IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 3, pp. 570–596. DOI:10.46586/tches.v2023.i3.570-596

RAFA: Redundancies-assisted Algebraic Fault Analysis and its implementation on SPN block ciphers

Zehong (Zephyr) Qiu[1], Fan Zhang[1,2,3†], Tianxiang Feng[1] and Xue Gong[1]

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[2] Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Hangzhou, China
[3] Zhengzhou Xinda Institute of Advanced Technology, Zhengzhou, China
zephyr.qiu.cn@gmail.com;fanzhang@zju.edu.cn

Abstract. *Algebraic Fault Analysis* (AFA) is a cryptanalysis for block ciphers proposed by Courtois *et al.*, which incorporates algebraic cryptanalysis to overcome the complexity of manual analysis within the context of *Differential Fault Analysis* (DFA). The effectiveness of AFA on lightweight block ciphers has been demonstrated. However, the complexity of the algebraic systems prevents it from attacking heavyweight block ciphers efficiently. In this paper, we propose a novel cryptanalysis called *Redundancies-assisted Algebraic Fault Analysis* (RAFA) to facilitate the solution of algebraic systems in the setting of heavyweight block ciphers. The core idea of RAFA is to expedite SAT solvers by modifying the algebraic systems, which is accomplished via two methods. The first method introduces redundant constraints, which is proposed for the first time in the context of algebraic cryptanalysis. The second one is a sophisticated linearization of the nonlinear *Algebraic Normal Form* (ANF). It takes RAFA for about 9.68 hours to attack AES-128. To the best of our knowledge, this is the first work that uses a general SAT solver to attack AES with only a single injection of byte-fault. Moreover, RAFA can attack AES-128 in 50.92 and 27.54 minutes for nibble- and bit-based fault model, respectively. In comparison, the traditional DFA algorithm implemented by pure C takes $4 \sim 5$ hours under all three fault models investigated in this work. Moreover, in order to show the generality of RAFA, we also apply it to other heavyweight block ciphers. The best results show that RAFA could recover the key of Serpent-256 and `SPEEDY-r-192` in 20.7 and 1.5 hours using only three faults, respectively. In comparison, AFA could not break these two ciphers even when 30 bits and 50 bits of their keys are known, respectively. Furthermore, no DFA work on Serpent or `SPEEDY` is known using comparable fault models.

Keywords: AES-128 · DFA · AFA · RAFA · SAT · Redundant Constraints · Linearization.

1 Introduction

Fault Attack is a class of active attacks in which an attacker injects faults into the cryptosystem before gathering the ciphertexts to recover the secret key. In the first stage (referred to as *Fault Injection*), these faults can be produced by altering the power supply voltage, the frequency of the external clock or by exposing the circuits to lasers during computing [BECN+06]. In the second stage (abbreviated as FA for *Fault Analysis*), the

†The corresponding author

Received: 2023-01-15 Accepted: 2023-03-15 Published: 2023-06-09

attacker conducts cryptanalysis on the known information in order to retrieve the key. FA was first proposed by Boneh *et al.* in 1997 [BDL97].

When analyzing block ciphers, the most popular choice for fault analysis is the *Differential Fault Analysis* (DFA) [BS97], which combines fault analysis with differential cryptanalysis. Typically, in the context of DFA, the faults are injected into the intermediate states close to the last round. Using a laser to manipulate the electronic circuit, Bar-El *et al.* [BECN+06] effectively injected a fault into an intermediate state as the AES program was executing, proving that DFA is a physically feasible threat to block ciphers. Since its proposal, DFA has gained much attention. In the early years of DFA research, substantial exploitations were conducted to examine AES in particular [BS03, DLV03, Gir04, AMT13]. Many types of DFA required multiple fault injections. In [TMA11], Tunstall *et al.* proposed a DFA algorithm on AES-128 using a single fault injection. In general, DFA on block ciphers is performed mostly by manual analysis. When fault injection occurs in a deeper round, the paths of fault propagation will overlap. The intricacy of the analysis on the overlapped pathways increases exponentially, making manual analysis exceedingly difficult [ZGZ+16].

To overcome the difficulty of manual analysis in DFA, Courtois *et al.* [CJW10] proposed to combine DFA with algebraic cryptanalysis [CP02]. This technique is known as *Algebraic Fault Analysis* (AFA). In the context of AFA, the designer constructs an algebraic system by describing the relationship between the known information (such as plaintexts, ciphertexts, the fault model, etc.) and the unknown variables (such as the round keys), then a machine solver is used to automatically recover the secret key. Due to their relatively basic architectures, AFA can effectively solve algebraic systems for lightweight stream ciphers like Trivium [MBB11] and block ciphers like LED [ZGZ+12] and DES [ZGZ+16].

However, when confronted with heavyweight block ciphers such as AES and SM4 [LJH+07], where the key lengths, block sizes and S-boxes are large, AFA is currently encountering a number of obstacles, most notably its considerable runtime of solving the algebraic system. Larger block ciphers result in more complicated algebraic systems, which impedes the viability of a general-purpose solution scheme despite the development of machine solvers.

In summary, AFA is preferable in terms of its generality, but is ineffective against heavyweight block ciphers. This motivates us to improve its efficiency while maintaining its generality. Another motivation of this work is the belief that since DFA can be considered a special solution scheme for the algebraic system constructed in AFA derived through manual analysis, AFA will be comparable to DFA if the algebraic systems accommodate the machine solvers more effectively. Therefore, we attempt to expedite AFA by transforming algebraic systems into equivalent ones, on which the performance of machine solvers will be enhanced.

Our Contribution. In this paper, we outline the limitations of AFA on heavyweight block ciphers (such as AES) and propose a generic variant of AFA to improve its efficiency. The main contributions of this work can be summarized as follows:

(1) We propose the *Redundancies-assisted Algebraic Fault Analysis* (RAFA), whose performance significantly surpasses that of previous AFA. Our RAFA framework takes about 9.68 hours on average to recover the key when implemented on AES-128 with the *byte-based fault model in the 8-th round* used in [Muk09], which requires a single injection of a byte-fault. While AFA outperforms DFA on several lightweight block ciphers, to the best of our knowledge, there are no reports of AFA with a single fault injection being effective on a heavyweight block cipher such as AES-128. When the key length, the block size, and the size of the S-boxes increase, the system becomes significantly more complicated, resulting in drastic decline in the performance of AFA. However, our RAFA algorithm can circumvent this difficulty. Moreover, in order to verify its generality, we apply RAFA to other heavyweight SPN

block ciphers. The best results show that RAFA could recover the key of Serpent-256 and `SPEEDY-r-192` in 20.7 and 1.5 hours using only three faults, respectively.

(2) We apply the concept of redundant constraints to the field of algebraic cryptanalysis. We note that the major objective of SAT-based algebraic cryptanalysis is the construction of an encoding for transforming *Constraint Satisfaction Problem* (CSP) into *Boolean Satisfiability Problem* (SAT). This conceptual transition motivates us to integrate redundant constraints into the system, a strategy recently studied in the research of converting CSP into SAT [BHN14, Gav07, MS11, Bjö11]. Using redundant constraints, RAFA becomes thousands of times more efficient than AFA. When discussing the mechanism of incorporating redundant constraints, we will focus on AES-128 to clarify the algorithms. But after that we will provide general algorithms suitable for common SPN block ciphers, and conduct experiments to show RAFA's effectiveness on other block ciphers.

(3) We demonstrate the adaptability and efficiency of the proposed RAFA using different fault models. Specifically, the performance of RAFA is improved when there are fewer solutions to the algebraic system as a result of adopting slightly stronger fault models. In contrast, the performance of non-algebraic analyses such as DFA tends to be independent of the number of solutions if the algorithms are not adjusted manually, as the search spaces of the first and second step are invariant. We validate this assertion on AES-128 since it has known effective DFA algorithm with two slightly stronger fault models, in which the expected number of solutions to the algebraic system decreases from 256 to 16 or 8. Experimental results show that our RAFA algorithm can complete in 50.92 or 27.54 minutes in these two scenarios, respectively. In comparison, the traditional DFA algorithm implemented by pure C takes $4 \sim 5$ hours under all three fault models investigated in this work.

2 Preliminary

2.1 Advanced Encryption Standard

Since October 2000, the *Advanced Encryption Standard* (AES) [DR99] has been the de facto standard for symmetric key cryptography. Due to its great resistance to common block cipher attacks [BS12, Mat93], the AES is extensively utilized in smart cards and secure microprocessors. Due to its prevalence, numerous types of cryptanalysis of AES have been documented in the literature.

There are three versions of AES, namely AES-128, AES-192 and AES-256, each with a different number of rounds and key length. In this paper, AES-128 is the principal topic of discussion. In AES, each of the round keys and intermediate data is represented by a 4×4 array of bytes called the state matrix.

The architecture of AES is based on the *Substitution-Permutation Network* (SPN). Each round of AES, excluding the final one, consists of a nonlinear operation `SubBytes` (SB) and two linear operations `ShiftRows` (SR) and `MixColumns` (MC), followed by `AddRoundKey` (AK), the single operation directly connected to the key operation. Note that each of these operations is reversible. In the decryption, there are hence corresponding inversive operations SB^{-1}, SR^{-1}, MC^{-1} and AK^{-1}.

2.2 Differential Fault Analysis on AES-128

Notation. In this paper, we will use the following notations:

- $SB_{r,i}, SB^*_{r,i}$: the i-th byte of the correct (*resp.*, faulty) output state of the `SubBytes` operation at r-th round, $0 \leq i < 16, 0 \leq r \leq 10$.

- $SR_{r,i}, SR^*_{r,i}$: similarly, SR stands for `ShiftRows`.
- $MC_{r,i}, MC^*_{r,i}$: similarly, MC stands for `MixColumns`.
- $AK_{r,i}, AK^*_{r,i}$: similarly, AK stands for `AddRoundKey`. In particular, $x_i := AK_{10,i}$ denotes the correct ciphertext and $x^*_i := AK^*_{10,i}$ denotes the faulty ciphertext.
- $K^{(r)}_i$: the i-th byte of the r-th round key.

The DFA algorithm consists of two steps:

(1) In the first step, parts of the constraints imposed by information leakage resulting from fault injection are recognized. In particular, each constraint equation in the first step should have as few variables as feasible, so that brute-force algorithms may quickly solve these relationships. Generally, the first step functions as a filter for candidate keys.

(2) In the second step, which is the most time-consuming, an exhaustive search will be conducted on the reduced key space generated in the first step, so that a very small key space whose elements satisfy certain sophisticated constraints will be obtained.

The fault model employed in [Muk09, TMA11] injects a random byte fault f at the input of the 8-th round. Without loss of generality, we may presume that the fault is introduced into the 0-th byte as we can easily replicate the analysis in other scenarios. In addition, since the `SubBytes` and `ShiftRows` operations will not diffuse the fault in the 0-th byte, we may presume that the fault is introduced into the 0-th byte of the output of the `ShiftRows` in the 8-th round.

Appendix A lists the equations for the first step, where the values F_0, F_1, F_2, F_3 are unknown intermediate fault values. In the first step, there are 4 sets of equations, each of which comprises four bytes of the last round key as unknown variables. Using a basic brute-force algorithm, the solutions to each set of equations can be found in seconds.

Theoretically, the size of key space is reduced from 2^{128} to 2^{32} by the first step analysis. To further restrict the key space, the authors of [TMA11] developed a set of additional equations for the second step, which are found by evaluating the fault values at a deeper round, where f is the unknown injected fault value, as shown in Eq. (1).

$$\begin{cases} 2f = AK_{8,0} \oplus AK^*_{8,0} \\ f = AK_{8,1} \oplus AK^*_{8,1} \\ f = AK_{8,2} \oplus AK^*_{8,2} \\ 3f = AK_{8,3} \oplus AK^*_{8,3} \end{cases} \tag{1}$$

This set of constraints is then solved by exploring the reduced key space acquired in the first step. Once an assignment of the last round key is set, the values appeared in the equations are uniquely determined. A candidate key is a value of the last round key that satisfies the second step constraints. According to a probabilistic reasoning, the expected number of candidates following the second step is 2^8 [TMA11]. Using the plaintext and the correct ciphertext, it is then possible to recover the master key. Notice that checking if an assignment of the last round key satisfies Eq. (1) is several times faster than attempting to encrypt the plaintext with the supplied last round key, due to the decline in the number of rounds that need to be considered. The complexity of solving constraints in Eq. (1) via brute-force algorithm can be characterized by 2^{32}. However, we remark that Charles *et al.* [BDF11] developed an automatic tool to search for a dedicated solution scheme of these two sets of constraints, which managed to reduce this complexity to 2^{24}.

2.3 Algebraic Fault Analysis

Algebraic Fault Analysis (AFA) was first proposed in [CJW10] as a mechanism for augmenting algebraic cryptanalysis with fault analysis. It incorporates both the correct and faulty encryption procedures into the algebraic system, in order to utilize the known plaintexts, correct and faulty ciphertexts. This concept was refined in [CFGR12] and implemented in [MBB11] to attack the stream cipher Trivium. Zhang *et al.* proposed a generic framework of AFA in [ZGZ+16], which successfully break the ultra-lightweight block cipher LBlock, and was applied to block ciphers with different structures, including DES, PRESENT and Twofish.

The AFA's algebraic system comprises a description of the encryption and fault information. There are two disparate principles of constructing the algebraic system. The first principle is to minimize the number of variables and equations [JKP12], in order to reduce the instance size. The second one is to include all the intermediate variables and encryption operations as described in [ZGZ+16], which makes the analysis simple and generic. In this work, we will demonstrate the efficiency and usefulness of adopting the second principle.

Once the algebraic system has been established, it can be translated to Boolean constraints and ultimately solved by SAT solvers. Despite its theoretical complexity [Coo71], remarkable improvements of SAT solvers [BHZ06, Hua08, Pet15, TTKB09] have made SAT a viable foundation for addressing a wide range of problems. In addition, as a result of the annual competition [SAT], SAT solvers are meticulously constructed to operate in different contexts with no user adjusting necessary. As a consequence, there is a growing interest in incorporating them to the problem-solving process. Indeed, SAT solvers have recently become the standard for the majority of algebraic cryptanalysis solutions.

Most SAT solvers are based on the modest code base of MiniSAT [ES03] and take as input DIMACS files, which contain lists of disjunction clauses. Furthermore, because XOR operations are so prevalent in cryptographic applications, extensive research has been conducted to translate linear equations over $GF(2)$ into *Conjunctive Normal Form* (CNF) clauses [BCJ07, SNC09]. In this work, we employ the CryptoMiniSAT [SNC09], which extends the MiniSAT solver [SE05] to allow the input to contain XOR clauses.

3 Overview

3.1 Motivation and Challenges

The principal motivation of this work is to discover a variant of AFA that is generic and efficient. While DFA described in Sec. 2.2 works well on AES-128, this framework is not generic enough to be replicated on other block ciphers: the adversary must manually build the constraint equations associated with unknown intermediate fault values, in a manner that each equation in the first step uses as few available bytes of the round keys as possible. In the meantime, the first step should be able to significantly limit the key search space, making the second step's brute-force search feasible. In other words, an elaborate balance has to be maintained when designing the two sets of constraints.

On the other hand, even though AFA employs a generic algebraic method with the use of machine solvers, it only works well on lightweight block ciphers, where the algebraic systems are tractable because of smaller key length, block size and S-box. Indeed, to the best of our knowledge, there are no reports of AFA with a single fault injection exploiting heavyweight block ciphers such as AES-128. An example of AFA on AES-128 was given in [vWBV+17], which, however, required at least two fault injections. And the faults were introduced at the input of the 9-th round, hence the workload for fault analysis was rather light and not comparable to the one we are evaluating (*i.e.* a single fault injection to

the 8-th round). In order to make AFA work on heavyweight block ciphers, we need to improve its efficiency to overcome the difficulty of increasing instance sizes.

3.2 Core Ideas

From our perspective, the primary objective of SAT-based algebraic cryptanalysis is to design an encoding for transforming *Constraint Satisfaction Problem* (CSP) into *Boolean Satisfiability Problem* (SAT). Recent trends in this math-related field include the introduction of so-called redundant constraints [BHN14, Gav07, MS11, Bjö11]. Consequently, we are inspired by the works of hybrid encodings [SBT17, BHN14], which combine different encoding strategies, such as direct encoding [Wal00], log encoding [IM94] and order encoding [TTKB09], into a single SAT instance. In this work, we propose the *Redundancies-assisted Algebraic Fault Analysis* (RAFA), in which the insertion of redundant constraints acts as the main component.

A potential issue of incorporating redundant constraints into the system is the increase in the instance size. However, Björk *et al.* [Bjö11] found only a minor link between the instance size and the difficulty for practical SAT solvers. They asserted that the number of *primary variables*, whose assignment alone will determine the assignment of all variables, is a much more significant factor than the number of variables and the number of clauses. For instance, the bits of the last round key can be considered the primary variables since they uniquely determine the entire system. Moreover, having more clauses tends to facilitate the unit propagation, allowing SAT solvers to finish more quickly [Bjö11]. This ensures us to focus more on the introduction of redundant constraints and less on the instance size.

In order to maintain the generality of the RAFA framework, we will not delve into the specific properties of the AES-128 block cipher, instead, we will provide two strategies to exploit the knowledge we may acquire from the fault propagation (Sec. 4.3.1) and the SPN structure (Sec. 4.3.3), which provides the redundant constraints.

1. Utilizing the information provided by the path of fault propagation. Due to the nature of the SPN structure, a diagram of fault propagation can be constructed once a fault model has been defined. A sequence of equations can also be written to describe the fault values (which may be zero) in the diagram, which aids the SAT solvers in identifying nontrivial interactions caused by fault injection.

2. Utilizing the invertibility of each SPN layer. For a block cipher of SPN structure, each operation must be invertible in order for an authentic decryption. We make use of this fact and equip the algebraic system with both encryption and decryption operations. This strategy improves the implicativity of SAT solvers by supporting inferences on both directions.

3.3 Fault Models on AES-128

We discover that the performance of the RAFA algorithm is highly related to the number of solutions due to the bottleneck of modern SAT solvers. To clarify this point, we evaluate RAFA using various fault models. We call the fault model discussed in Sec. 2.2 the *byte-based fault model in the 8-th round*, where a random byte fault is induced in the 0-th byte of the output of the 8-th `ShiftRows`. Suppose the induced fault value is f, then $1 \leq f < 256$ in the *byte-based fault model in the 8-th round*. Two slightly stronger fault models are also evaluated in this work. The first one is the *nibble-based fault model in the 8-th round*, where it is known that the fault is injected into the first nibble (4 bits) of the same byte, *i.e.* $1 \leq f < 16$; the second one is the *bit-based fault model in the 8-th round*, where it is known that only one bit of the faulty byte is flipped, *i.e.* $f \in \{1, 2, 4, 8, 16, 32, 64, 128\}$. The last two fault models help us reduce the number of candidate keys and thus result in significant acceleration of the RAFA algorithm as shown by the experiments.

4 Framework of RAFA

4.1 Overview

The comparison between the RAFA framework and the AFA framework is demonstrated in Fig. 1, where the red dashed block depicts the procedures of RAFA and the blue one describes the procedures of AFA.

As for AFA, it starts with including the known values of the correct and faulty ciphertext into the algebraic system in ①. Then, equations describing the fault values based on the given fault model are established in ②, followed by a full encryption of the known plaintext in ④. The relationships between the known values are described by a sequence of forward constraints in ⑤, which include all the intermediate values and operations. In order to linearize the nonlinear components, only *Standard Strategy* (SS, see Sec. 4.4) is adopted in ⑧. Finally, the algebraic system of AFA is solved by a SAT solver to recover the key.

As for RAFA, it starts with the same procedure ①. However, instead of merely considering the injected fault value, it will establish equations describing all the intermediate fault values according to the fault propagation in ②. Afterwards, in addition to the forward constraints in ⑤, RAFA also adds the backward constraints in ⑥, which corresponds to the decryption operations. Moreover, prior to completing the linearization by SS in ⑧, further linearizing strategies are adopted in ⑦ (see Sec. 4.4). The algebraic system of RAFA no longer requires the plaintext information during the online equation construction. Instead, the plaintext information is merely used offline to check whether each output of the SAT solver is correct or not by feeding the pair of correct plaintext and ciphertext.

In a glimpse, the proposed RAFA discards direct usage of plaintext ④ in the equation construction. Additionally, RAFA utilizes the techniques of incorporating redundant constraints (③ and ⑥), which is an augmentation of ②, and employing further linearizing strategies (⑦) to facilitate the entire solving. All of these make it possible to solve the algebraic equation of AES-128 with a single fault injection in the 8-th round.

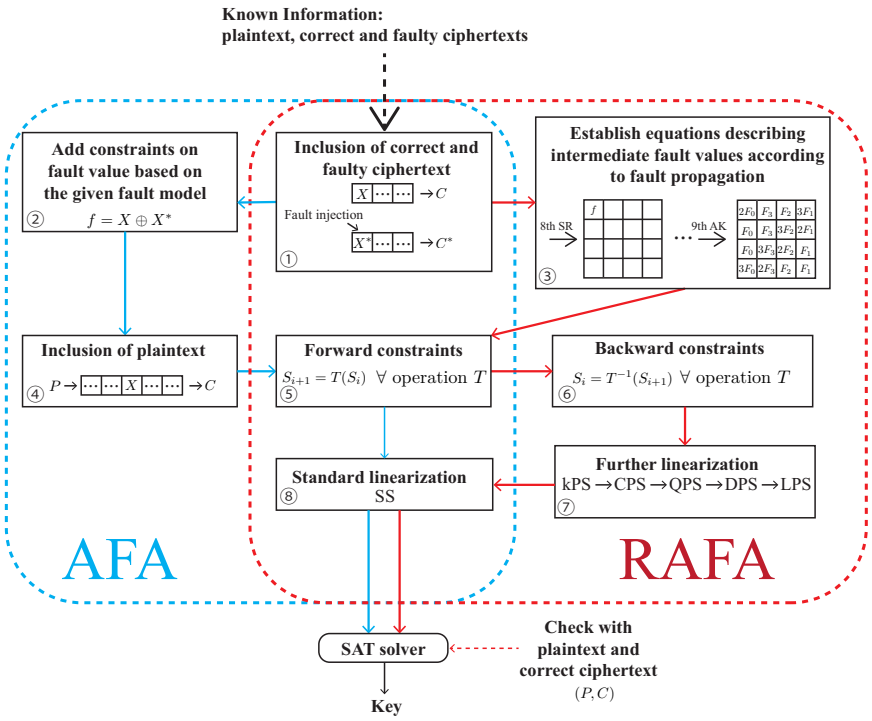

Figure 1: Comparison between RAFA Framework (Red) and AFA Framework (Blue).

4.2 Construct Basic Algebraic System

Firstly, we construct the basic algebraic system of RAFA, which corresponds to the procedures ① and ⑤ in Fig. 1. We will focus our discussion on AES-128 for convenience, but the algorithm is replicable on other SPN block ciphers by replacing the encryption operations and the `Key Expansion` scheme.

The basic structure of the AES algorithm was given in Sec. 2.1. In the following we recall the precise design of each operation, as specified in [DR99]. We follow the convention that each byte is represented by a polynomial

$$\sum_{i=0}^{7} a_i X^i \in \frac{GF(2)[X]}{(X^8 + X^4 + X^3 + X + 1)} \cong GF(2^8),$$

where a_0 is the least significant bit.

4.2.1 Encryption.

The operation `ShiftRows` and `MixColumns` can be characterized by two 128×128 0-1 matrices if we view each state as a column vector of 128 bits over $GF(2)$. The operation `AddRoundKey` is simply adding the state vector with the round key vector.

The operation `SubBytes` is the only nonlinear component in AES. It replaces each byte in a state with its image under an invertible S-box mapping. Since we need eventually to encode the relations in bits before feeding them into the SAT solver, it is helpful to firstly transform the output of S-box into Algebraic Normal Form (ANF) [CB07], *i.e.* representing each output bit of the S-box in terms of a multivariate polynomial of the input bits. As the ANF of a Boolean function is known to be unique [SP09], we do not worry about how it is computed. There are many open-source computer algebra tools that can calculate ANF [MSP+17, The20].

4.2.2 Key Expansion

The encryption operations are invertible, hence the key of a single round is sufficient to determine the full round keys in AES-128. In general, the primary concern is the last round key, as it is the closest to the known output ciphertexts. Since the fault injection happens in the 8-th round, and we do not include verification equations with plaintext into the system according to Sec. 3.2, the only round keys we may utilize are $(K_i^{(8)}), (K_i^{(9)}), (K_i^{(10)})$ (see the notation in Sec. 2.2). We take them as variables and add the corresponding constraints according to the `Key Expansion` scheme , without considering other round keys.

4.2.3 Reservation of Round Function Operations

We adopt the convention of keeping the intermediates in the algebraic system as in [ZGZ+16], *i.e.* when a state s_0 is input to a sequence of operations $f_1, f_2, \cdots, f_n$, the algebraic system would contain $s_i = f_i(s_{i-1})$ for all $1 \leq i \leq n$, instead of simply $s_n = f_n \circ \cdots \circ f_1(s_0)$. While Zhang *et al.* adopt this convention primarily to simplify the analysis, making it generic enough to deploy the framework on any block cipher, we discover that this representation actually boosts the SAT problem-solving process, which may be explained by the fact that linearity of the system is maintained [SNC09] and the implicativity of SAT solvers is enhanced due to the decrease in the length of the generated CNF clauses [Bjö11]:

- Firstly, the linearity of the system is maintained. Due to the nonlinear operation `SubBytes`, all constraints become nonlinear multivariate polynomials if the operations are compacted. In this situation, each linear operation is utilized exactly once during

the substitutions, as can be seen. However, if the linear equations are kept in the system, they can be accessed multiple times (in form of CNF clauses), allowing SAT solvers to detect conflicts at an earlier stage. In addition, CryptoMiniSAT [SNC09] exploits the additional information offered by linear equations to a great extent.

- Secondly, the length of the equations is reduced. Consequently, the generated CNF clauses are shorter, resulting in higher implicativity [Bjö11]. Given that the key spaces of block ciphers are so vast, the density of solutions is quite close to zero. As a result, SAT solvers are likely to encounter a conflict in the majority of search branches. Higher implicativity permits early detection of conflicts, which expedites the SAT solvers.

Moreover, a key distinction between the basic algebraic system of RAFA and the one of AFA proposed in [ZGZ+16] is that the plaintext information is not included into the system, making the algebraic system of RAFA much simpler. In AFA, the incorporation of fault information was regarded as an enhancement over the primitive algebraic analysis. Indeed, the AFA framework described in [ZGZ+16] constructs an algebraic system that describes not only relationships between the correct and faulty ciphertext, but also relationships between the plaintext and the correct ciphertext. Since the fault injection occurs at a distance from the plaintext, it is necessary to add constraint equations describing many rounds of encryption to the algebraic system in order to include the plaintext; however, the expense of doing so significantly outweighs the benefit of knowing the plaintext, in the context of heavyweight block ciphers. Therefore, in this work, the plaintext information has been omitted and thus the algebraic system remains incomplete. In other words, it may contain several solutions that match to different candidate keys during the online solving phase. During the offline phase, after identifying a candidate key for the master key, its correctness can be immediately verified by encryption the plaintext using the candidate key.

Besides, we wish to highlight one additional advantage of retaining the intermediates. That is, it enables us to easily introduce redundant constraints into the RAFA framework, as mentioned in Sec. 4.3.

4.3 Incorporate Redundant Constraints into the System

At this stage, we illustrate how to incorporate redundant constraints into the algebraic system, which corresponds to procedures ③ and ⑥ in Fig. 1.

As specified in Sec. 3.2, the main task of algebraic cryptanalysis is transforming CSP into SAT. We identify that one of the most significant challenges of this task is the loss of information. For example, suppose x, y, z are Boolean variables, consider the following constraints:

$$z = x \wedge y, y = \overline{x} \tag{2}$$

Based on inferences on the CSP instance, we obtain $z = x \wedge \overline{x} = 0$.

The constraints in Eq. 2 can be expressed in CNF via logics:

$$z \leftrightarrow x \wedge y \equiv (\overline{z} \vee x) \wedge (\overline{z} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z)$$
$$y \leftrightarrow \overline{x} \equiv (\overline{y} \vee \overline{x}) \wedge (x \vee y)$$

Thus, when they are directly encoded into SAT instance, the following clauses are generated: (i) $\overline{z} \vee x$; (ii) $\overline{z} \vee y$; (iii) $\overline{x} \vee \overline{y} \vee z$; (iv) $\overline{y} \vee \overline{x}$; (v) $y \vee x$.

A SAT solver may first try $z = 1$, which results in $x = y = 1$ via (i) and (ii), then a contradiction is found in (iv). The SAT solver can then decide $z = 0$. We can see that the SAT solving can be accelerated if we include the "lost information" $z = 0$ into the instance, since the branch $z = 1$ does not need to be searched now. The constraint $z = 0$ is redundant because it can be derived from the existing constraints.

To summarize, since CSP instance is in general described in an abstract manner, its "nice" structure (*e.g.* $z = x \wedge \overline{x} = 0$) might be completely lost after being converted to SAT [Pet15]. This motivates us to utilize the knowledge gained from the CSP instances and feed it into corresponding SAT instances, which is accomplished by incorporating redundant constraints into the system in this paper. The redundant constraints arise from the fault propagation and the SPN structure as specified in the following.

4.3.1 Fault Propagation on AES-128

The only information about the *byte-based fault model in the 8-th round* that has been utilized when constructing the basic algebraic system in Sec. 4.2 is the position of the fault injection, which specifies the starting point for constructing the basic algebraic systems. Nevertheless, additional information can be obtained from the fault model, mostly via constraints describing intermediate fault values generated by the diffusion layers of block ciphers.

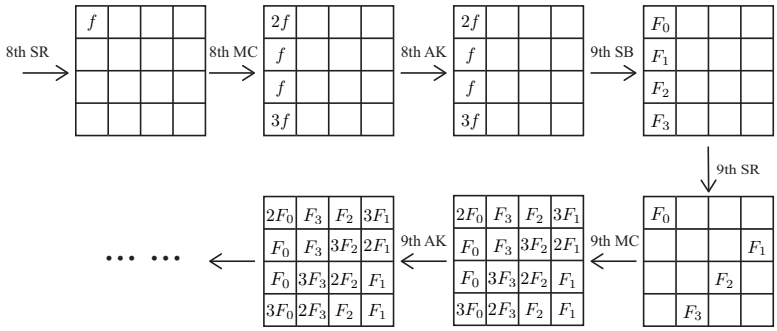

Figure 2: Diagram of Fault Propagation from 8th SR.

For most block ciphers and known fault models, a fault propagation diagram similar to that in Fig. 2 can be easily determined. We can derive two types of relations from the diagram of fault propagation:

(1) Differences on the clean boxes in Fig. 2. By considering the bytes where faults have not been spread, we have the following equations:

$$\begin{aligned} SR_{8,i} &= SR^*_{8,i}, \forall i \geq 1 \\ MC_{8,i} &= MC^*_{8,i}, \forall i \geq 4 \\ AK_{8,i} &= AK^*_{8,i}, \forall i \geq 4 \\ SB_{9,i} &= SB^*_{9,i}, \forall i \geq 4 \\ SR_{9,i} &= SR^*_{9,i}, \forall i \neq 0, 7, 10, 13 \end{aligned}$$

(2) Differences on the labelled boxes in Fig. 2, whose fault values are related to some intermediate faults appearing multiple times. Faults after linear layers `ShiftRows`, `MixColumns`, `AddRoundKey` are easily calculated by linearity. Fault after nonlinear layers `SubBytes` are substituted by new fault variables. In Appendix B, we list the equations calculating the fault values related to f, F_0, F_1, F_2, F_3.

4.3.2 Fault Propagation on General SPN Block Ciphers

In general, the common operations of SPN block ciphers are linear transformation, S-box, and `AddRoundKey`. Suppose the block size is n, then each linear transformation corresponds

to an $n \times n$ 0-1 matrix. Suppose the size of an S-box is k, WLOG we can assume it maps the ik-, $(ik+1)$-, ..., $(ik+k-1)$-th input bits into the ik-, $(ik+1)$-, ..., $(ik+k-1)$-th output bits for any $i = 0, 1, \cdots, n/k-1$, because we can append permutations before and after the S-box operation.

Now we can consider the property of fault propagation on general SPN block ciphers. Because of linearity, the input and output faults of a linear operation are linearly related. The i-th output of an S-box is clean if and only if the ik-, $(ik+1)$-, $\cdots$, $(ik+k-1)$-th bits are clean, otherwise we would leave the output fault alone since the input and output faults are related with the correct input and faulty input through high-degree ANF. The `AddRoundKey` operation preserves the input and output fault because the key is invariant.

Furthermore, we note that when multiple faults are injected, we would add such redundancies for every pair of the encryptions. The redundancies related to fault propagation is summarized as follows. In the pseudo-code, we will use := for definition of Boolean variable, i.e. the corresponding equation is fed into the SAT solver; we will use $\leftarrow$ for the usual assignment.

Algorithm 1: Redundancies related to Fault Propagation.

for Every pair of the encryptions **do**
  Suppose the initial states are S_1 and S_2
  Define $f := S_1 \oplus S_2$
  for $i = 0, 1, \cdots, n-1$ **do**
    if the fault is not injected into the i-th bit in the adopted model **then**
      $f_i := 0$
      $f_i \leftarrow 0$
  for operation T starting from the fault position **do**
    $S'_1 = T(S_1), S'_2 = T(S_2)$ `// state variables are defined when constructing the basic algebraic systems`
    if T is a linear operation **then**
      Suppose the corresponding matrix is L.
      Define $f' := S'_1 \oplus S'_2$
      for $i = 0, 1, \cdots, n-1$ **do**
        $eq \leftarrow f'_i$
        for $j = 0, 1, \cdots, n-1$ **do**
          if $L_{i,j} = 1$ and $f_j \neq 0$ **then**
            $eq \leftarrow eq \oplus f_j$
        Add equality $eq := 0$
        if $eq = f'_i$ **then**
          $f'_i \leftarrow 0$
    else if T is an S-box operation **then**
      Define $f' := S'_1 \oplus S'_2$
      for $i = 0, 1, \cdots, n/k-1$ **do**
        if $f_{ik} = f_{ik+1} = \cdots = f_{ik+k-1} = 0$ **then**
          for $j = 0, 1, \cdots, k-1$ **do**
            $f'_{ik+j} := 0$
            $f'_{ik+j} \leftarrow 0$
    else if T is `AddRoundKey` **then**
      $f' \leftarrow f$
    $f \leftarrow f', S_1 \leftarrow S'_1, S_2 \leftarrow S'_2$

4.3.3 Enhancement of Forward-Backward Communication

Since all encryption operations in a block cipher with an SPN structure are invertible, we can include both encryption operations and decryption operations in the system to boost SAT solver inferences in both directions. Suppose the input state and output state of an encryption operation T other than `AddRoundKey` are S_i and S_{i+1}, respectively. Both S_i and S_{i+1} are variables included in the algebraic system as described in Sec. 4.2. And we can append the following constraints into the system:

$$\begin{aligned} S_{i+1} &= T(S_i) \\ S_i &= T^{-1}(S_{i+1}) \end{aligned}$$

The inverse of a linear operation is just the inverse of a matrix over $GF(2)$. The inverse S-box can be handled by strategies mentioned in Sec. 4.4. Even though the search process of SAT solvers is conducted on the variables according to the ordering of their indices [Bjö11], the determination of the variables during the search does not have to be sequential. For example, it may happen that when the solver is running, the byte value of S_i is fixed, then it can resort to the relation $S_{i+1} = T(S_i)$ to uniquely identify the value of S_{i+1} via unit propagation. And the opposite may happen, where the relation $S_i = T^{-1}(S_{i+1})$ becomes helpful. Although the actual scenario is more complicated, as the relations are encoded in CNF clauses, this argument shows an insight of functionality of the Forward-Backward Communication. Note that such redundancies are naturally applicable to all SPN block ciphers.

4.4 Handle Nonlinear ANF by Linearization

This section demonstrates the procedures ⑦ and ⑧ in Fig. 1.

After constructing the basic algebraic system in Sec. 4.2 and introducing redundant constraints in Sec. 4.3, we obtain a system of equations which can be divided into two parts:

1. Nonlinear multivariate polynomials over $GF(2)$, *i.e.* the ANF representing the nonlinear S-box.
2. Linear equations over $GF(2)$, *i.e.* in the form of $x_1 \oplus x_2 \oplus \cdots \oplus x_n = c$, where $c \in GF(2)$ is a constant.

Most SAT solvers take CNF clauses as input, which means we have to further transform all constraints into a standard format. However, XOR operations are so ubiquitous in cryptographic problems that extensive research has been done to transform linear equations over $GF(2)$ into CNF clauses [BCJ07, SNC09]. In this paper, we utilize the CryptoMiniSAT [SNC09], which extends the MiniSAT solver [SE05] to accept linear equations over $GF(2)$ (XOR operations) as input, besides CNF clauses. Therefore, we only focus on how to transform the nonlinear ANF into linear equations and CNF clauses.

A basic idea for linearizing a nonlinear ANF is to introduce an auxiliary variable a for each monomial $x_1x_2\cdots x_n$ of order ≥ 2, resulting in a system of linear equations. The equivalence of the monomial and the auxiliary variable a is described by the following CNF formula:

$$\begin{aligned} & a \leftrightarrow x_1x_2\cdots x_n \\ \equiv & (a \rightarrow x_1 \wedge \cdots \wedge x_n) \wedge (x_1 \wedge \cdots \wedge x_n \rightarrow a) \\ \equiv & (\overline{a} \vee x_1) \wedge (\overline{a} \vee x_2) \wedge \cdots \wedge (\overline{a} \vee x_n) \wedge (\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_n} \vee a) \end{aligned}$$

We can replace every monomial $x_1x_2\cdots x_n$ with a by appending the preceding CNF clauses to our algebraic system. When all nonlinear monomials are eliminated, linearization

is completed. It is called *Standard Strategy* (SS) in [JK10], which is a common convention adopted in AFA [ZGZ+16]. However, since the nonlinearity of AES S-box dominates, it is costly to include an auxiliary variable for each monomial of order ≥ 2.

In order to decrease the length of the output linear equations and the number of auxiliary variables, several supplementary strategies have been proposed in [JK10].

Cubic Partner Strategy (CPS). Replace $x_1x_2x_3 \oplus x_1x_2x_4$ with an auxiliary variable $a \in GF(2)$, appending the following CNF clauses:

$$\begin{aligned}&(x_1 \vee \overline{a}) \wedge (x_2 \vee \overline{a}) \wedge (x_3 \vee x_4 \vee \overline{a}) \wedge (\overline{x_3} \vee \overline{x_4} \vee \overline{a})\\ &\wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee a)\\ &\wedge (\overline{x_1} \vee \overline{x_2} \vee x_3 \vee \overline{x_4} \vee a)\end{aligned}$$

Quadratic Partner Strategy (QPS). Replace $x_1x_2 \oplus x_1x_3$ with an auxiliary variable $a \in GF(2)$, appending the following CNF clauses:

$$\begin{aligned}&(x_1 \vee \overline{a}) \wedge (x_2 \vee x_3 \vee \overline{a}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{a})\\ &\wedge(\overline{x_1} \vee \overline{x_2} \vee x_3 \vee a) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee a)\end{aligned}$$

Double Partner Strategy (DPS). Replace $x_1x_2 \oplus x_1 \oplus x_2 \oplus 1$ with an auxiliary variable $a \in GF(2)$, appending the following CNF clauses:

$$(\overline{a} \vee \overline{x_1}) \wedge (\overline{a} \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee a)$$

Linear Partner Strategy (LPS). Replace $x_1x_2 \oplus x_1$ with an auxiliary variable $a \in GF(2)$, appending the following CNF clauses:

$$(\overline{a} \vee x_1) \wedge (\overline{a} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee a)$$

Inspired by these strategies, we here propose the following strategy to further reduce monomials of higher order.

n-Partner Strategy (nPS). For any $n \geq 2$, replace $x_1x_2 \cdots x_n \oplus x_2 \cdots x_n$ with an auxiliary variable $a \in GF(2)$, appending the following CNF clauses:

$$\begin{aligned}&(\overline{a} \vee \overline{x_1}) \wedge (\overline{a} \vee x_2) \wedge (\overline{a} \vee x_3) \vee \cdots \wedge (\overline{a} \vee x_n)\\ &\wedge(a \vee x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \cdots \vee \overline{x_n})\end{aligned}$$

In RAFA, we linearize the ANF of S-box using `nPS`, `CPS`, `QPS`, `DPS`, `LPS` and `SS`, then feed the solver with auxiliary CNF clauses and the consequent linear system.

4.5 RAFA in a Nutshell

Finally, our RAFA scheme can be summarized as follows.

Algorithm 2: RAFA Scheme.

input : Plaintext P, correct ciphertext C_1 and a set of faulty ciphertexts $\{C_2, \cdots, C_m\}$

```
// Construct basic algebraic system including intermediates
for i = 1, 2, ··· , m do
    Start from the state of fault injection, declare the state S_i
    for operation T from current position to the output do
        Declare S'_i to be variables representing the intermediate outputs
        // Forward Backward
        Add forward constraints S'_i := T(S_i)
        Add backward constraints S_i := T^{-1}(S'_i)
        S_i ← S'_i
    Add constraints S_i := C_i; // inclusion of known information
KeyConstraints()
RedundantFP() // redundancies from fault propagation
nPS() // linearize x_1x_2···x_n ⊕ x_2···x_n, ∀n ≥ 2
CPS() // linearize x_1x_2x_3 ⊕ x_1x_2x_4
QPS() // linearize x_1x_2 ⊕ x_1x_3
DPS() // linearize x_1x_2 ⊕ x_1 ⊕ x_2 ⊕ 1
LPS() // linearize x_1x_2 ⊕ x_1
SS() // complete linearization
while The solver generates a candidate key K do
    C' ← Encrypt(P, K)
    if C' = C_1 then
        return K
    else
        Block the solution K.
```

The procedure `KeyConstraints` adds constraints representing the relationships between previously used round keys according to the key expansion algorithm . The procedure `RedundantFP` adds redundant constraints derived from the fault propagation as described in Sec. 4.3.1. The procedures `nPS`, `CPS`, `QPS`, `DPS`, `LPS`, `SS` are linearizing strategies discussed in Sec. 4.4. When a solution to the system is found by the solver, we attempt to encrypt the plaintext P with the corresponding candidate key. If the output is C_1, the candidate key is correct; otherwise we avoid this solution by introducing blocking clauses and restart the SAT solver.

5 Evaluation

All experiments were performed on a workstation having sixteen 2.10GHz Xeon cores and 8GB of RAM. We use CryptoMiniSAT 5.8.0 as the SAT solver. Unless otherwise specified, a runtime recorded in the experiments is the average of ten random instances. Each instance has a random plaintext, key and fault value satisfying the assumed fault model. In addition, only the runtime of SAT solving is recorded, as the cost of preprocessing is stable and negligible.

5.1 Evaluation on AES-128

Some experiments may take too much time to complete, hence we provide the SAT solver with the first k bytes of the last round key in some scenarios, where $0 \leq k \leq 16$. This is equivalent to considering the first k bytes of the last round key as known.

5.1.1 Evaluation of RAFA *v.s.* AFA

In this section, the *byte-based fault model in the 8-th round* is adopted for the experiments.

In previous sections, we indicated that there are no reports of effective AFA on heavyweight block ciphers with a single fault injection, therefore we cannot execute a complete AFA on AES-128 for all k. Indeed, we can only track the runtime of AFA when $k \geq 13$.

k	Algorithm	Runtime (s)	Acceleration Ratio
13	AFA	81399.14	1.00
	RAFA	**2.20**	**36999.61**
14	AFA	1449.28	1.00
	RAFA	**2.27**	**638.45**
15	AFA	33.97	1.00
	RAFA	**1.64**	**20.71**

Table 1: RAFA *v.s.* AFA for $13 \leq k \leq 15$.

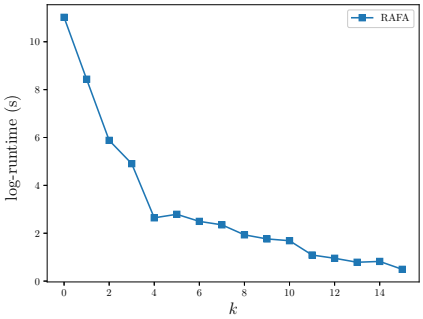

Figure 3: log-runtime of RAFA for $0 \leq k \leq 15$

In Table 1 and the followings, the metric "Acceleration Ratio" denotes the speedup over the scenario where the "Acceleration Ratio" is set to be one with the same k. For example, the "Acceleration Ratio" 36999.61 in Table 1 is computed by the quotient 81399.14/2.20. The results shown in Table 1 demonstrate a significant improvement of RAFA over AFA algorithm in the context of heavyweight block ciphers. Fig. 3 plots the log-runtime of RAFA for $0 \leq k \leq 15$.

5.1.2 Evaluation of Different Linearizing Strategies

In this section, the *byte-based fault model in the 8-th round* is adopted for the experiments.

In Sec. 4.4, we discussed several linearizing strategies to deal with nonlinear ANF. In the following experiments, we exhibit the functions of these techniques and demonstrate that our linearization combining all of them yields the best performance. Similarly, we set $k = 2$ to shorten the experiment duration time. If the "Linearizing Strategy" is SS, only SS is adopted to linearize the nonlinear ANF; if it is $LPS + SS$, LPS is first applied to the nonlinear ANF, followed by SS to complete the linearization. Likewise for $DPS+SS, QPS+SS, CPS+SS, nPS+SS$, and $nPS+CPS+QPS+DPS+LPS+SS$, where the last one is abbreviated as $RAFA$ in the Tab. 2.

Table 2: Evaluation of RAFA with different linearizing strategies in Byte-based Fault Model, where $k = 2$.

Linearizing Strategies	Runtime (min)	Acceleration Ratio	Standard Deviation (min)
SS	10.58	1.00	3.69
$LPS + SS$	7.85	1.35	1.71
$DPS + SS$	9.75	1.09	3.59
$QPS + SS$	6.36	1.66	**1.43**
$CPS + SS$	7.62	1.39	4.24
$nPS + SS$	8.20	1.29	2.98
$RAFA$	**6.09**	**1.74**	1.63

Table 2 shows that when augmented with supplementary strategies outlined in Sec. 4.4, the resulting linearization is able to facilitate the SAT solving. In particular, when all supplementary strategies are employed, RAFA completes in 6.09 minutes, attaining the highest acceleration ratio (1.74). Therefore, we may conclude that when linearizing ANF, the idea of minimizing the number of auxiliary variables and the length of the output linear equations is beneficial.

5.1.3 Evaluation of Different RAFA Components

In this section, the *byte-based fault model in the 8-th round* is adopted for the experiments.

We claim that the RAFA paradigm is predicated on the concept of incorporating redundant constraints into the system. In Sec. 4.3, two strategies were presented to implement this idea in the context of RAFA. Now, we conduct experiments to illustrate their significance. We set $k = 5$ for the same reason as above.

The redundant constraints we include into the algebraic system are listed in the column "Redundancies". If it is "None", it corresponds to RAFA without redundant constraints; if it is "Inv", only the invertibility is concerned (see Sec. 4.3.3); if it is "FP", only the fault propagation (see Sec. 4.3.1) is appended; if it is "RAFA", it refers to the complete RAFA, which includes both classes of redundant constraints.

Table 3: Evaluation of RAFA with different redundant constraints in Byte-based Fault Model, where $k = 5$.

Redundancies	time (min)	Acceleration Ratio	Standard Deviation (min)
None	168.27	1.00	90.52
Inv	51.43	3.27	14.09
FP	1.89	89.03	0.39
RAFA	**0.59**	**285.20**	**0.22**

As each individual class of redundant constraints results in significant speedup and altogether these redundant constraints allow RAFA to attain the highest acceleration ratio (285.20), we may conclude that the idea of introducing redundant constraints is grossly beneficial to the SAT solvers.

5.2 Evaluation of Different Fault Models

In the following experiments, we will show that one of the advantages of RAFA over DFA is that it responds to different fault assumptions easily and effectively. After using a stronger fault assumption, the efficiency of DFA is unchanged if the algorithm is not adjusted manually. On the other hand, the search space of RAFA will be reduced and the efficiency will be improved automatically. Because there is a known effective DFA on AES-128, we choose to conduct the experiments in this section on AES-128.

During the experiments, we found that if the ANF-to-CNF converter called Bosphorus [CSCM19] is used to convert the S-boxes, the performance may be improved in practice, which may be explained by its discovery implicit linearity and compression of the CNF formula. Bosphorus is an MIT-licensed open-source software, which is aimed at simplifying ANF-to-CNT conversion over $GF(2)$. It employs a variety of techniques, including XL/XSL [CKPS00, CP02], Brickenstein's ANF-to-CNF conversion [BD09] and Gauss-Jordan elimination, etc. As a result, two alternatives for handling with the nonlinear ANF arised in the basic algebraic system are available: Linearization (see Sec. 4.4) and Bosphorus. We will use t^{Lin} to denote the time when Linearization is adopted to handle ANF, and t^{Bos} to denote the runtime when Bosphorus is adopted. The remaining components in RAFA are invariant.

Table 4: Evaluation of RAFA with different options of handling nonlinear ANF, $k \leq 2$.

Fault Model	k	t^{Lin}	$\boldsymbol{t^{Bos}}$
Byte-based	2	5.97 min	**2.35 min**
	1	1.28h	**0.74 h**
	0	16.99 h	**9.68 h**
Nibble-based	2	2.57 min	**0.53 min**
	1	38.83 min	**14.68 min**
	0	101.03 min	**50.92 min**
Bit-based	2	2.34 min	**0.39 min**
	1	17.90 min	**4.46 min**
	0	65.53 min	**27.54 min**

5.2.1 Byte-based Fault Model

First we experiment on the *byte-based fault model in the 8-th round*, where the fault value induced in SR_0 can be anything between 1 and 255. The results are shown in the first three rows of Table 4.

Even though RAFA with Bosphorus completes in 9 hours when $k = 0$, it is still slower than DFA, which takes $4 \sim 5$ hours under the same condition. The major problem, from our point of view, is that the system has too many solutions. According to [TMA11], there are about 2^8 theoretical solutions.

Meanwhile, modern SAT solvers are incapable of efficiently identifying multiple solutions. A common strategy to find multiple solutions is that, when a solution is found by a SAT solver, a new disjunction clause is added to the SAT solver to prohibit the answer from being found again, then the SAT solver is restarted to search for a new solution. It is likely that finding each individual solution takes the same amount of time. Indeed, the difficulty of efficiently finding multiple solutions of a SAT instance is the core concern of another study area called All-SAT [McM02, YSTM14]. All-SAT research, on the other hand, is not as sophisticated as SAT research. For instance, there are no All-SAT benchmarks or contests, and only a few open-source All-SAT solvers which are not frequently used or evaluated are available.

To support our claim that the primary obstacle is locating multiple solutions, we also monitor the time required to locate the first solution. The experiments shown in Fig. 4 are conducted on 100 random instances. In Fig. 4, AVG and SD represent the average runtime and the standard deviation, respectively.

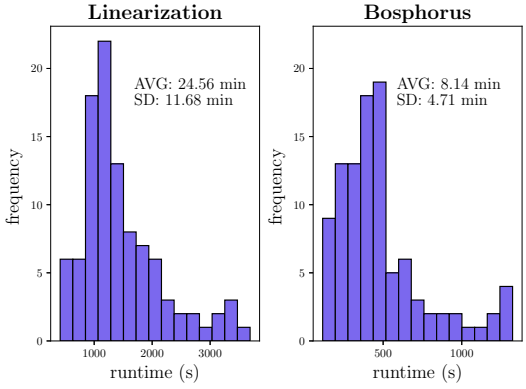

Figure 4: Histogram of the Runtime of Finding the First Solution in Byte-based Fault Model, where $k = 0$.

We can observe that RAFA is able to efficiently find the first candidate key. It indicates that the number of possible solutions to the algebraic system is negatively related to the performance of the RAFA algorithm, and even of general SAT-based algebraic cryptanalysis. To address this issue, we alter the fault model somewhat to reduce the number of solutions to the algebraic system. We shall find that these slightly stronger assumptions assist our RAFA significantly. In contrast, if no extra manual analysis is done to modify DFA algorithm when adopting stronger assumptions, its runtime will be invariant because the stronger fault models make no difference on the first step, which involves relations in the 9-th round, and consequently the second step, which iterates through the output of the first step, will not be affected.

5.2.2 Nibble-based Fault Model

One of the most significant advantage of algebraic cryptanalysis, in particular the RAFA, over DFA is that it is flexible to fit different assumptions of fault model. In this subsection, we experiment on the *nibble-based fault model in the 8-th round* (see Sec. 3.3). Therefore, the algebraic system can be appended by the constraint $\overline{f_4} \wedge \overline{f_5} \wedge \overline{f_6} \wedge \overline{f_7}$, where f_i is the i-th bit of induced fault value f shown in Fig. 2.

Under this fault model, the expected number of solutions drops to 16 according to the probabilistic argument provided in [TMA11], which should allow the SAT solver to recover the master key more efficiently based on previous arguments. Meanwhile, we can observe that this alternation of fault model has no effect on the DFA, since the equations from first step and second step (see Sec. 2.2) stay the same, whose solution requires the same iterations.

The results in the 6-th row of Table 4 show that RAFA with Bosphorus (*resp.* RAFA with Linearization) can break AES-128 in 50.92 minutes (*resp.* 101.03 minutes) under the *nibble-based fault model in the 8-th round*. In comparison, DFA takes $4 \sim 5$ hours under the same condition.

5.2.3 Bit-based Fault Model

If we adopt the *bit-based fault model in the 8-th round*, the expected number of solutions becomes 8, which again expedites the RAFA algorithm while having no effect on the DFA. Since now $f \in \{1, 2, 4, 8, 16, 32, 64, 128\}$, we can add the following at-most-one constraint into the algebraic system:

$$AMO(f) \equiv \bigwedge_{0 \le i < j < 8} (\overline{f_i} \vee \overline{f_j})$$

The results in the last row of Table 4 shows that RAFA with Bosphorus (*resp.* RAFA with Linearization) can break AES-128 in 27.54 minutes (*resp.* 65.53 minutes). The decreases in runtime correspond to the decrease in the number of solutions.

5.2.4 Discussion

Finally, under various scenarios when $k = 0$, Fig. 5 shows the timings of finding the first 9 solutions to the algebraic system given a random instance. The results back up our statement that the worst-case runtime is approximately proportional to the number of solutions. In addition, three observations can be made with Fig. 5:

- RAFA with Bosphorus is more efficient than RAFA with Linearization because the former can locate each individual solution more quickly.
- In RAFA with Bosphorus (the bottom three lines in Fig. 5), the time required to find each individual solution tends to be irrelevant to the fault models.

- In RAFA with Linearization (the top three lines in Fig. 5), the time required to find each individual solution tends to be positively correlated to the number of solutions. We conjecture that this phenomenon is due to the number of auxiliary variables introduced by Linearization. Some of the state bits in deeper rounds may be inferred sooner if the fault value is known to satisfy certain conditions. This variation is amplified because a state bit can be related to many auxiliary variables. For example, if a state bit x is set to be zero, any auxiliary variable associated with a monomial containing x is immediately set to zero via unit propagation. Since the auxiliary variables dominate in the algebraic system of RAFA with Linearization, the performance of SAT solvers may be significantly affected.

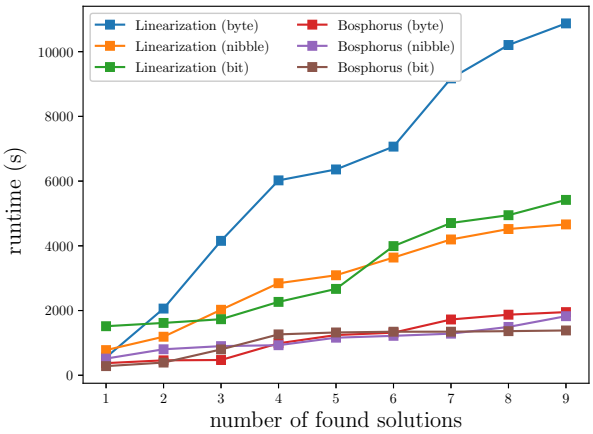

Figure 5: Timings of Finding the First 9 Solutions Given a Random Instance.

5.3 Extensions to Other Block Ciphers

5.3.1 Application to Serpent

Serpent [ABK98] is a SPN block cipher that was designed by Ross Anderson, Eli Biham, and Lars Knudsen as a candidate for the Advanced Encryption Standard (AES). It is a 128-bit block cipher that uses a variable-length key, ranging in size from 128 bits to 256 bits. Serpent consists of linear operations IP, FP, and L, 4-bit S-boxes $S_0, S_1, \cdots, S_7$ and `AddRoundKey`. The encryption can be described by the following equations:

$$\begin{aligned} B_0 &= IP(P) \\ B_{i+1} &= R_i(B_i) \qquad i = 0, 1, \cdots, 31 \\ C &= FP(B_{32}) \end{aligned}$$

where

$$\begin{aligned} R_i(X) &= L(S_{i \bmod 8}(X \oplus K_i)) \qquad i = 0, 1, \cdots, 30 \\ R_i(X) &= S_{i \bmod 8}(X \oplus K_i) \oplus K_{32} \qquad i = 31 \end{aligned}$$

The input key will be padded to 256 bits and separated into eight 32-bit words $w_{-8}, \cdots, w_{-1}$. The prekeys $w_0, \cdots, w_{131}$ are generated by

$$w_i := (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \texttt{0x9e3779b9} \oplus i) <<< 11$$

The round keys are generated by

$$K_i := S_{(35-i) \bmod 8}(w_{4i}||w_{4i+1}||w_{4i+2}||w_{4i+3}) \qquad i = 0, 1, \cdots, 32$$

Since the length of S-boxes is 4 bits, the fault model we use is injecting nibble-faults after the S-box operation at the 28-th round (again, the faults are assumed to be injected to the 0-th nibble, analysis on other situations is replicable). We attempt to recover the 256-bit master key, which is determined by the last two round keys. The following table documents the runtime of RAFA and AFA when three faults are injected. We use k to denote the number of bits of the last round key fed to the SAT solver.

Table 5: RAFA v.s. AFA on Serpent-256 using three faults (an empty cell means failing to solve in 48 hours.)

k	100	80	60	40	30	20	10	0
RAFA	**1.6 s**	**5.3 s**	**18.6 s**	**4.4 m**	**7.6 m**	**16.1 m**	**18.9 min**	**20.7 h**
AFA	34.7 s	50.7 s	6.2 m	1.0 h	—	—	—	—

The results show that RAFA demonstrates a significant speedup over AFA on Serpent-256. Furthermore, with only three faults, RAFA is able to recover the master key of Serpent-256 in a reasonable time, demonstrating real threat to this heavyweight block cipher. In contrast, the only DFA on Serpent [WZD+14] requires 48 faults at 30th and 31st round on average to recover the master key.

5.3.2 Application to SPEEDY

`SPEEDY-r-`6ℓ [LMMR21] is a family of ultra low-latency block ciphers with block and key size 6ℓ, iterating over `r` rounds. In the following experiments, we consider the situation where $\ell = 32$, which is the setting recommended by the authors. SPEEDY consists of linear operations SC and MC, a 6-bit S-box, `AddRoundKey` and `AddRoundConstant`. It is clear that `AddRoundConstant` preserves the input and output fault (indeed, `AddRoundConstant` is a special case of `AddRoundKey`, where the round key is known constant). The round functions are

$$\begin{aligned} R_i(X) &= C_i \oplus (MC \circ SC \circ SB \circ SC \circ SB(K_i \oplus X)) \quad i = 0, 1, \cdots, \mathtt{r} - 2 \\ R_i(X) &= K_{\mathtt{r}} \oplus (SB \circ SC \circ SB(K_{\mathtt{r}-1} \oplus X)) \qquad\qquad\qquad\quad i = \mathtt{r} - 1 \end{aligned}$$

The input key will form the 0-th round key (k_0), the round keys are generated by $k_{i+1} = \mathrm{PB}(k_i)$, where PB is a permutation of the bits.

Since the length of the S-box is 6 bits, the fault model we use is injecting 6-bit faults after the S-box operation at the $(\mathtt{r} - 2)$-th round (again, the faults are assumed to be injected to the first 6 bits, analysis on other situations is replicable). We attempt to recover the 192-bit master key. Note that the exact value of `r` does not matter because each round key is a permutation of the master key. As a result, the constraints related to the key schedule will not slow down the solution significantly, hence RAFA works successfully.

Table 6: RAFA v.s. AFA on `SPEEDY-r-192` (an empty cell means failing to solve in 48 hours.)

k	130	120	110	100	80	60	50	40	30	20	10	0
RAFA	**0.3 s**	**0.2 s**	**0.3 s**	**0.5 s**	**1.0 s**	**5.5 s**	**36.4 s**	**4.0 m**	**13.4 m**	**25.6 m**	**1.2 h**	**1.5 h**
AFA	5.2 m	10.4 m	14.5 m	19.5 m	4.4 h	12.8 h	—	—	—	—	—	—

The results show that RAFA demonstrates a significant speedup over AFA on `SPEEDY-r-192`. Furthermore, with only three faults, RAFA is able to recover the master key of `SPEEDY-r-192` efficiently, demonstrating real threat to this heavyweight block cipher. Since the S-box has small *uniformity* and *linearity* [LMMR21], `SPEEDY` was claimed to provide strong resistance

against differential and linear attacks. Indeed, there is no known effective DFA breaking SPEEDY.

5.3.3 Discussion

In this section, we have successfully broken two different heavyweight SPN ciphers using RAFA. The results imply that RAFA is a generic and efficient variant of AFA.

However, we should point out that RAFA is not able to overcome the problem of AFA related to the number of rounds. For example, we have implemented RAFA on Kuznyechik [Dol16], which is a nine-round 128-bit block cipher with 256-bit key. Even though RAFA turned out to be faster than AFA, at least 200 bits of the last two round keys should be known if the fault is injected to the 8-th round, which implies the algorithm is inefficient on Kuznyechik. The primary obstacle is the complicated key schedule algorithm as shown in Fig. 6, where X represents the Xor layer, S represents the 8-bit S-box, and L represents the linear transformation. Therefore, if we attempt to relate K_7, K_8 with K_9, K_{10} in the algebraic system, we have to include eight rounds of LSX operations, whose cost is similar to the one of injecting the fault in a deep round on AES.

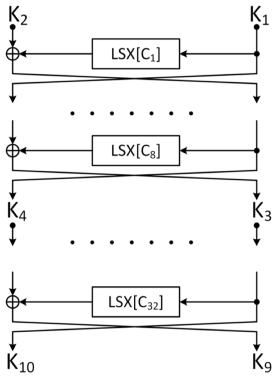

Figure 6: Key Schedule of Kuznyechik

This analysis explains why RAFA is able to break SPEEDY so efficiently: the key schedule of SPEEDY is trivial. Since each round key is a permutation of the input master key, we only need to maintain 192 Boolean variables to represent all round keys, which do not need to be related using additional constraints. Similarly, even though the key schedules of AES and Serpent are not trivial, a round key is related to the next round key through few linear/nonlinear layers.

6 Conclusions and Future Work

This paper proposes the *Redundancies-assisted Algebraic Fault Analysis* (RAFA), which is a novel variant of AFA, with the introduction of generic redundant constraints being the focus. By modifying the representation of the algebraic system, RAFA is able to circumvent the challenges encountered by AFA on heavyweight block ciphers . We implement RAFA on AES-128 under three fault models, one of which was previously studied in the context of DFA. The results demonstrate that RAFA is able to break heavyweight block ciphers such as AES-128. In particular, when restricted to fault models with fewer solutions to the algebraic system, the performance of RAFA will be significantly enhanced. The experiments using stronger fault models show that RAFA's adaptability to additional information is stronger than DFA. In order to prove its generality, we also implement

RAFA on Serpent-256 and SPEEDY. The results demonstrate that RAFA is significantly more efficient than AFA on different SPN block ciphers and is able to break heavyweight ciphers using a few fault injections.

Future work can be derived in three different directions. Firstly, even though RAFA should work on most SPN block ciphers because its redundant components do not rely on the specific arrangements and designs of the SPN networks, it is not efficient enough to deal with the situations when the number of operation rounds is too large. For example, RAFA fails on Kuznyechik [Dol16] because the relations between its round keys are complicated, as shown in Fig. 6. It will be interesting to further enhance RAFA to handle these situations. Secondly, as discussed in Sec. 5, we find that supplementary linearizing strategies benefit the SAT solvers. Thus, we can discover further linearizing strategies other than the ones presented in Sec. 4.4. Finally, we have shown that a main impediment of SAT-based algebraic cryptanalysis is that modern SAT solvers lack the ability to locate all solutions efficiently. Thus, building an efficient All-SAT solver may potentially increase the performance of RAFA.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2020AAA0107700), by National Natural Science Foundation of China (62227805, 62072398), by SUTD-ZJU IDEA Grant for visiting professors (SUTD-ZJUVP201901), by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by National Key Laboratory of Science and Technology on Information System Security (6142111210301), by State Key Laboratory of Mathematical Engineering and Advanced Computing, and by Key Laboratory of Cyberspace Situation Awareness of Henan Province (HNTS2022001).

Appendix

A First-Step of DFA

$$\begin{cases} 2F_0 = S^{-1}\left(x_0 \oplus K_0^{(10)}\right) \oplus S^{-1}\left(x_0^* \oplus K_0^{(10)}\right) \\ F_0 = S^{-1}\left(x_{13} \oplus K_{13}^{(10)}\right) \oplus S^{-1}\left(x_{13}^* \oplus K_{13}^{(10)}\right) \\ F_0 = S^{-1}\left(x_{10} \oplus K_{10}^{(10)}\right) \oplus S^{-1}\left(x_{10}^* \oplus K_{10}^{(10)}\right) \\ 3F_0 = S^{-1}\left(x_7 \oplus K_7^{(10)}\right) \oplus S^{-1}\left(x_7^* \oplus K_7^{(10)}\right) \end{cases} \quad \begin{cases} 3F_1 = S^{-1}\left(x_4 \oplus K_3^{(10)}\right) \oplus S^{-1}\left(x_4^* \oplus K_4^{(10)}\right) \\ 2F_1 = S^{-1}\left(x_1 \oplus K_1^{(10)}\right) \oplus S^{-1}\left(x_1^* \oplus K_1^{(10)}\right) \\ F_1 = S^{-1}\left(x_{14} \oplus K_{14}^{(10)}\right) \oplus S^{-1}\left(x_{14}^* \oplus K_{14}^{(10)}\right) \\ F_1 = S^{-1}\left(x_{11} \oplus K_{11}^{(10)}\right) \oplus S^{-1}\left(x_{11}^* \oplus K_{11}^{(10)}\right) \end{cases}$$

$$\begin{cases} F_2 = S^{-1}\left(x_8 \oplus K_8^{(10)}\right) \oplus S^{-1}\left(x_8^* \oplus K_8^{(10)}\right) \\ 3F_2 = S^{-1}\left(x_5 \oplus K_5^{(10)}\right) \oplus S^{-1}\left(x_5^* \oplus K_5^{(10)}\right) \\ 2F_2 = S^{-1}\left(x_2 \oplus K_2^{(10)}\right) \oplus S^{-1}\left(x_2^* \oplus K_2^{(10)}\right) \\ F_2 = S^{-1}\left(x_{15} \oplus K_{15}^{(10)}\right) \oplus S^{-1}\left(x_{15}^* \oplus K_{15}^{(10)}\right) \end{cases} \quad \begin{cases} F_3 = S^{-1}\left(x_{12} \oplus K_{12}^{(10)}\right) \oplus S^{-1}\left(x_{12}^* \oplus K_{12}^{(10)}\right) \\ F_3 = S^{-1}\left(x_9 \oplus K_9^{(10)}\right) \oplus S^{-1}\left(x_9^* \oplus K_9^{(10)}\right) \\ 3F_3 = S^{-1}\left(x_6 \oplus K_6^{(10)}\right) \oplus S^{-1}\left(x_6^* \oplus K_6^{(10)}\right) \\ 2F_3 = S^{-1}\left(x_3 \oplus K_4^{(10)}\right) \oplus S^{-1}\left(x_3^* \oplus K_3^{(10)}\right) \end{cases}$$

B Redundant Constraints Given by Labelled Boxes

$$f = SR_{8,0} \oplus SR_{8,0}^*, \quad 2f = MC_{8,0} \oplus MC_{8,0}^*, \quad f = MC_{8,1} \oplus MC_{8,1}^*,$$

$$f = MC_{8,2} \oplus MC_{8,2}^*, \quad 3f = MC_{8,3} \oplus MC_{8,3}^* \quad 2f = AK_{8,0} \oplus AK_{8,0}^*,$$

$$f = AK_{8,1} \oplus AK_{8,1}^*, \quad f = AK_{8,2} \oplus AK_{8,2}^*, \quad 3f = AK_{8,3} \oplus AK_{8,3}^*$$

$$F_0 = SB_{9,0} \oplus SB_{9,0}^* \quad F_1 = SB_{9,1} \oplus SB_{9,1}^* \quad F_2 = SB_{9,2} \oplus SR_{9,2}^*$$

$$F_3 = SB_{9,3} \oplus SR_{9,3}^* \quad F_0 = SR_{9,0} \oplus SR_{9,0}^* \quad F_1 = SR_{9,13} \oplus SR_{9,13}^*$$

$$F_2 = SR_{9,10} \oplus SR_{9,10}^* \quad F_3 = SR_{9,7} \oplus SR_{9,7}^* \quad 2F_0 = MC_{9,0} \oplus MC_{9,0}^*$$

$$F_0 = MC_{9,1} \oplus MC_{9,1}^* \quad F_0 = MC_{9,2} \oplus MC_{9,2}^* \quad 3F_0 = MC_{9,3} \oplus MC_{9,3}^*$$

$$F_3 = MC_{9,4} \oplus MC_{9,4}^* \quad F_3 = MC_{9,5} \oplus MC_{9,5}^* \quad 3F_3 = MC_{9,6} \oplus MC_{9,6}^*$$

$$2F_3 = MC_{9,7} \oplus MC_{9,7}^* \quad F_2 = MC_{9,8} \oplus MC_{9,8}^* \quad 3F_2 = MC_{9,9} \oplus MC_{9,9}^*$$

$$2F_2 = MC_{9,10} \oplus MC_{9,10}^* \quad F_2 = MC_{9,11} \oplus MC_{9,11}^* \quad 3F_1 = MC_{9,12} \oplus MC_{9,12}^*$$

$$2F_1 = MC_{9,13} \oplus MC_{9,13}^* \quad F_1 = MC_{9,14} \oplus MC_{9,14}^* \quad F_1 = MC_{9,15} \oplus MC_{9,15}^*$$

References

[ABK98] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard. *NIST AES Proposal*, 174:1–23, 1998.

[AMT13] Sk Subidh Ali, Debdeep Mukhopadhyay, and Michael Tunstall. Differential fault analysis of AES: towards reaching its limits. *Journal of Cryptographic Engineering*, 3(2):73–97, 2013.

[BCJ07] Gregory V Bard, Nicolas T Courtois, and Chris Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. *Cryptology ePrint Archive*, 2007.

[BD09] Michael Brickenstein and Alexander Dreyer. Polybori: A framework for Gröbner-basis computations with boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326–1345, 2009.

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic search of attacks on round-reduced aes and applications. In *Annual Cryptology Conference*, pages 169–187. Springer, 2011.

[BDL97] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques*, pages 37–51. Springer, 1997.

[BECN+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[BHN14] Pedro Barahona, Steffen Hölldobler, and Van-Hau Nguyen. Efficient SAT-encoding of linear CSP constraints. In *ISAIM*, 2014.

[BHZ06] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys (CSUR)*, 38(4):12–es, 2006.

[Bjö11] Magnus Björk. Successful SAT encoding techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):189–201, 2011.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[BS03] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In *International Conference on Financial Cryptography*, pages 162–181. Springer, 2003.

[BS12] Eli Biham and Adi Shamir. *Differential cryptanalysis of the data encryption standard*. Springer Science & Business Media, 2012.

[CB07] Nicolas T Courtois and Gregory V Bard. Algebraic cryptanalysis of the data encryption standard. In *IMA International Conference on Cryptography and Coding*, pages 152–169. Springer, 2007.

[CFGR12] Claude Carlet, Jean-Charles Faugere, Christopher Goyet, and Guénaël Renault. Analysis of the algebraic side channel attack. *Journal of Cryptographic Engineering*, 2(1):45–62, 2012.

[CJW10] Nicolas T Courtois, Keith Jackson, and David Ware. Fault-algebraic attacks on inner rounds of DES. In *E-Smart'10 Proceedings: The Future of Digital Security Technologies*. Strategies Telecom and Multimedia, 2010.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 392–407. Springer, 2000.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[CP02] Nicolas T Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *International conference on the theory and application of cryptology and information security*, pages 267–287. Springer, 2002.

[CSCM19] Davin Choo, Mate Soos, Kian Ming A Chai, and Kuldeep S Meel. Bosphorus: Bridging ANF and CNF solvers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 468–473. IEEE, 2019.

[DLV03] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on AES. In *International Conference on Applied Cryptography and Network Security*, pages 293–306. Springer, 2003.

[Dol16] Vasily Dolmatov. Gost r 34.12-2015: Block cipher" kuznyechik". Technical report, 2016.

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[Gav07] Marco Gavanelli. The log-support encoding of CSP into SAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 815–822. Springer, 2007.

[Gir04] Christophe Giraud. DFA on AES. In *International Conference on Advanced Encryption Standard*, pages 27–41. Springer, 2004.

[Hua08] Jinbo Huang. Universal booleanization of constraint models. In *International Conference on Principles and Practice of Constraint Programming*, pages 144–158. Springer, 2008.

[IM94] Kazuo Iwama and Shuichi Miyazaki. SAT-variable complexity of hard combinatorial problems. In *In Proceedings of the World Computer Congress of the IFIP*. Citeseer, 1994.

[JK10] Philipp Jovanovic and Martin Kreuzer. Algebraic attacks using SAT-solvers. 2010.

[JKP12] Philipp Jovanovic, Martin Kreuzer, and Ilia Polian. An algebraic fault attack on the LED block cipher. *Cryptology ePrint Archive*, 2012.

[LJH+07] Fen Liu, Wen Ji, Lei Hu, Jintai Ding, Shuwang Lv, Andrei Pyshkin, and Ralf-Philipp Weinmann. Analysis of the SMS4 block cipher. In *Australasian Conference on Information Security and Privacy*, pages 158–170. Springer, 2007.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The speedy family of block ciphers-engineering an ultra low-latency cipher from gate level for secure processor architectures. *Cryptology ePrint Archive*, 2021.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[MBB11] Mohamed Saied Emam Mohamed, Stanislav Bulygin, and Johannes Buchmann. Using SAT solving to improve differential fault analysis of Trivium. In *International Conference on Information Security and Assurance*, pages 62–71. Springer, 2011.

[McM02] Ken L McMillan. Applying SAT methods in unbounded symbolic model checking. In *International Conference on Computer Aided Verification*, pages 250–264. Springer, 2002.

[MS11] Norbert Manthey and Peter Steinke. Quadratic direct encoding vs. linear order encoding. In *First International Workshop on the Cross-Fertilization Between CSP and SAT (CSPSAT'11)*. Citeseer, 2011.

[MSP+17] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, January 2017.

[Muk09] Debdeep Mukhopadhyay. An improved fault based attack of the advanced encryption standard. In *International Conference on Cryptology in Africa*, pages 421–434. Springer, 2009.

[Pet15] Justyna Petke. *Bridging Constraint Satisfaction and Boolean Satisfiability*. Springer, 2015.

[SAT] SAT competition. http://www.satcompetition.org/.

[SBT17] Takehide Soh, Mutsunori Banbara, and Naoyuki Tamura. Proposal and evaluation of hybrid encoding of CSP to SAT integrating order and log encodings. *International Journal on Artificial Intelligence Tools*, 26(01):1760005, 2017.

[SE05] Niklas Sorensson and Niklas Een. Minisat v1. 13-a SAT solver with conflict-clause minimization. *SAT*, 2005(53):1–2, 2005.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer, 2009.

[SP09] Bernd Steinbach and Christian Posthoff. *Logic functions and equations: examples and exercises*. Springer Science & Business Media, 2009.

[The20] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.0)*, 2020. `https://www.sagemath.org`.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices*, pages 224–233. Springer, 2011.

[TTKB09] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[vWBV+17] Jasper van Woudenberg, Cees-Bart Breunesse, Rajesh Velegalati, Panasayya Yalla, and Sergio Gonzalez. Differential fault analysis using symbolic execution. In *Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop*, pages 1–9, 2017.

[Wal00] Toby Walsh. SAT v CSP. In *International Conference on Principles and Practice of Constraint Programming*, pages 441–456. Springer, 2000.

[WZD+14] Li Wei, Tao Zhi, Gu Dawu, Sun Li, Qu Bo, Liu Zhiqiang, and Liu Ya. An effective differential fault analysis on the serpent cryptosystem in the internet of things. *China Communications*, 11(6):129–139, 2014.

[YSTM14] Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. All-SAT using minimal blocking clauses. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 86–91. IEEE, 2014.

[ZGZ+12] Xinjie Zhao, Shize Guo, Fan Zhang, Tao Wang, Zhijie Shi, and Keke Ji. Algebraic differential fault attacks on LED using a single fault injection. *Cryptology ePrint Archive*, 2012.

[ZGZ+16] Fan Zhang, Shize Guo, Xinjie Zhao, Tao Wang, Jian Yang, Francois-Xavier Standaert, and Dawu Gu. A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers. *IEEE Transactions on Information Forensics and Security*, 11(5):1039–1054, 2016.