TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 21/24 pp576–588
DOI: 10.26599/TST.2023.9010035
Volume 29, Number 2, April 2024

Fault Analysis on AES: A Property-Based Verification Perspective

Xiaojie Dai, Xingxin Wang, Xue Qu, Baolei Mao, and Wei Hu*

Abstract: Fault analysis is a frequently used side-channel attack for cryptanalysis. However, existing fault attack methods usually involve complex fault fusion analysis or computation-intensive statistical analysis of massive fault traces. In this work, we take a property-based formal verification approach to fault analysis. We derive fine-grained formal models for automatic fault propagation and fusion, which establish a mathematical foundation for precise measurement and formal reasoning of fault effects. We extract the correlations in fault effects in order to create properties for fault verification. We further propose a method for key recovery, by formally checking when the extracted properties can be satisfied with partial keys as the search variables. Experimental results using both unprotected and masked advanced encryption standard (AES) implementations show that our method has a key search complexity of 2^{16}, which only requires two correct and faulty ciphertext pairs to determine the secret key, and does not assume knowledge about fault location or pattern.

Key words: side-channel attack; fault analysis; fault propagation model; property extraction; fault verification

1 Introduction

Side channel analysis (SCA) has become the most practical cryptanalysis technique since Kocher's pioneering work in timing attack[1] and over two decades' research efforts in other SCA methods such as power, electronic magnetic (EM), thermal, and fault analysis[2]. Among various SCA techniques, fault attack takes an active approach, which injects faults into the execution process of cryptographic implementations rather than merely collecting and analyzing side channel traces. Carefully crafted fault attacks can be far more efficient than passive SCA methods, requiring only a small number of fault injections to recover the entire key. In addition, typical SCA countermeasures, such as randomization, blinding, and masking, which can be effective for mitigating timing, power, as well as EM side channels, may not provide sufficient protection against active SCA through fault injection[3].

However, existing fault attack methods usually need to precisely inject desired patterns of faults at specific locations or rely on knowledge of faults occurred. Such assumptions are hard to satisfy in physical attacks considering the randomness in fault behaviors. Thus, these methods usually require a large number of fault injections in order to filter out enough useful fault traces for key recovery. In addition, most methods need to perform mathematically complex fault fusion analysis, computation-intensive statistical analysis or massive trace processing to create a template or model for key selection. Little research work has taken fault related properties as a design constraint beyond functional correctness. Especially, there is a lack of precise model for automated fault fusion and formal fault verification.

Motivated by the recent advances in property driven security solutions[4, 5], we propose to perform fault analysis through property-based formal verification in this work. We derive a formal model for precise measurement of fault effects, which provides the

• Xiaojie Dai, Xingxin Wang, Xue Qu, and Wei Hu are with School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China. E-mail: {daixiaojie, weihu}@nwpu.edu.cn; {w2x, quxue}@mail.nwpu.edu.cn.
• Baolei Mao is with School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450001, China. E-mail: maobaolei@zzu.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2023-03-07; revised: 2023-04-28; accepted: 2023-05-03

mathematical foundation for formal fault verification. We automatically extract fault correlation related properties for fault verification and develop a fault attack method by formally checking the extracted properties to search for the correct key. Specifically, this work makes the following contributions:

• Deriving a formal model for precise fault propagation and fault effect measurement;

• Proposing a fault attack method through fault correlation related property extraction and formal verification;

• Presenting experimental results using both unprotected and masked cryptographic implementations to demonstrate the effectiveness of the proposed method.

The remainder of this paper is organized as follows. In Section 2, we summarize different fault analysis methods. A brief overview of attack and fault models is provided in Section 3. Section 4 provides our formal model that precisely measures fault effects. We discuss fault related property extraction in Section 5 and describe our fault analysis method through formal verification in Section 6. We present experimental results to demonstrate the effectiveness of the proposed method in Section 7 and conclude the paper in Section 8.

2 Related Work

The concept of fault analysis was first introduced by Boneh et al.[6] using the RSA-CRT crypto-system as a demonstration. Over the last two decades, researchers have developed a variety of fault attack methods, including differential fault analysis (DFA)[7], algebraic fault analysis (AFA)[8], ineffective fault analysis (IFA)[9], fault rate analysis (FRA)[10], statistical fault analysis (SFA)[11], statistical ineffective fault analysis (SIFA)[12], and persistent fault analysis (PFA)[13], as summarized in Table 1. Among these, DFA, AFA, and IFA precisely model the propagation of fault effects and leverage the characteristics in fault behaviors to search for the correct key. The remaining methods measure how faults change the distribution of intermediate results or the ciphertext and perform statistical analysis to determine the correct key.

DFA[7] was proposed by Biham and Shamir in 1997. It takes correct and faulty ciphertext pairs as inputs, which makes guesses about the key, performs differential cryptanalysis to determine the fault status of intermediate encryption results, and leverages the characteristics in fault patterns to distinguish the correct key guess from incorrect ones. It has been shown to be effective in attacking the data encryption standard (DES)[7], PRESENT[14], and advanced encryption standard (AES)[15–19] implementations. Momeni et al.[15] performed a single-byte fault injection attack on an field program gate array (FPGA) implementation of AES, and filtered out the key using differential S-Box tables. Ali and Mukhopadhyay[16] successfully implemented DFA on AES-128 key expansion with a single fault injection. Other researchers[17–19] leveraged the correlation between different faulty locations to obtain multiple key bytes per analysis. Wang et al.[17] successfully attacked AES for all key lengths using no more than four correct and faulty ciphertext pairs through correlation fault analysis. Similarly, an attacker can exploit the fault correlation in key expansion to conduct fault analysis to search for the key[18, 19], which can be slightly more complex than attacking the round function. Although DFA is simple yet effective, it requires precise fault injection at specific locations and complex mathematical derivation.

Table 1 Summary of fault attacks.

Method	Description	Technical approach
DFA	Analyze the difference between the correct and faulty ciphertexts to recover the key	Mathematical analysis
AFA	Combine differential fault attack and an algebraic technique to search for the key	Algebraic or Boolean analysis
IFA	Leverage faults that have no effect on ciphertext	Mathematical analysis
FRA	Use clockwise collision to retrieve the key	Statistical analysis
SFA	Exploits statistics of faulty ciphertexts	Statistical analysis
SIFA	Combine statistical fault analysis and ineffective fault analysis	Statistical analysis
PFA	Inject persistent faults based on statistical analysis for key recovery	Statistical analysis

AFA[8] was proposed by Courtois et al. in 2010 to attack DES. They showed that obtaining 24 key bits using a single faulty ciphertext with two bits flipped in round 13 only takes 0.01 h. AFA combines the benefits of mathematical analysis and conventional fault attack, which establishes a fault model and derives the algebraic formulation of the cryptographic algorithm to check for the correct key. Such mathematical derivation can be complex and challenging. Zhang et al.[20] established a framework for evaluating AFA from three levels on lightweight block ciphers. They implemented fault injection attacks against Lblock, targeting both

encryption transformation and key extension. Gay et al.[21] created a framework for automated construction of fault attacks on hardware implementation of block ciphers using AFA, which supports multiple fault injections.

Early fault attacks often require one or several pairs of correct and faulty ciphertexts or different faulty ciphertexts corresponding to the same plaintext, which is somewhat demanding in realistic attack scenarios. As an alternative approach, an adversary can realize ciphertext-only attacks using faulty ciphertexts from different plaintexts through SFA[11]. SFA first establishes a fault model, and then uses statistical analysis to find key values that match the characteristics of the model, which can be the key candidates. Both SIFA and PFA fall into the category of SFA methods.

IFA[9] was proposed to account for ineffective fault injections, where faults injected do not cause any observable effects, e.g., on the ciphertext. In cryptograph hardware and embedded systems (CHES) 2018, statistical ineffective fault analysis[12] was proposed to combine IFA and SFA, which exploited statistical characteristics of both faulty and correct (i.e., when a fault is ineffective) ciphertexts to recover the key. Saha et al.[22] proposed a deep learning-based SIFA method, which demonstrated the possibility of employing recent machine learning and artificial intellegence techniques for fault analysis.

Fault attack models usually assume random faults, which mean that an adversary needs to repeatedly inject faults in order to filter out enough desired fault patterns. In CHES 2018, PFA was proposed by Zhang et al.[13] to take advantage of persistent faults, which exist until power off once a fault is successfully injected. This method makes it possible to bypass dual modular redundancy protection, but requires a large amount of faulty ciphertexts and complex mathematical analysis. In a successive work, the authors further improve the analysis method so that PFA can support attacks on AES and PRESENT with or without knowing the details of the S-Box design[23]. Xu et al.[24] proposed an improved PFA that allows faults to propagate through two levels of faulty S-Boxes before reaching the ciphertext output while the original PFA only supports one level. The benefit is that the distribution of the faulty ciphertexts will become more biased, making it possible to recover the key with fewer ciphertexts.

Fault analysis is a very effective method against unprotected cryptographic implementations. In addition, it has been shown that cryptographic designs with mask protection can also be vulnerable to fault attack. Although Bae et al.[25] have proved that the AES design using Akkar and Giraud's mask can be secure against fault attack, it is generally accepted that masking does not provide sufficient protection against fault attack[3, 26]. FRA has been proposed to break masked AES by Wang et al.[10] They inject clock glitches into the critical path of S-Box so that the mask fails to protect the byte substitution, which would make it easier to attack the protected design. Dobraunig et al.[27] have demonstrated that SIFA can also break masked AES.

Although there are various fault analysis methods, they typically require collecting a large number of faulty ciphertexts, perform complex algebraic or statistical analysis to determine the correct key. This work takes a formal approach to fault analysis. It automatically extracts fault related properties using a small amount of faulty ciphertexts. It employs a formal solver to verify the extracted property and automatically search for the correct key that satisfies the property.

3 Attack and Fault Model

We assume that an attacker has full knowledge of the cryptographic algorithm as well as the implementation details. He/she can send plaintexts to the cryptographic core and observe the ciphertext outputs. We also assume that the attacker can inject random faults into desired encryption rounds. He/she can collect correct and faulty ciphertext pairs or faulty ciphertexts under different fault patterns corresponding to the same plaintext for fault analysis to recover the secret key.

We adopt the fault model frequently employed in DFA on AES [17]. When faults are injected between the MixColumns operations of rounds N_r-2 and N_r-1, we consider the cases where at most one single-byte random fault occurs in each column of the state matrix. There can be multiple (at most four) faulty bytes distributed across different columns of the state matrix. When faults are injected between the MixColumns operations of rounds N_r-3 and N_r-2, only a single-byte random fault is allowed. In these two cases, the linear correlation in fault effects would be significant by column at the input of round N_r. In other cases, there are either a large number of possible combinations in fault locations or the linear correlation in fault effects would become non-significant.

4 Modeling Fault Propagation Effect

In this section, we first derive a formal model for precise fault propagation and then demonstrate how the model can be used for measuring fault effects.

4.1 Formal model for precise fault propagation

Fault propagation aims to understand which design components and regions will be influenced by a fault source. Existing methods usually change the value at the fault source while keeping other inputs untouched, and observe which design nets will flip consequently. These methods require testing two design instances and performing an XOR of the observed values to determine the fault status, which are ineffective for testing internal design points. As an alternative solution, one can mark the fault source as 'X' and perform X-propagation to determine the fault status of different design components. Such methods have the problems of X-optimism and X-pessimism. Few research work has investigated dedicated models for precise fault propagation.

In order to create a formal model for understanding the propagation of fault effects, we associate each signal bit in a hardware design with a fine-grained fault label. Specifically, for a signal bit S, we add a fault label bit S_e to represent its fault status. Specifically, $S_e = 1$ when S is in a faulty state; otherwise, $S_e = 0$. For example, the fault label at the fault source should be logical 1. A fault model calculates the fault labels of intermediate signals and primary outputs upon the input values as well as their fault labels. It can be defined as the following Boolean function:

$$O_e = f_m(I^0, I^1, \ldots, I^n, I_e^0, I_e^1, \ldots, I_e^n) \quad (1)$$

where $I^0, I^1, \ldots, I^n$ are the original inputs and $I_e^0, I_e^1, \ldots, I_e^n$ are their fault labels, respectively.

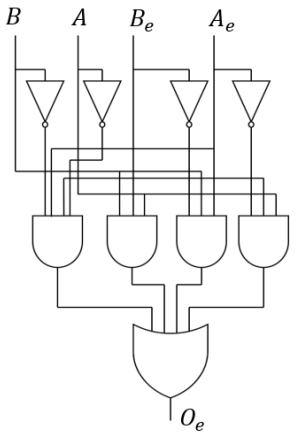
Consider the two-input AND (AND-2) gate. Let the logic function of AND-2 be $O = A \,\&\, B$. Given the original inputs A and B as well as their fault labels A_e and B_e, the fault propagation model calculates the fault label of O.

Figure 1a shows a partial truth table for deriving the formal fault propagation model of AND-2. We eliminate the entries where both A and B are fault free since the output will certainly be fault free in these cases.

Take the third row as an example, where $A = 1, B = 0, A_e = 0$, and $B_e = 1$. This indicates that A is fault free while B is faulty, i.e., B should actually take the value of logical 1 and thus the output O would flip to 1 as well. In other words, the fault in B propagates to the output. Now consider the fifth row, where $A = 0, B = 0, A_e = 1$, and $B_e = 0$. In this case, the output will stay logical 0 because the fault free 0 in input B prevents the fault in A from propagation.

#	A	B	A_e	B_e	O	O_e
1	0	0	0	1	0	0
2	0	1	0	1	0	0
3	1	0	0	1	0	1
4	1	1	0	1	1	1
5	0	0	1	0	0	0
6	0	1	1	0	0	1
7	1	0	1	0	0	0
8	1	1	1	0	1	1
9	0	0	1	1	0	1
10	0	1	1	1	0	0
11	1	0	1	1	0	0
12	1	1	1	1	1	1

(a)



(b)

Fig. 1 Deriving fault propagation model for AND-2. (a) Partial truth table for calculating the fault label of AND-2 output, (b) formal fault propagation model for AND-2.

From the full truth table for calculating the output fault label, we can derive the following formal fault propagation model for AND-2.

$$O_e = AB_e\overline{A_e} + BA_e\overline{B_e} + \overline{A}\,\overline{B}A_eB_e + AB_eB_e \quad (2)$$

Since the two-input OR gate (OR-2) is the dual of AND-2, it is easy to derive its fault propagation model using the DeMorgan's Law by simply inverting the original inputs. One can verify the correctness of the model by enumerating a truth table similar to the one shown in Fig. 1b.

$$O_e = \overline{A}B_e\overline{A_e} + \overline{B}A_e\overline{B_e} + ABA_eB_e + \overline{A}\,\overline{B}B_e \quad (3)$$

The fault propagation model for the inverter is straightforward to specify since any faults in the input would cause the output to be faulty.

$$O_e = I_e \quad (4)$$

Similarly, we can create the formal fault propagation models for other Boolean gates. From the above examples, it is not necessarily any faults in inputs would cause the output to be faulty; certain input conditions will prevent fault from propagation. Our method can precisely model fault propagation behaviors.

4.2 Measuring fault propagation effects

We use AES as an example to illustrate how the formal model developed in Section 4.1 can be used to measure fault propagation effects.

The first step is to create the fault propagation model for the entire AES design. With the fault propagation

models for primitive gates, we can compose more complex models in a constructive manner. Specifically, we can synthesize the AES design into gate netlist and then instantiate the corresponding fault propagation model for each primitive gate in the netlist. In this way, we can generate fault propagation models for large circuit designs in polynomial time.

Figure 2 demonstrates how our formal model can measure fault propagation effects in one AES round. As shown in Fig. 2a, our model associates an addition fault label for each signal to indicate the fault status of that signal, e.g., sbox_e represents the fault label of the S-Box output. Note that each signal bit has its own label bit in order to differentiate the fault status across different signal bits. We use fault propagation models denoted as FM(·) to measure fault effects. The models calculate the outputs as well as their fault status upon the original inputs and their fault labels.

From Fig. 2b, a word in the round input is 32'hEA835CF0 and its fault label is 32'h01000000, indicating that the lowest bit of the first byte is faulty. After S-Box, the fault status of the corresponding byte becomes 8'h6E. The ShiftRows transform does not affect the fault status. After MixColumns, the fault in that byte spreads into a column and the fault status becomes 32'hDC6E6EB2.

Our fault propagation model can be used to calculate the fault labels of arbitrary intermediate signals and outputs. This automates fault fusion analysis and allows retrieving the fault status of any internal signal at runtime. Since our model can be described using Boolean function like those formalized in Eqs. (2) to (4), we can use standard hardware description language (HDL) to describe the model and perform fault simulation under standard EDA tools such as ModelSim. Figure 2c shows the fault simulation results, which are identical with theoretic analysis.

5 Fault Related Property Extraction

In order to perform formal fault verification, we need high quality fault related properties since verification is usually a property driven process. We define that a fault related property is an invariant fault related design behavior over a set of fault traces. For a better understanding, consider the invertor, where a fault in the input always propagates to the output. We can formalize this invariant design behavior as the following property.

$$\text{assert} \quad O_e = I_e \tag{5}$$

Existing formal verification solutions usually require manual specification of properties. This puts heavy burdens on verification engineers while providing no guarantee on the quality or completeness of the properties generated. We aim to automate the fault related property generation process. In the following subsections, we describe our property extraction and checking procedures, respectively.

5.1 Property extraction

We consider two types of fault related properties, namely fault propagation property and fault correlation property.

A fault propagation property describes the relationship in fault status between a fault source and an observation point. It specifies if the fault in a source node could propagate to a specific destination. Formula (6) shows two example properties regarding if a fault in the AES key could propagate (the $\rightarrow$ operator) or not (the $\not\rightarrow$ operator) to two primary outputs.

$$\begin{aligned} &\text{assert} \quad aes\_key \rightarrow ciphertext, \\ &\text{assert} \quad aes\_key \not\rightarrow cipher\_ready \end{aligned} \tag{6}$$

Given the source and destination points, fault propagation properties can be automatically generated. In addition, such properties can be mapped to standard assertion languages and verified using our formal fault propagation model. Formula (7) shows

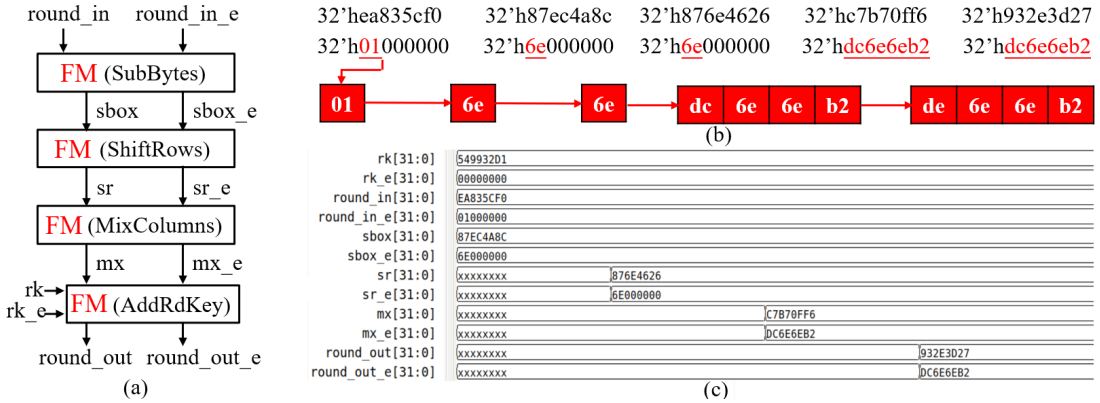


Fig. 2 Fault fusion in one AES round using our fault propagation model.

the translated assertion properties, where aes_key_e, $ciphertext_e$, and $cipher\_ready_e$ are the fault labels of the corresponding signals.

$$\begin{aligned} &\text{assume} \quad aes\_key_e = 1, \\ &\text{assert} \quad ciphertext_e = 1, \\ &\text{assert} \quad cipher\_ready_e = 1 \end{aligned} \tag{7}$$

A fault correlation property describes the correlation in fault status among multiple fault locations. Oftentimes, such correlation is implicit, making this type of property harder to extract. We follow the definition of fault related properties and extract them from fault traces. Figure 3 shows our property mining process.

Given a cryptographic hardware design, we use the constructive method introduced in Section 4 to generate its formal fault propagation model. Since the model can be described using standard HDL, we are able to perform random simulation on the model to collect a set of fault traces for property extraction.

An important step in the property extraction process is to mine invariant properties from the fault simulation traces. We consider a single fault source each time and perform random simulation to identify the intermediate nodes that are influenced by the designated fault source. The fault status at different locations should be correlated due to the identical fault origin. We specify the corresponding fault status as truth tables and recover the correlation through regression analysis or logic minimization.

Consider the AES round shown in Fig. 2, a fault in the S-Box input would cause four bytes to be faulty at the round output. These four bytes are considered to be correlated in fault status. We denote these four bytes as $rdo0$, $rdo1$, $rdo2$, and $rdo3$, respectively. Take $rdo0$ and $rdo1$ as an example. The following programmable logic array (PLA) describes their corresponding fault status, where $rdo0_e$ and $rdo1_e$ are the fault labels of $rdo0$ and $rdo1$, respectively.

```
.module  p01
.i   rdo0-e
.o   rdo1-e
.pla
00000000   00000000
00000001   00000010
...
11011100   01101110
...
```

We then use logic minimizers such as *Espresso* to perform logic simplification and retrieve the fault correlation. However, the fault correlation extracted usually depends on the number of fault traces collected. As we increase the number of fault traces, the fault correlation gradually converges to the most accurate one.

As an example, when using 128 fault traces, the fault correlation recovered from the above PLA is shown in Eq. (8).

$$\text{assert} \quad rdo1_e == \{rdo0_e[6:0], 0\} \tag{8}$$

When increasing the number of fault traces to 256, the fault correlation recovered is shown in Eq. (9), which is more accurate than Eq. (8). Here, $\oplus$ is the exclusive OR operator.

$$\begin{aligned} \text{assert} \quad & rdo1_e == \{rdo0_e[6:0], 0\} \oplus \{3'b0, \\ & rdo0_e[7], rdo0_e[7], 0, rdo0_e[7], 0\} \end{aligned} \tag{9}$$

In order to check if the generated property is complete in covering all possible fault patterns, we further perform a property checking and refining step to increase the coverage of incomplete properties.

5.2 Property checking and refining

For properties that only cover a small or moderate scale state space, we can perform exhaustive testing and enumerate full truth tables for property generation. For properties that involve a huge state space, we can only perform limited test and obtain a partial truth table. In such cases, we employ regression analysis or do not care based logic minimization techniques for deriving higher-quality properties. This would also lead to more expressive properties. In addition, we need to check if the properties generated could accurately cover the fault behaviors of the design.

Consider the fault traces collected for the AES round output bytes $rdo1$ and $rdo2$. The following shows the PLA that describes the corresponding fault status of these two signals over 32 traces.

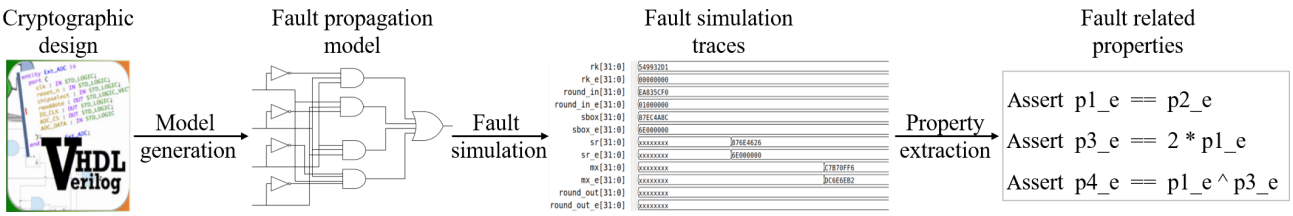


Fig. 3 Fault related property mining process.

```
.module  p12
.i   rdo1-e
.o  rdo2-e
.pla
00000000   00000000
00000001   00000001
...
00011111   00011111
...
```

Linear regression analysis yields the property shown in Eq. (10).

$$\text{assert} \quad rdo2_e = rdo1_e \tag{10}$$

Since this property is extracted from a limited number of fault traces, we need to check if the property holds under all possible fault events. This can be formally verified under our formal fault propagation model. In order to use a Boolean Satisfiability analysis tool to prove the property, we create a tag signal, where

$$tag = rdo2_e \oplus rdo1_e \tag{11}$$

We then employ the Yosys verification tool to prove that tag is always 0. Figure 4 shows the formal proof result. It indicates that the extracted property is complete to cover all possible fault behaviors.

In case the proof fails, Yosys will return a counter example, corresponding to fault events that cannot be covered by or should be excluded from the property. We further incorporate the counterexample into the PLA for property refining. Such iterations should continue until no additional counter example is returned or a maximum iteration limit is reached.

Our property extraction and refining idea is motivated by the recent progress in template matching and machine learning-based side channel analysis methods. By comparison, those methods typically create a template or model as a distinguisher for correct key selection. They

```
Setting up SAT problem:
Import set-constraint: \round_in_e [119:0] = 120'000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000
Import set-constraint: \rk_e = 128'00000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000
Final constraint equation: { \rk_e \round_in_e [119:0] } = 248'00000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
Imported 24396 cells to SAT database.
Import proof-constraint: \tag = 8'00000000
Final proof equation: \tag = 8'00000000

Solving problem with 146127 variables and 388304 clauses..
SAT proof finished - no model found: SUCCESS!

                  /$$$$$$      /$$$$$$$$     /$$$$$$$
                 /$$__  $$    | $$_____/    | $$__  $$
                | $$  \ $$    | $$          | $$  \ $$
                | $$  | $$    | $$$$$       | $$  | $$
                | $$  | $$    | $$__/       | $$  | $$
                | $$/$$ $$    | $$          | $$  | $$
                |  $$$$$$/ /$$| $$$$$$$$ /$$| $$$$$$$//$$
                 \____ $$$|__/|________/|__/|_______/|__/
                      \__/
```

Fig. 4 Formal verification result for fault property related to *rdo*0 and *rdo*2.

are generally lack of a formal analysis step to verify the effectiveness and completeness of the template or model created. Our method employs a formal solver to perform such a proof step, which provides the theoretic bias for our method.

Using our property extraction method, we can generate three properties for the AES round shown in Fig. 2 as formalized in Eqs. (9), (10), and (12). In our successive analysis, we denote these properties as P01, P12, and P123, respectively.

$$\text{assert} \quad rdo3_e = rdo0_e \oplus rdo1_e \tag{12}$$

The properties extracted when injecting faults into round eight of AES are similar to those generated by injecting faults into round nine. The difference lies in the byte orders in property mapping, which we will discuss in the following section. In addition, a single byte fault injected before Mixcolumns in round eight will spread to all 16 state bytes that can be divided into four independent groups for property mapping.

6 Fault Attack Through Formal Analysis

This section describes our fault analysis method through formal verification. We focus on single random faults and do not require knowledge about the location or type of fault injected. We assume the design components beyond the fault location correctly propagate fault effects. We collect correct and faulty ciphertexts pairs for fault analysis. Our method can recover the key and determine the fault location as well as the actual fault occurred through formal solver assisted property checking.

6.1 Property mapping

We have verified the completeness of the extracted properties for covering single-byte faults injected into the SubBytes, ShiftRows, and MixColumns transformations of round nine at different locations.

Table 2 shows the fault injection locations in round nine, the corresponding faulty bytes at S-Box inputs (i.e., $rdo0$, $rdo1$, $rdo2$, and $rdo3$) in round ten and the fault related property mapping relations. We can discover that there are four different property mapping patterns, which differ in byte order at the input of round ten.

From an attacker's perspective, the only observable information is the ciphertext. Thus, we need to further understand how the transforms in round ten would affect the fault patterns, locations, and correlations in the ciphertext output. According to the transforms of the AES algorithm, SubBytes would change the fault pattern in a state byte; ShiftRows only changes

Table 2 Fault locations and satisfied properties at the S-Box input of round ten.

Faulty location			Faulty byte of round 10 input	Corresponding property
SubByte	ShiftRow	MixColumn		
S_0	S_0	S_0	S_0, S_1, S_2, S_3	P01($rdo0, rdo1$), P12($rdo1, rdo2$), P123($rdo0, rdo1, rdo3$)
S_4	S_4	S_4	S_4, S_5, S_6, S_7	
S_8	S_8	S_8	S_8, S_9, S_{10}, S_{11}	
S_{12}	S_{12}	S_{12}	$S_{12}, S_{13}, S_{14}, S_{15}$	
S_1	S_{13}	S_{13}	$S_{12}, S_{13}, S_{14}, S_{15}$	P01($rdo1, rdo2$), P12($rdo2, rdo3$), P123($rdo1, rdo3, rdo0$)
S_5	S_1	S_1	S_0, S_1, S_2, S_3	
S_9	S_5	S_5	S_4, S_5, S_6, S_7	
S_{13}	S_9	S_9	S_8, S_9, S_{10}, S_{11}	
S_2	S_{10}	S_{10}	S_8, S_9, S_{10}, S_{11}	P01($rdo2, rdo0$), P12($rdo0, rdo3$), P123($rdo0, rdo2, rdo1$)
S_6	S_{14}	S_{14}	$S_{12}, S_{13}, S_{14}, S_{15}$	
S_{10}	S_2	S_2	S_0, S_1, S_2, S_3	
S_{14}	S_6	S_6	S_4, S_5, S_6, S_7	
S_3	S_7	S_7	S_4, S_5, S_6, S_7	P01($rdo3, rdo0$), P12($rdo0, rdo1$), P123($rdo0, rdo3, rdo2$)
S_7	S_{11}	S_{11}	S_8, S_9, S_{10}, S_{11}	
S_{11}	S_{15}	S_{15}	$S_{12}, S_{13}, S_{14}, S_{15}$	
S_{15}	S_3	S_3	S_0, S_1, S_2, S_3	

the position of a faulty byte without affecting its fault status; AddRoundKey does not have an effect on fault propagation. Table 3 shows the fault locations, fault correlations, and property mapping patterns at the ciphertext output.

Table 3 shows the four types of faulty ciphertext byte distributions and the corresponding properties satisfied. We can use the distribution patterns to select the ciphertext bytes for property mapping and leverage these properties for fault verification.

6.2 Fault analysis through formal verification

Figure 5 illustrates the design flow of our fault attack method through formal property checking. We select correct and faulty ciphertext bytes according to the fault combinations given in Table 3 as input constraints of the formal fault verification model. The fault related properties are specified as SystemVerilog assertions, which will be checked by formal solvers. We leave

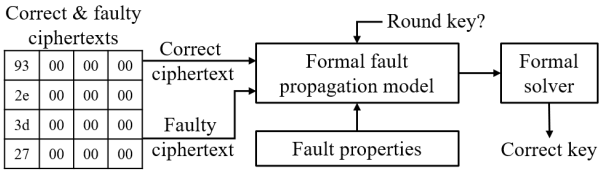


Fig. 5 Fault attack method through formal analysis.

Table 3 Faulty locations and satisfied properties at the ciphertext output.

Fault location in round 9			Fault in ciphertext	Corresponding property
SubByte	ShiftRow	MixColumn		
S_0	S_0	S_0	C_0, C_7, C_{10}, C_{13}	P01($rdo1, rdo2$), P12($rdo2, rdo3$), P123($rdo1, rdo3, rdo0$)
S_2	S_{10}	S_{10}	C_2, C_5, C_8, C_{15}	
S_9	S_5	S_5	$C_1, C_4, C_1 1, C_{14}$	
S_{11}	S_{15}	S_{15}	C_3, C_6, C_9, C_{12}	
S_1	S_{13}	S_{13}	C_3, C_6, C_9, C_{12}	P01($rdo3, rdo0$), P12($rdo0, rdo1$), P123($rdo0, rdo3, rdo2$)
S_3	S_6	S_6	C_1, C_4, C_{11}, C_{14}	
S_8	S_8	S_8	C_2, C_5, C_8, C_{15}	
S_{10}	S_2	S_2	C_0, C_7, C_{10}, C_{13}	
S_4	S_4	S_4	C_1, C_4, C_{11}, C_{14}	P01($rdo2, rdo0$), P12($rdo0, rdo3$), P123($rdo0, rdo2, rdo1$)
S_6	S_{14}	S_{14}	C_3, C_6, C_9, C_{12}	
S_{13}	S_9	S_9	C_2, C_5, C_8, C_{15}	
S_{15}	S_3	S_3	C_0, C_7, C_{10}, C_{13}	
S_5	S_1	S_1	C_0, C_7, C_{10}, C_{13}	P01($rdo0, rdo1$), P12($rdo1, rdo2$), P123($rdo0, rdo1, rdo3$)
S_7	S_{10}	S_{10}	C_2, C_5, C_8, C_{15}	
S_{12}	S_{12}	S_{12}	C_3, C_6, C_9, C_{12}	
S_{14}	S_6	S_6	C_1, C_4, C_{11}, C_{14}	

the round key as an unconstrained variable and employ a formal solver to search over the possible state space. When formal verification completes, the solver will return the correct key value.

The fault related properties are extracted from the correlations in fault effects within each column of the state matrix. Thus, we leverage the properties to recover the round key matrix column by column, which is 32 bits in width. If we construct a formal verification instance to retrieve 32 round key bits directly, the search complexity is $O(2^{32})$. Instead, we can take a divide and conquer solution to this search problem. We can first search for two round key bytes (i.e., 16 round key bits) using property P12 as the constraint, which has a search complexity of $O(2^{16})$. With these two key bytes determined, we can subsequently use them to reduce the search complexity of verifying properties P01 and P123 from $O(2^{16})$ and $O(2^{24})$, respectively, to $O(2^8)$. This results in an overall complexity of $O(2^{16}) + O(2^8) + O(2^8)$, which is at the scale of $O(2^{16})$.

As we will show in Section 7, using a single pair of correct and faulty ciphertext as input constraints may lead to multiple possible key candidates. We can create multiple parallel proof instances under different correct and faulty ciphertext pairs to prune out the incorrect key guesses. Figure 6 shows such a solution.

In practice, two or at most three proof instances are enough to find the unique correct key. The correct and faulty ciphertext pairs can be from different plaintexts or the same plaintext under different fault patterns. However, for the latter case, the fault location should be identical, i.e., faults should be injected into the same byte. Otherwise, more property mapping trails and proof runs are needed.

Our method does not need to perform fault diffusion analysis or use a huge number of fault traces to train a template or model as the key distinguisher. Instead, it implements fault attack through property extraction and checking. The attack only needs correct and faulty ciphertext pairs as input constraints for key search. The property set satisfied (i.e., the actual property mapping order) reveals the fault injection location.

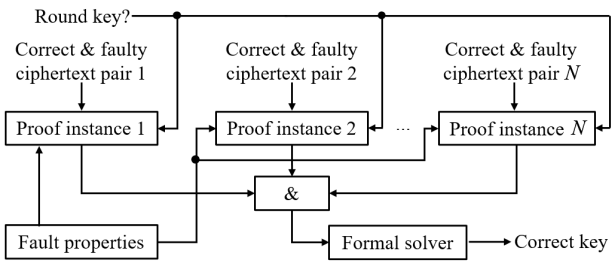


Fig. 6 Creating multiple proof instances to reduce the number of candidate keys.

7 Experimental Result

In this section, we present fault simulation and attack results. We perform fault attack on both unprotected and masked AES using our formal approach.

7.1 Fault fusion analysis

We use the AES core from the Aoki Laboratory to demonstrate how our formal model can be used to measure fault propagation effects. We first use a logic synthesis tool to convert the AES design into gate-level netlist and then create the formal fault propagation model for the design by discretely instantiating fault propagation models for the primitive gates in the netlist. In our model, each signal is augmented with a fault label denoted as $*_e$ to reflect the fault status of the signal. Since our model is described using standard HDL, we perform fault simulation under the ModelSim simulation tool. We inject faults into the input of round nine and observe the propagation of the faults all the way to the ciphertext output. Figure 7 shows the fault simulation results.

Consider the first test vector where we inject fault into the highest byte of the round nine input, i.e., $Din9_e$ =8'h47. After the SubBytes operation, the output of S-Box is 8'h87 while its fault status is 8'h12. This indicates that the faulty S-Box output should actually be 8'h87 $\oplus$ 8'h12 = 8'h95. The ShiftRows operation does not change the location of the faulty byte in the first row of the state matrix. The successive MixColumns operation spreads the fault into the first column, resulting in a fault status of 32'h24121236. The non-linear SubBytes transform in round ten changes the fault status to 32'h75DA324E. The next ShiftRows operation changes the fault locations to four discrete bytes in the ciphertext without changing the fault status. The simulation results precisely reveal the fault propagation behavior through the AES transforms.

We also inject faults into other locations in the first row of the state matrix of round nine input. Fault simulation shows the propagation of faults to different ciphertext bytes, which well agrees with the fault distributions and property mapping patterns as shown in Tables 2 and 3.

7.2 Fault attack on unprotected AES

We use the same AES core used in fault simulation

Din9[127:0] =	EA835CF00445332D655D98AD8596B123			
Din9_e[127:0] =	47000000000000000000000000000000	00000000560000000000000000000000	00000000000000003200000000000000	00000000000000000000000059000000
sb9[127:0] =	87EC4A8CF26EC3D84D4C46959790C826			
sb9_e[127:0] =	12000000000000000000000000000000	00000000F2000000000000000000000000	00000000000000001600000000000000	00000000000000000000000011000000
sr9[127:0] =	876E4626F24CC88C4D904AD897ECC395			
sr9_e[127:0] =	12000000000000000000000000000000	00000000F2000000000000000000000000	00000000000000001600000000000000	00000000000000000000000011000000
mx9[127:0] =	C7B70FF66FA5BA8AA3703AA64C9F42BC			
mx9_e[127:0] =	24121236000000000000000000000000	00000000FFF2F20D0000000000000000	0000000000000000 2C16163A00000000	00000000000000000000000022111133
rk9[127:0] =	549932D1F08557681093ED9CBE2C974E			
out9[127:0] =	932E3D279F20EDE2B3E3D73AF2B3D5F2			
out9_e[127:0] =	24121236000000000000000000000000	00000000FFF2F20D0000000000000000	0000000000000000 2C16163A00000000	00000000000000000000000022111133
sb10[127:0] =	DC3127CCDBB755986D110E80896D0389			
sb10_e[127:0] =	75DA324E000000000000000000000000	00000000B029547000000000000000	0000000000000000 B6F776E300000000	00000000000000000000000F9571FF1
sr10[127:0] =	DCB70E89DB1103CC6D6D279889315580			
sr10_e[127:0] =	75000000000004E0000320000DA0000	00020000B0000000000004700009500	0000760000F70000B6000000000000E3	000000F100001F0000570000F9000000
rk10[127:0] =	13111D7FE3944A17F307A78B4D2B30C5			
out10[127:0] =	CFA613F63885549DB9E6A8013C41A6545			
out10_e[127:0] =	75000000000004E0000320000DA0000	00020000B0000000000004700009500	0000760000F70000B6000000000000E3	000000F100001F0000570000F9000000

Fig. 7 Fault fusion simulation results.

for fault attack. We inject random faults before the MixColumns operation of round nine. We collect correct and faulty ciphertext pairs for the attack. The first step is to classify the faulty ciphertexts into four different groups according to the fault locations. We only need two correct ciphertexts denoted as $Dout$ and $Dout2$ as well as their fault labels $Dout\_e$ and $Dout\_e2$, where the fault labels are the exclusive OR of correct and faulty ciphertexts. The correct ciphertexts and their fault labels as mapped as input constraints of the formal fault propagation model of AES. The tag is a flag that indicates if both mapped properties are satisfied. We use the Yosys tool to prove the mapped properties and search for the key. Figure 8 shows the fault attack results through formal proof.

tag is 1 as shown in Fig. 8, it implies that both properties mapped with the correct ciphertexts and their fault labels as constraints are satisfied. By checking the property mapping patterns according to Table 3, we can determine the locations where the faults are actually injected. The sub, shf, and mix signals in Fig. 8 show the fault locations in the intermediate results of the SubBytes, ShiftRows, and MixColumns transforms, respectively.

We also injected faults into round eight of AES and

Signal Name	Dec	Hex	Bin
\Din	322741934	133ca6ae	00010011001111001010011010101110
\Din2	322741934	133ca6ae	00010011001111001010011010101110
\Din_e	-2073206680	846d6068	10000100011011010110000001101000
\Din_e2	187214498	b28aaa2	00001011001010001010101010100010
\Dout	1862042575	6efc83cf	01101110111111001000001111001111
\Dout2	1862042575	6efc83cf	01101110111111001000001111001111
\Dout_e	1277917038	4c2b776e	01001100001010110111011101101110
\Dout_e2	941820689	38230b11	00111000001000110000101100010001
\mix1	0	0	0000
\mix2	0	0	0000
\rk10	320317227	1317a72b	00010011000101111010011100101011
\shf1	0	0	0000
\shf2	0	0	0000
\sub1	0	0	0000
\sub2	0	0	0000
\tag	1	1	1
\type	0	0	00

```
Solving problem with 23527 variables and 62773 clauses..
SAT solving finished - no more models found (after 1 distinct solutions).
```

Fig. 8 Fault attack results on round nine of AES.

successfully recovered the last round key. When faults are injected before the MixColumns operation of round eight, we can recover 128 key bits with two pairs of correct and fault ciphertexts. By comparison, 32 key bits can be recovered if faults only propagate through a single MixColumns operation. For the latter case, we need additional fault injections at different locations to recover the entire key.

Once the properties are exacted and verified using the method described in Section 5, we do not rely on knowledge about the input values of intermediate encryption results during fault analysis. We only need correct and faulty ciphertext pairs as input constraints to search for the round key by formally checking which properties can be satisfied.

We perform additional tests to understand how the number of key candidates changes when using different numbers of pairs of correct and faulty ciphertexts. The experimental results are shown in Fig. 9.

From Fig. 9, when using a single pair of correct and faulty ciphertexts, searching for 32 key bits only returns 2077 candidates in the worst case while 866 candidates in the best case. On average, around 1000 key candidates will be reported. With an additional pair, the number of key candidates is no more than three. When using three

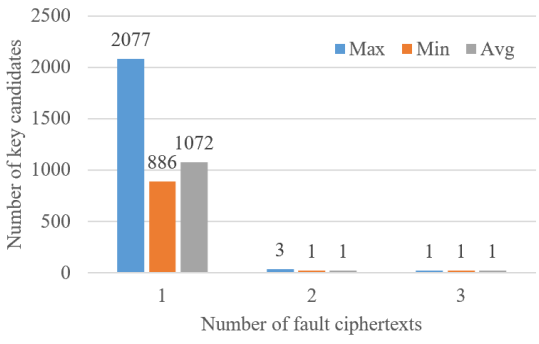


Fig. 9 Number of key candidates using different number of pairs of correct and faulty ciphertexts.

pairs, the key candidate is unique.

7.3 Fault attack on RSM AES

We further use the RSM AES core[28] for fault analysis. We create the formal fault propagation model for the masked AES core in a similar way. We then inject faults before the MixColumns operation in round eight of the masked design. Two correct and faulty ciphertext pairs are randomly selected as input constraints to the fault verification model and the Yosys tool is employed to prove the mapped properties and search for the last round key. As shown in Fig. 10, the 128-bit key can be successfully recovered with two pairs of correct and faulty ciphertexts by proving four groups of properties associated with four tag bits, i.e., $tag = 1111$. In the meanwhile, the fault location is automatically determined. Specifically, $location1$ and $location2$ are both zero, indicating that faults are both injected into the first byte.

We make a comparison with other common fault attack methods in Table 4.

The proposed formal analysis based fault attack approach has relatively lower attack efforts and key search complexity. In addition, it does not need to perform complex fault fusion analysis or require knowledge about the location or type of fault injected.

In our tests, we use a static analysis tool (SAT) to search for the key, which can be insufficient for sequential implementations of cryptographic functions. To overcome such a drawback, we unroll the round iterations of a given sequential implementation as a fully pipelined architecture and then remove the registers to make the entire design combinational for SAT analysis. Another possible solution is to use more powerful commercial formal solvers that can better support sequential designs.

In addition to the unprotected and masked AES cores, we have also successfully attacked other block ciphers such as SM4 and the lightweight cipher LED64 using our fault related property extraction and formal verification approach. The attack implementation details vary slightly due to the difference in fault fusion characteristics of these block ciphers.

8 Conclusion

This paper proposes a new fault analysis method through formal verification. We establish a formal fault propagation model for precisely measuring the fault propagation effects and exploit the correlation in fault effects as properties to guide a formal solver to search for the key. The proposed method does not require knowledge about the location or type of fault injected. Instead, it can determine these two pieces of information by checking the properties satisfied. Experimental results using both unprotected and masked AES implementations show that our method has a key search complexity below 2^{16}, requiring only two pairs of correct and faulty ciphertexts to recover the key. We will explore this fault analysis approach on more block ciphers such as PRESENT and IDEA in our future work.

Acknowledgment

This research is supported by the National Key R&D Program of China (No. 2021YFB3100901) and the National Natural Science Foundation of China (Nos. 62074131 and 62004176).

Table 4 Performance comparison of different fault attack methods.

Metric	This work	CFA	PFA	DFA
Pairs of ciphertexts	2	2	1641	2
Search complexity using a pair of ciphertexts	2^{16}	2^{16}	–	2^{16}
Search complexity using two pairs of ciphertexts	2^{16}	2^{17}	–	2^{17}
Algorithmic details required	No	Yes	Yes	Yes
Fault fusion analysis required	No	Yes	Yes	Yes

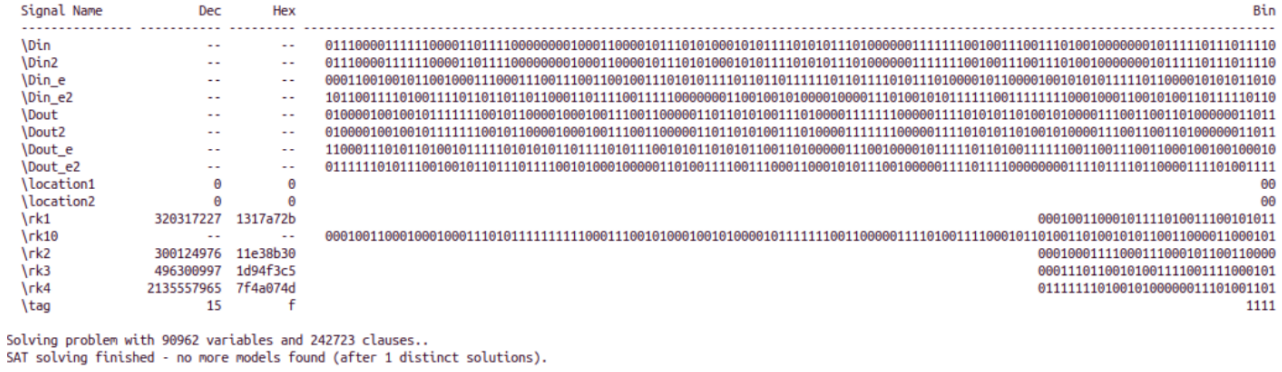


Fig. 10 Fault attack results on round eight of RSM AES.

References

[1] P. C. Kocher, Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems, in *Advances in Cryptology—CRYPTO'96*, N. Koblitz, ed. Berlin, Germany: Springer, 1996, pp. 104–113.

[2] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, Systematic classification of side-channel attacks: A case study for mobile devices, *IEEE Commun. Surv. Tutor.*, vol. 20, no. 1, pp. 465–488, 2018.

[3] A. Boscher and H. Handschuh, Masking does not protect against differential fault attacks, in *Proc. 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, Washington, DC, USA, 2008, pp. 35–40.

[4] W. Hu, A. Althoff, A. Ardeshiricham, and R. Kastner, Towards property driven hardware security, in *Proc. 2016 17th Int. Workshop on Microprocessor and SOC Test and Verification* (*MTV*), Austin, TX, USA, 2017, pp. 51–56.

[5] H. Wang, H. Li, F. Rahman, M. M. Tehranipoor, and F. Farahmandi, SoFI: Security property-driven vulnerability assessments of ICs against fault-injection attacks, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 3, pp. 452–465, 2022.

[6] D. Boneh, R. A. DeMillo, and R. J. Lipton, On the importance of checking cryptographic protocols for faults, in *Advances in Cryptology — EUROCRYPT'97*, W. Fumy, Ed. Berlin, Germany: Springer, 1997, pp. 37–51.

[7] E. Biham and A. Shamir, Differential fault analysis of secret key cryptosystems, in *Advances in Cryptology—CRYPTO'97*, W. Fumy, Ed. Berlin, Germany: Springer, 1997, pp. 513–525.

[8] N. T. Courtois, K. Jackson, and D. Ware, Fault-algebraic attacks on inner rounds of DES, in *Proc. the Strategies Telecom and Multimedia*, Montreuil, France. 2010.

[9] C. Clavier and A. Wurcker, Reverse engineering of a secret AES-like cipher by ineffective fault analysis, in *Proc. 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Los Alamitos, CA, USA, 2013, pp. 119–128.

[10] A. Wang, M. Chen, Z. Wang, and X. Wang, Fault rate analysis: Breaking masked AES hardware implementations efficiently, *IEEE Trans. Circuits Syst. II*, vol. 60, no. 8, pp. 517–521, 2013.

[11] T. Fuhr, E. Jaulmes, V. Lomné, and A. Thillard, Fault attacks on AES with faulty ciphertexts only, in *Proc. 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Los Alamitos, CA, USA, 2013, pp. 108–118.

[12] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, Sifa: Exploiting ineffective fault inductions on symmetric cryptography, *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 547–572, 2018.

[13] F. Zhang, X. Lou, X. Zhao, S. Bhasin, W. He, R. Ding, S. Qureshi, and K. Ren, Persistent fault analysis on block ciphers, *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp.150–172, 2018.

[14] G. Wang and S. Wang, Differential fault analysis on PRESENT key schedule, in *Proc. 2010 Int. Conf. Computational Intelligence and Securit*y, Nanning, China, 2011, pp. 362–366.

[15] H. Momeni, M. Masoumi, and A. Dehghan, A practical fault induction attack against an FPGA implementation of AES cryptosystem, in *Proc. World Congress on Internet Security* (*WorldCIS-2013*), London, UK, 2014, pp. 134–138.

[16] S. S. Ali and D. Mukhopadhyay, A differential fault analysis on AES key schedule using single fault, in *Proc. 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Nara, Japan, 2011, pp. 35–42.

[17] X. X. Wang, H. Wei, T. Jing, Z. Jiacheng, and T. Shibo, Correlation fault attack on aes, *Journal of Xidian University*, vol. 48, no. 4, pp. 192–199, 2021.

[18] J. Takahashi and T. Fukunaga, Differential fault analysis on AES with 192 and 256-bit key, in *Proc. 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Santa Barbara, CA, USA, 2010, pp. 3–9.

[19] L. Han, N. Wu, F. Ge, F. Zhou, J. Wen, and P. Qing, Differential fault attack for the iterative operation of AES-192 key expansion, in *Proc. 2020 IEEE 20th Int. Conf. Communication Technology* (*ICCT*), Nanning, China, 2020, pp. 1156–1160.

[20] F. Zhang, S. Guo, X. Zhao, T. Wang, J. Yang, F. X. Standaert, and D. Gu, A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers, *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 5, pp. 1039–1054, 2016.

[21] M. Gay, T. Paxian, D. Upadhyaya, B. Becker, and I. Polian, Hardware-oriented algebraic fault attack framework with multiple fault injection support, in *Proc. 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography* (*FDTC*), Atlanta, GA, USA, 2019, pp. 25–32.

[22] S. Saha, M. Alam, A. Bag, D. Mukhopadhyay, and P. Dasgupta, Leakage assessment in fault attacks: A deep learning perspective, https://eprint.iacr.org/2020/306, 2020.

[23] F. Zhang, Y. Zhang, H. Jiang, X. Zhu, S. Bhasin, X. Zhao, Z. Liu, D. Gu, and K. Ren, Persistent fault attack in practice, *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 172–195, 2020.

[24] G. Xu, F. Zhang, B. Yang, X. Zhao, W. He, and K. Ren, Pushing the limit of PFA: Enhanced persistent fault analysis on block ciphers, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 40, no. 6, pp. 1102–1116, 2021.

[25] K. Bae, S. Moon, D. Choi, Y. Choi, H. D. Kim, and J. Ha, A practical analysis of fault attack countermeasure on AES using data masking, in *Proc. Int. Conf. Computing and Convergence Technology* (*ICCCT*), Seoul, Republic of Korea, 2012, pp. 508–513.

[26] X. Wang, J. Zheng, L. Wu, J. Zhu, and W. Hu, A correlation fault attack on rotating S-box masking AES, in *Proc. 2021 Asian Hardware Oriented Security and Trust Symp.* (*AsianHOST*), Shanghai, China, 2022, pp. 1–6.

[27] C. Dobraunig, M. Eichlseder, H. Gross, S. Mangard, F. Mendel, and R. Primas, Statistical ineffective fault attacks on masked AES with fault countermeasures, in *Proc. 24th Int. Conf. Theory and Application of Cryptology and Information Security*, Taipei, China, 2018, pp. 315–342.

[28] Description of the masked AES of the DPA contest v4, https://www.dpacontest.org/v4/data/rsm/aes-rsm.pdf, 2022.

Xiaojie Dai received the MS degree from Tsinghua University in 2013. She is currently pursuing the PhD degree in Northwestern Polytechnical University, China. Her current research interests are hardware security, formal security verification, and cryptanalysis.

Xingxin Wang received the MS degree from Northwestern Polytechnical University in 2023. She is currently pursuing the PhD degree with School of Cybersecurity in the same university. Her current research interests are hardware security, including side channel analysis, fault injection attack, and cryptanalysis.

Xue Qu received the BS degree in network engineering from Qufu Normal University in 2021. She is currently pursuing the MS degree at the School of Cybersecurity in Northwestern Polytechnical University, China. Her current research interest is hardware security with a focus on side channel analysis.

Baolei Mao received the PhD degree in network and information security from Northwestern Polytechnical University in 2018. He is currently with Zhengzhou University, China. He is a recipient of Internet Security Graduate Student Award from China Internet Development Foundation. His research interests are side channel analysis, hardware information flow analysis, and formal verification.

Wei Hu is a full professor with the School of Cybersecurity, Northwestern Polytechnical University, China. He received the PhD degree in control science and engineering from the same university in 2012. His research interests are hardware security, cryptography, formal security verification, and reconfigurable computing.