*, page 747, 2014.

[SHW+14b] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 158–178, 2014.

[SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, pages 379–394, 2016.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[SSD+18] Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the Demirci-Selçuk meet-in-the-middle attack with constraints. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2018.

[ST17] Yu Sasaki and Yosuke Todo. New algorithm for modeling S-box in MILP based differential and division trail search. In Pooya Farshim and Emil Simion, editors, *Innovative Security Solutions for Information Technology and Communications - 10th International Conference, SecITC 2017, Bucharest, Romania, June 8-9, 2017, Revised Selected Papers*, volume 10543 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2017.

[SWW17] Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for ARX ciphers and word-based division property. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 128–157, 2017.

[SWW18] Ling Sun, Wei Wang, and Meiqin Wang. More accurate differential properties of LED64 and Midori64. *IACR Trans. Symmetric Cryptol.*, 2018(3):93–123, 2018.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symmetric Cryptol.*, 2021(1):269–315, 2021.

[The22] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 1.4.2)*, 2022. `https://www.sagemath.org`.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 250–279, 2017.

[Udo21] Aleksei Udovenko. MILP modeling of Boolean functions by minimum number of inequalities. *IACR Cryptol. ePrint Arch.*, page 1099, 2021.

[WHG$^+$19] Senpeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. MILP-aided method of searching division property using three subsets and applications. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 398–427. Springer, 2019.

[WW11] Shengbao Wu and Mingsheng Wang. Security evaluation against differential cryptanalysis for block cipher structures. *IACR Cryptol. ePrint Arch.*, page 551, 2011.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 648–678, 2016.

[YK21] Tarun Yadav and Manoj Kumar. MILES: modeling large S-box in MILP based differential characteristic search. *IACR Cryptol. ePrint Arch.*, page 1388, 2021.

[Zie07] Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2007.