Using State Space Encoding To Counter Biased Fault Attacks on AES Countermeasures

Sikhar Patranabis, Abhishek Chakraborty, Debdeep Mukhopadhyay, and P.P. Chakrabarti
Department of Computer Science and Engineering
IIT Kharagpur, India
{sikhar.patranabis, abhishek.chakraborty, debdeep, ppchak}@cse.iitkgp.ernet.in

***Abstract*—Biased fault attacks such as the Differential Fault Intensity Analysis (DFIA) have been a major threat to cryptosystems in recent times. DFIA combines principles of side channel analysis and fault attacks to try and extract the key using faulty ciphertexts only. Biased fault attacks have also been shown to weaken traditional redundancy based countermeasures, such as Concurrent Error Detection (CED) techniques, that provide security against classical fault attacks such as Differential Fault Analysis (DFA). While these countermeasures are effective under the assumption that the adversary uses a uniform fault model, they are vulnerable to attacks using biased fault models. Till date, no effective countermeasure against such biased fault attacks has been reported in literature. In this work, we propose a countermeasure strategy that combines the principles of redundancy with that of fault space transformation to achieve security against both classical and biased fault attacks. The novelty in the proposed countermeasure lies in the concept of transforming the fault space, that reduces the probability that the adversary can bypass the redundant checks by introducing the same fault in the original and redundant computations. All claims have been validated via practical experiments on a SASEBO GII board.**

***Index Terms*—Cryptanalysis, Time Redundancy, Biased Faults, AES**

I. Introduction

Side channel analysis (SCA) attacks such as the Differential Power Analysis (DPA) and the Correlation Power Analysis(CPA), as well as active fault analysis attacks (FA) such as Differential Fault Analysis(DFA) and Differential Fault Intensity Analysis (DFIA) on cryptographic devices have raised serious issues regarding the security of cryptosystems which were previously considered robust and secure owing to their strong mathematical algorithms. Security against classical cryptanalytic attacks such as linear and differential cryptanalysis is no longer a sufficient criteria for an algorithm to be termed as secure. Several newly devised cryptosystems employing AES block ciphers that are secure against classical cryptanalysis have been rendered vulnerable by SCA and FA owing to leakages from their implementations.

Active fault analysis of block ciphers has been studied extensively in literature since the seminal work by Boneh *et.al* [10] in 1996 that demonstrated fault attacks on the RSA cryptosystem. Their work triggered an extensive study of fault analysis with respect to all popular cryptosystems, including symmetric key systems such as the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES).AES is the standard secret key algorithm today and is employed extensively for security in a number of consumer products and dedicated hardware such as smart cards, servers, FPGAs, and TV set-top boxes. Although AES is secure against classical cryptanalysis [28], a hardware implementation of AES, unless carefully designed, could result in security vulnerabilities. An adversary could inject malicious faults into a cryptographic device and build correlations between faulty and fault free ciphertexts to be able to drastically reduce the search key space and extract the key in a short time. This technique of using faulty and fault free ciphertext pairs to extract the key is popularly known in literature as Differential Fault Analysis (DFA). DFA of of AES has been studied extensively [20], [33], [37], [42], [43] and has been demonstrated to be both practical and inexpensive [4], [5], [21], [29]. An adversary can inject both transient faults and permanent or stuck-at faults into AES implementations using clock glitches [2], [3], [46], power glitches [6], [12], [32], [49], laser guns [1], [11], [12] and even electromagnetic pulses [13]. While most DFAs focus on the AES data path, DFA of the AES key schedule has also been proposed. State of the art DFA techniques use a variety of practically achievable fault models to extract the key in an efficient manner and may require as few as a single fault injection to recover the entire key [50].

Recent literature has focused on an alternate variety of active fault analysis that requires only faulty ciphertexts instead of pairs of faulty and fault free ciphertexts. The idea was first proposed by Fuhr *et.al* in FDTC 2013 [17] and was further extended by Ghalaty *et.al* to Differential Fault Intensity Analysis (DFIA) that combines fault analysis with DPA. The approaches are similar in their use of biased fault models to try and recover the secret key. The basic idea is to make several key hypotheses on the affected state bytes in order to retrieve a key dependent state value whose distribution is strongly biased. An alternative approach to DFIA, namely the Fault Differential Entropy(FDE) has been proposed in [24]. FDE also requires only faulty ciphertexts and uses fault differentials to distinguish the correct key hypothesis. Similar to DFIA, this technique also exploits the bias in the underlying fault model and has been found to weaken several classical CED techniques. The advantage of using biased fault models is that while correct key hypothesis produces small changes to the faulty state, incorrect ones infer big, random changes. The usage of appropriate distinguisher functions therefore allows recovery of the correct key with a practically feasible number of fault injections. However, none of these works present a

mathematical basis to quantify the bias of a fault model or compare the degree of bias of various fault models.

With fault attacks now being an established and potent threat to cryptosystems, sound countermeasures against them must be introduced into cryptographic devices. Countermeasures against DFA come in two major flavors - detection based countermeasures and infection based countermeasures. Detection based measures essentially use concurrent error detection (CED) [12] and are used for both reliability and security purposes. CEDs use four major types of redundancies that have been studied in literature - information [7], [30], [41], [53], time [38], [39], hardware [26], [39] and hybrid [22], [23], [31], [47] redundancies. Detection based countermeasures are vulnerable to attacks to the comparison step itself. In this paper, we explore other variety of fault attack that could render CEDs vulnerable. In particular, we focus on the time and hardware redundancy countermeasures for AES-128. Both time and hardware redundancy are described in literature as effective classical fault tolerance technique that work well against fault attack models where the fault distribution is uniform. In this paper, we analyze the security of both schemes in the light of biased fault models and demonstrate that the bias in the fault model indeed weakens the countermeasure significantly.

The other important challenge is to develop countermeasure schemes to counter biased fault attacks. Since most biased fault attacks such as DFIA combine principles of side channel analysis with active fault analysis, it would be interesting to investigate the effectiveness of standard countermeasures to SCA against biased fault attacks. One such technique is masking. Masking involves concealing any intermediate value related with the key with a mask that is is randomly chosen and varies from encryption to encryption []. In this paper, we demonstrate that masking fails to counter biased fault attacks on time and hardware redundancy countermeasures. This also motivates developing alternative countermeasure schemes that can effectively counter biased fault attacks.

Contributions. This paper demonstrates that naive implementations of classical detection based countermeasure techniques, specifically the time and hardware redundancy countermeasures, are vulnerable to biased fault attacks. In order to obtain a faulty ciphertext by attacking a time or hardware redundant implementation of AES, the adversary must introduce exactly the same fault in both the original and redundant computations corresponding to the same round. When the fault distribution is unbiased (as classically assumed) the probability of occurrence of this event is very low. But a biased fault model augments this probability to the extent that it is feasible to obtain sufficient number of faulty ciphertexts to recover the key, while using only a practical number of fault injections. The paper first proposes a mathematical quantification of the bias of a fault model using the variance of the probability distribution of various faults in the fault model. The paper then proposes a second order fault analysis of the time and hardware redundancy countermeasure implementations using a practically achievable biased fault model. We then explore possible countermeasure strategies against our proposed attack strategy. A popular choice of countermeasure against side channel attacks such as DPA is boolean masking. We demonstrate in this paper that although biased fault attacks are essentially a form of side channel analysis, masking fails to counter these attacks since it does not transform the fault space between the original and redundant rounds. The paper then proposes an alternative countermeasure scheme against biased fault attacks. The proposed scheme uses the concept of fault space mapping to thwart the adversary from being able to inject the same faults in the original and redundant rounds to beat the time and hardware redundant schemes. All results have been validated via simulations and real life experiments on a Spartan 3A FPGA on a SASEBO GII platform.

Organization. The rest of the paper is organized as follows. Section II introduces the time and hardware redundancy countermeasures and mentions briefly the attack strategy used in the paper. Section IV then presents a mathematical quantification of the bias of the fault model using the variance of the probability distribution. Section III presents a practically achievable precise cum biased fault model obtained via clock glitches that can be used to attack the countermeasure schemes. This is followed by a description of the actual attack procedure in Section V. We then focus on countermeasure techniques against our proposed attack. Section VII demonstrates that boolean masking fails to counter biased fault attacks. An alternative countermeasure scheme is developed in Section VIII using the concept of fault space transformation. Finally, the effectiveness of the countermeasure scheme is verified via practical experiments performed on a Spartan 3A FPGA on a SASEB0 GII platform in Section IX followed by conclusions in Section X.

II. Preliminaries

A. Fault Attacks on AES

Recent research has focused on two broad categories of fault analysis of AES - attacks that require correct and faulty ciphertext pairs, and attacks that require faulty ciphertexts only. The first category principally includes Differential Fault Analysis(DFA). In DFA, the adversary compares the response of the cipher with and without fault injections [8], [44], [51], [34]. The other category of fault attacks on AES require only faulty ciphertexts to retrieve the key, as proposed by Fuhr et. al. [18]. The attack uses stuck-at fault models and depends on the degree of control the adversary has on the distribution of the injected fault. A very similar approach proposed by Ghalaty et. al. is the Differential Fault Intensity Analysis (DFIA) [19] that uses a biased but slightly less restrictive single byte fault model. Both these approaches make several key hypotheses on the affected state bytes in order to retrieve a hypothetical value whose distribution is strongly biased. An alternative approach to DFIA, namely the Fault Differential Entropy(FDE) has been proposed in []. FDE also requires only faulty ciphertexts and uses fault differentials to distinguish the correct key hypothesis. Similar to DFIA, this technique also exploits the bias in the underlying fault model and has been found to weaken several classical CED techniques. Table

TABLE I: Fault Attacks on AES: State of the art

Attack Nature	Fault Model	No. of Faulty Ciphertexts	Key Search Space	Fault Source			
				Clock	Power	Laser	EM
DFA	**Faults are injected in any round and at any location**						
	Random	2^{128}	2^{128}				
	Faults are injected in round 0 into AddRoundKey						
	Single Bit [9]	128	1				
	Faults are injected between the 7^{th} round MixColumns output and the 8^{th} round MixColumns input						
	Single Byte [43]	2	2^{40}		[49]	[15]	[14]
	Single Byte [42]	2	2^{32}			[15]	[14]
	Single Byte [50]	1	2^{8}			[15]	[14]
	Multiple Byte DM_0 [46]	1	2^{32}	[46]			[14]
	DM_1 [46]	1	2^{64}	[46]			[14]
	DM_2 [46]	1	2^{96}	[46]			[14]
	DM_3 [46]	1	2^{128}	[46]			[14]
	Faults are injected between the 8^{th} round MixColumns output and the 9^{th} round MixColumns input						
	Single Bit [20]	≈ 50	1	[1]			[14]
	Single Byte [16]	≈ 40	1		[6]	[15]	[14]
	Single Byte [40]	6	1			[15]	[14]
	Multiple Byte DM_0 [40]	6	1				[14]
	DM_0 [40]	1.5	1				[14]
	Faults are injected in the 10^{th} round						
	Single Byte [36]	≈ 36	2^{12}	[36]			
DFIA	**Faults are injected just after the penultimate AddRoundKey operation**						
	Single Byte [18], [19]	10-14	2^{12}	[19]			
	Faults are injected just after the ante-penultimate AddRoundKey operation						
	Single Byte [18]	14-80	2^{34}				
	Faults are injected in the 7^{th} round						
	Diagonal Fault (Stuck-at) [18]	4-10	2^{32}				

I summarizes the different versions of fault attacks on AES reported in literature.

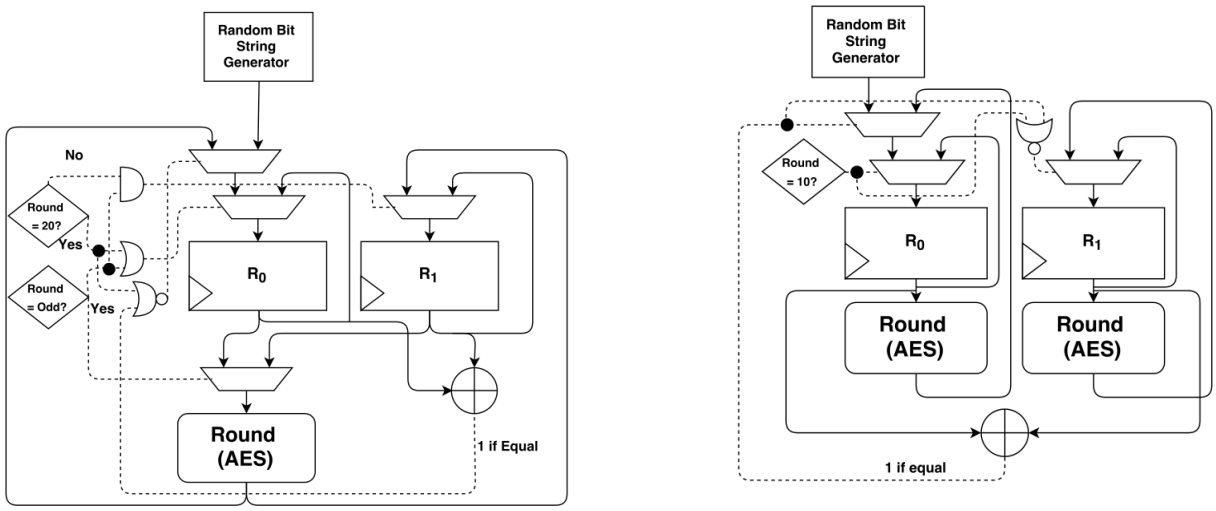
In this paper, we demonstrate that biased fault attacks can be used to weaken both the time and hardware redundancy techniques. Biased fault models expose a significant vulnerability of the classical detection based countermeasures. Unlike in a uniform fault distribution, a biased fault distribution implies that the adversary can introduce the same fault in both the normal and the redundant computation cycles with high probability. This reduces the number of fault injections required per faulty ciphertext and makes the attack practically feasible. As in DFIA, our attack achieves the desired fault distribution using clock glitches at various frequencies. We then propose a countermeasure scheme that uses fault space transformations to thwart biased fault attacks on CEDs.

B. The Time and Hardware Redundancy Countermeasures

Figures 1a and 1b illustrate the use of time and hardware redundancy respectively in fault detection. Time redundancy is a fault tolerance technique that uses additional time to perform the functions of a system multiple times and compares the results to detect faults if any. A particular advantage of this approach is its low area overhead. The basic time redundancy technique has essentially three important aspects - repetition of function computation, storage of results of original and redundant computations and comparison of results for fault detection. Hardware redundancy, on the other hand, uses two copies of the same hardware that operate in parallel and the results are compared after each operation. Hardware redundancy has greater area footprint as compared to time redundancy but is more efficient in terms of time requirement. Both time and hardware redundancy belong to a broad class of detection based countermeasures, also known as concurrent error detection(CED) techniques against DFA []. Some notable CEDs using time redundancy proposed in literature include re-computation [39] as well as double data rate computation [38]. Hardware redundancy based CEDs have been proposed in [26], [39].

A popular second order fault attack against CEDs already mentioned in literature involves injecting a fault in the state of the implementation, followed by a second fault injection to bypass the detection step []. An interesting alternative to this

Fig. 1: The Target Concurrent Error Detection Techniques



(a) Time Redundancy

(b) Hardware Redundancy

approach is to explore the possibility of bypassing the fault detection step by injecting the same fault in both the original and redundant computations. This is difficult to achieve for a classical uniform fault model. However, it seems to be a rather promising technique in a biased fault setting where not all faults occur with equal probability.

III. Fault Model and Fault Injection Set Up

In this section, we describe the fault model used for our attack and the fault injection set up employed to achieve this fault model.

A. Fault Model

Depending on the type and method of fault injection, different types of faults may occur with varying granularity such as single bit upsets, multi bit upsets, single and multi byte upsets, and diagonal upsets. Some previous works have considered *random effect on one byte*, where a single state byte may have changed to any random value [25], [8], [35], [51]. However, such a fault model has a uniform distribution. More recent work [19] has demonstrated that single-bit, two-bit, three-bit and four-bit upsets are achievable using clock glitches, and that one can control the granularity of fault injection by varying the fault intensity. We have ourselves verified that such faults can be achieved in hardware implementations of AES-128 via introduction of clock glitches at varying frequencies (refer III-B).

For further discussions in this paper , we distinguish between major classes of faults that covers the entire possible fault state. Table IIa summarizes these categories. Our experiments have shown that SBU is the most suitable fault model for our attacks on time or hardware redundant AES implementations. However, we also present results for SBDBU, SBTBU and SBQBU to show the impact of fault model granularity on the performance of our attacks. Note that the degree of control that the attacker has on the fault location impacts the fault models in terms of the number of possible fault (N) under that fault model. We distinguish between the following two situations - Situation-1 when the attacker has perfect control over the faulty byte and Situation-2 when the attacker does not have control over the faulty byte.

In the case of single byte faults, if k be the number of bit upsets in the target byte, then the number of possible faults in either scenario is different. In Situation-1, any k bits of the fixed target byte is affected, so number of possible faults is $\binom{8}{k}$. In Situation-2, however k bits of any target byte could be affected, so number of possible faults is $16\binom{8}{k}$, which is 16 times greater than in Situation-1.

Table IIb captures the number of possible faults under various fault models in both situations. Evidently, precision in terms of fault location restricts the set of possible faults under a fault model significantly. Note that n is the total number of faults possible under the fault model.

B. Fault Injection Set Up

Figure 2 describes our set up for fault injection in a CED based implementation of AES-128.The set up consists of an FPGA (Spartan-3A XC3S400A), a PC and an external arbitrary function generator (Tektronix AFG3252). The FPGA has a DUT (Device Under Test) block, which is a time or hardware redundant AES implementation. Faults are injected using clock glitches and the fault intensity is controlled by increasing/decreasing the glitch frequency. The system has two clock signals - clk_{slow} and clk_{fast}, derived from an external clock signal clk_{ext} via a Xilinx Digital Clock Manager (DCM) module. The clk_{ext} is generated by the external function generator and can take frequency values up to 120 MHz. The clk_{slow} signal has the same frequency as clk_{ext} and is used for fault-free operation of the DUT. The clk_{fast} signal has a frequency equal to twice the frequency of clk_{ext} and is used to create the glitches for fault injection. The appropriate signal is fed to the DUT via a MUX. The select line of the MUX

TABLE II: Fault Model Description

(a) The Fault Model

Symbol	Fault Model
FF	Fault Free
SBU	Single Bit Upset
SBDBU	Single Byte Double Bit Upset
SBTBU	Single Byte Triple Bit Upset
SBQBU	Single Byte Quadruple Bit Upset
OSB	Other Single Byte Faults
MB	Multiple Byte Faults

(b) Impact of fault location precision

Fault Model	Faults Possible(n) (Situation-1)	Faults Possible(n) (Situation-2)
SBU	8	128
SBDBU	28	448
SBTBU	56	896
SBQBU	70	1120
OSB	93	1488

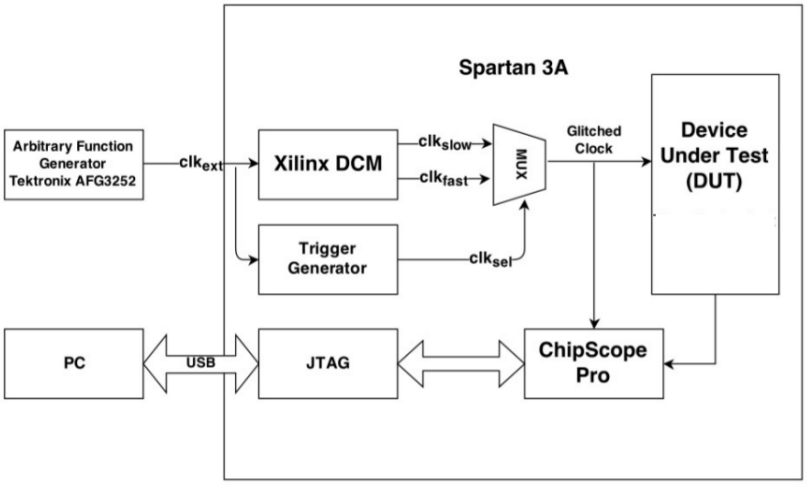


Fig. 2: Fault Injection Setup

is the clk_{sel} signal which is output by the trigger generator and is set to high when clk_{fast} is to be fed to the DUT. The faulty states of the registers were monitored using Chipscope Pro 12.3 analyzer.

We injected faults in both the original and redundant computations of the target round of the DUT by varying the clk_{ext} over a wide range of frequencies. The target rounds were chosen to be rounds 8 and 9 respectively. Since the Chipscope pro 12.3 Analyzer limits the number of observable samples at a given frequency to 1024, we observed 512 samples for the original computation and 512 samples for the redundant computation. Table IIIa summarize the average fault patterns obtained in either case, while Table IV elucidates the common frequency ranges between either round where each type of fault model is predominant.

TABLE IV: Fault Models and Corresponding Frequency Ranges

Fault Model	Frequency Range (Original and redundant computations) (MHz)
FF	< 125.3
SBU	125.3-125.4
SBDBU	125.6-125.7
SBTBU	126.0-126.1
SBQBU	126.3-126.4
OSB	126.5
MB	> 127.2

IV. Quantifying the Bias of a Fault Model

In this section, we present a formal quantification of the degree of bias of a fault model. A fault model may be defined as a two tuple $(\mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the fault space and $\mathcal{P}$ is the fault probability distribution followed by the model. Let the set of faults that can occur under a given fault space $\mathcal{F}$ is given by $\mathcal{F} = \{f_1, \ldots, f_i, \ldots, f_n\}$, where n is the total number of faults possible under the fault model. Let F be a discrete random variable that denotes the outcome of a single fault injection under this fault model, and let p_i be the probability of occurrence of fault f_i given by $p_i = Pr[F = f_i]$. Evidently, the fault model follows the probability distribution $\mathcal{P} = \{p_1, \ldots, p_i, \ldots, p_n\}$. Also, let Var denote that variance of $\mathcal{P}$. From the standard definition of variance of a probability distribution $Var = \frac{\sum_{i=1}^{n} {p_i}^2}{n} - \frac{1}{n^2}$. We now propose the following definition for the degree of bias of a fault model.

Definition 1: The degree of bias of a fault model is defined to be equal to the variance Var of the probability distribution $\mathcal{P}$ followed by the fault model.

1) Justification for the definition:: In literature, a fault model is said to be unbiased or uniform if all faults under this fault model occur with equal probability. If the fault model does not satisfy the above criteria is said to be biased. For a uniform fault model, $p_i = p_j = \frac{1}{n} \forall i, j$ and $Var = 0$. Thus according to our definition, the bias of a uniform fault model is 0. On the other hand, for a biased fault model Var is a finite non-zero value and so is the bias. In the most biased scenario, the outcome of the fault injection is deterministic, that is, exactly one fault occurs with a probability of 1 and all the rest have a zero probability of occurrence. in such a scenario the variance $Var = \frac{1}{n} - \frac{1}{n^2}$. We claim that this is

TABLE III: Fault Distribution

(a) Fault Distribution Pattern - Time Redundancy

Fast Clock Frequency (MHz)	FF	SBU	SBDBU	SBTBU	SBQBU	OSB	MB
125.0	512	0	0	0	0	0	0
125.1	503	9	0	0	0	0	0
125.2	489	22	1	0	0	0	0
125.3	456	50	6	0	0	0	0
125.4	425	59	22	6	0	0	0
125.5	396	45	43	28	0	0	0
125.6	354	34	112	32	0	0	0
125.7	303	23	101	85	0	0	0
125.8	260	11	55	86	0	0	0
125.9	208	5	46	147	6	0	0
126.0	176	1	39	228	68	0	0
126.1	143	0	18	211	136	4	0
126.2	115	0	10	94	178	15	0
126.3	101	0	8	95	251	49	8
126.4	65	0	9	45	232	141	20
126.5	32	0	5	16	131	187	141
126.6	13	0	3	8	98	101	289
126.7	5	0	1	4	32	112	358
126.8	0	0	1	2	5	105	399
126.9	0	0	1	2	3	88	421
127.0	0	0	0	1	2	33	476
127.1	0	0	0	0	1	12	499
127.2	0	0	0	0	0	0	512
127.3	0	0	0	0	0	0	512
127.4	0	0	0	0	0	0	512
127.5	0	0	0	0	0	0	512

(b) Fault Distribution Pattern - Hardware Redundancy

Fast Clock Frequency (MHz)	FF	SBU	SBDBU	SBTBU	SBQBU	OSB	MB
70.0	512	0	0	0	0	0	0
70.1	512	0	0	0	0	0	0
70.2	504	8	0	0	0	0	0
70.3	475	34	3	0	0	0	0
70.4	460	47	5	0	0	0	0
70.5	416	63	29	4	0	0	0
70.6	378	38	71	25	0	0	0
70.7	345	29	120	32	0	0	0
70.8	299	21	164	28	0	0	0
70.9	234	14	120	144	2	0	0
71.0	216	4	39	247	6	0	0
71.1	189	2	35	220	66	0	0
71.2	130	0	15	180	176	11	0
71.3	105	0	10	104	278	15	0
71.4	83	0	10	66	227	100	26
71.5	50	0	8	46	157	162	90
71.6	27	0	5	16	113	125	226
71.7	21	0	4	10	98	118	261
71.8	13	0	3	6	50	103	337
71.9	7	0	3	5	21	107	369
72.0	5	0	3	2	10	99	393
72.1	2	0	1	1	8	44	456
72.2	1	0	0	1	6	19	485
72.3	1	0	0	0	2	8	501
72.4	0	0	0	0	1	5	506
72.5	0	0	0	0	0	0	512

the maximum possible value of Var and prove our claim as follows.

$$\begin{aligned} Var &= \frac{\sum_{i=1}^{n} {p_i}^2}{n} - \frac{1}{n^2} \\ &\leq \frac{\sum_{i=1}^{n} p_i}{n} - \frac{1}{n^2} \\ &= \frac{1}{n} - \frac{1}{n^2} \end{aligned}$$

Thus it is indeed justified to use the variance of the probability distribution Var to estimate the degree of bias of a fault model. The usefulness of this metric lies in the fact that it not only allows one to analytically evaluate fault models in terms of their bias but also makes it possible to compare two fault models with respect to their relative degree of bias.

A. The Fault Collision Probability

In order to get a faulty ciphertext in time or hardware redundant AES, the same fault f_i must occur in both the original and redundant computations. Let F_{org} and F_{red} be the random variables denoting the outcome of fault injections in the original and redundant rounds respectively. Since the fault injection in the original and redundant rounds are independent, we have $Pr[F_{org} = f_i, F_{red} = f_j] = p_i p_j$. We focus on the event where $F_{org} = F_{red}$. Let the probability of this event be denoted by $\tilde{p}$.

$$\tilde{p} = \sum_{i=1}^{n} Pr[F_{org} = f_i, F_{red} = f_i] = \sum_{i=1}^{n} {p_i}^2. \quad (1)$$

Evidently, this is also the probability of leakage of faulty ciphertexts. Our objective is to find if there is a correlation between the biased nature of the fault distribution and this probability of fault co-occurrence. Since the degree of bias of the fault model is quantified using the variance of the probability distribution as per our definition, the following relation is of interest and demonstrates that with increase in degree of bias, the fault collision probability increases.

$$\tilde{p} = nVar + \frac{1}{n} \quad (2)$$

Both time and hardware redundancy countermeasures, if naively implemented, would fail to detect the occurrence of a fault as long as the adversary could inject the same fault in both the original and redundant computations. The use of a biased fault model would make the probability of success fairly high for such an attack and would allow recovery of the key using practically feasible number of fault injections.

V. Description of the Attack

In this section, we describe the detailed procedure of the performed attacks on a time and hardware redundant versions of AES. The attack essentially extends DFIA [19] to two target

TABLE V: Notations used in the attack procedure

P	Plaintext
C	Fault-free ciphertext
q	A specific fault instance
Q	The total number of faulty ciphertexts obtained
F	The total number of fault injections(inclusive of all fault models)
C'_q	The faulty ciphertext under fault q
r	A round of AES
k	A key hypothesis
K	The correct key
$S^r{}_K$	The fault free cipher state in round r for key K
$S'^r{}_{k,q}$	A guess for the faulty cipher state in round r under fault q and key hypothesis k

rounds instead of just one, and also uses an additional distinguisher function - the Squared Euclidean Imbalance (SEI) to identify the correct key hypothesis. The attack procedure introduces the fault into either round 8 or round 9 of AES, and exploits the biased nature of the introduced fault to decipher the key. Please refer to Table V for the notations used for describing the attack procedure. Note that our fault model for the attack only comprises SBU, SBDBU, SBTBU and SBQBU (refer Table IIa), i.e, all the fault models are *single byte fault models*.

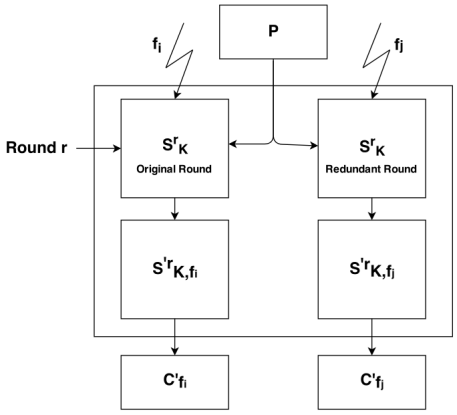


Fig. 3: Attack Steps

A. General Attack Procedure

We first present the general steps of the attack, irrespective of the round in which the fault is introduced. A more round-specific treatment of the attack is presented following the general discussion. The steps of the attack on the time and hardware redundant implementations of AES are also elucidated in Figure 3. Please not that here round r refers to the target round of the original AES and not the time or hardware redundant versions. This is done for the purpose of clarity in presenting a common description of the attack procedure.

Step 1: In this step the adversary induces faults f_i and f_j in both the normal and redundant computation of the target round r. However, the adversary can get the desired faulty ciphertext C'_{f_i} only if f_i and f_j are identical; otherwise the ciphertext is randomized. Note that a random ciphertext cannot distinguish between correct and incorrect key hypotheses and so, does not contribute to key hypothesis testing. This only increases the number of fault injections required to recover the key. We consider N_C to be the *number of non-random faulty ciphertexts* and N_F to be the *overall number of fault injections*.

Step 2: Once the adversary collects the value of faulty ciphertext C'_{f_i}, he can compute the value of faulty state $S'^r{}_{k,f_i}$ under key hypothesis k. He computes this value for every possible key hypothesis k.(Note that it is sufficient to hypothesize only those bytes of k that affect the faulty byte of $S'^r{}_{k,f_i}$ since our fault model allows only single byte faults). After doing this for several collected ciphertexts, the adversary uses a distinguisher to identify the correct key hypothesis.

Step-3: The adversary chooses the key hypothesis k that minimizes/maximizes the appropriate distinguisher function for the chosen fault model. A detailed description of the distinguisher functions is presented in V-B. If no satisfactory key guess can be made, N_C is to be increased and the test repeated. Note that in time or hardware redundant AES with suppression, the number of fault injections N_F is greater than N_C as not all fault injections yield a faulty ciphertext.

B. Distinguisher Functions

Distinguisher functions are used by the adversary to decide on the correct key byte(s) by selecting the key hypothesis that corresponds to the expected bias in the faulty state. For our attacks, we use two well known distinguisher functions - *Hamming Distance* [19] and *Squared Euclidean Imbalance* [18], [45]. Equations 3 and 4 describe these functions, with k as the key hypothesis and b as the affected byte of the AES state.

$$H(k) = \sum_{i=1}^{N_C} \sum_{j=1}^{i-1} HD(S'^r{}_{k,f_i}, S'^r{}_{k,f_j}) \tag{3}$$

$$S(k) = \sum_{i=1}^{N_C} \sum_{\delta=0}^{255} \left(\frac{\#\{b \mid S'^r{}_{k,f_i}[b] = \delta\}}{N_C} - \frac{1}{256}\right)^2 \tag{4}$$

C. The Attack on the Countermeasure Implementations of AES-128

We describe the fault attack procedure where the faults are introduced in rounds 8 and 9 of AES, and the choice of distinguisher function is made accordingly.

1) Attack on the 8th round:

Fault Location: The fault f_i is injected just after the ante-penultimate AddRoundKey operation of the AES, modifying a random byte b of $S^8{}_K$ [18]. The injection occurs in both the original and redundant rounds of computation.

Attack Procedure: Equation 6 summarizes the relation between the faulty ciphertext and the faulty state. The adversary can hypothesize on 4 bytes of K_{10} and

TABLE VI: Summary of the Attack Procedure

Fault Model	Target Round	Key Search Space	Distinguisher Used
Single Byte	Round 9	2^{12}	Hamming Distance
Single Byte	Round 8	2^{34}	Hamming Distance Squared Eucledian Imbalance

one byte of K_9 to get the corresponding states and then use the SEI distinguisher to identify the correct key hypothesis, because the Hamming Distance is found to require more faulty ciphertexts in this case to arrive at the key hypothesis.

Attack Complexity: The attack requires 2^{32} key hypotheses for recovering 4 bytes of the key [18], and a total of 4 such sets for recovering the entire key, leading to an overall requirement of $4 \times 2^{32} = 2^{34}$ hypotheses. Once again, both time and hardware redundancy demand that the actual number of attacks be greater than the required number of faulty ciphertexts.

$$S'^{9}{}_{K,f_i} = SB^{-1}(SR^{-1}(C'_{f_i} \oplus K_{10})) \tag{5}$$

$$S'^{8}{}_{K,f_i} = SB^{-1}(SR^{-1}((MC^{-1}((SB^{-1}(SR^{-1}(C'_{f_i} \oplus K_{10})) \oplus K_9)))) \tag{6}$$

2) Attack on the 9th round:

Fault Location: The fault f_i is injected just after the penultimate AddRoundKey operation of the AES, modifying a random byte b of $S^9{}_K$. The injection must occur in both the original and redundant rounds of computation.

Attack Procedure: Since the last round involves no MixColumns operation, we have Equation 7. The adversary collects several faulty ciphertexts $C'_1, \ldots, C'_N$ on the same P and hypothesizes on one byte of the key to obtain 256 guesses of the faulty state $S'^{9}{}_{k,f_i}$ - one for each key hypothesis k. This is followed by the computation of $H(k)$ to identify the correct key hypothesis. It should be noted that the SEI distinguisher is useless in this context, as the distance to the uniform distribution will be the same for each hypothesis [18].

Attack Complexity: The attack requires 256 key hypotheses for recovering each byte of the key.

$$S'^{9}{}_{K,f_i} = SB^{-1}(SR^{-1}(C'_{f_i} \oplus K_{10})) \tag{7}$$

Table VI summarizers the overall attack procedure in brief.

VI. Attack Simulations

In this section, we present some simulation results of our proposed attacks on software implementations of the time and hardware redundancy countermeasures for AES-128. The simulation studies are divided into two major halves. In the first half, we assume the same fault for the original and redundant rounds so that each fault injection gives us a faulty ciphertext, i.e., N_C is same as N_F . Our aim here is to estimate the number of faulty ciphertexts required to recover the full key under different fault models. In the second half, we vary the probability distribution for each fault model to confirm the correlation of the bias with the fault collision probability, as described by Equation 2. We quantify the bias of the fault model using the variance of the fault probability distribution, and the fault collision probability by the number of fault injections required per faulty ciphertext.

TABLE VII: Number Of Faulty Ciphertexts Required To Guess the Entire Key With 99% Probability

Round	Fault Model	N_C
8	SBU	320-340
	SBDBU	580-600
	SBTBU	1000-1040
	SBQBU	1900-2000
9	SBU	288-320
	SBDBU	608-640
	SBTBU	832-880
	SBQBU	1360-1440

A. Simulation: Part-1

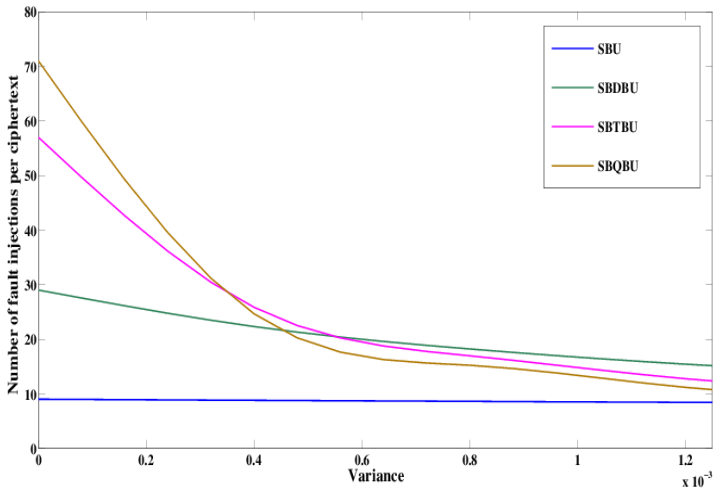
In this part of the simulation, we assume identical faults in both the original and redundant computation rounds and aim to estimate the average number of faulty ciphertexts required to recover the entire key. Note that since the actual attack procedure is independent of the countermeasure scheme being targeted (time or hardware redundancy), the simulation results are presented for a general attack on either countermeasure scheme.

In the simulation, a byte of the state at the desired attack point is chosen at random and then fault is introduced into a certain number of bits belonging to that byte, varying from 1 to 4. Note that these bits are also chosen at random. We simulate the attacks in rounds 8 and 9 respectively. In each case, the appropriate distinguisher function is used to choose the key hypothesis. Table VII summarizes the number of faulty ciphertexts required for each fault model to guess the entire 128-bit key with 99% accuracy for the attacks on rounds 8 and 9.

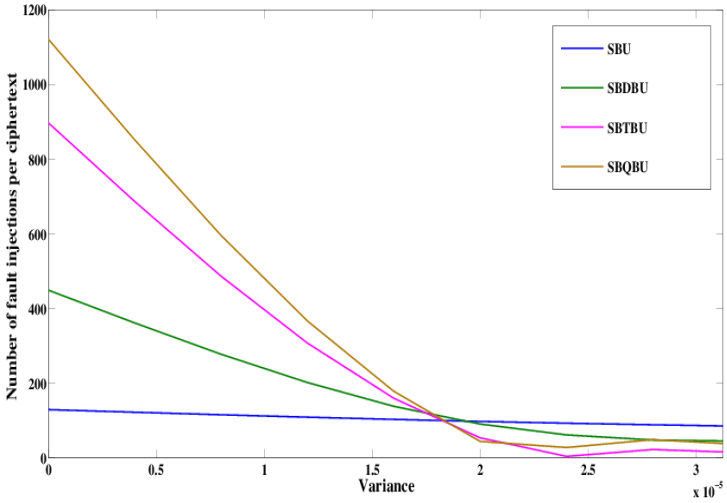
B. Simulation: Part-2

In the second half of the simulation, we varied the degree of bias for each fault model by controlling the variance of the fault probability distribution for each model and observed the average number of fault injections required per faulty ciphertext, computed over a set of 100 ciphertexts. In this experiment, the assumption was that the countermeasure suppresses the ciphertext on fault injection. Our experiment considered two distinct scenarios, in which the adversary has perfect and no control respectively over the target byte in which the fault is to be induced. For the first scenario, the fault was injected only in a fixed target byte, while in second scenario, the target byte was randomly chosen. In either scenario, we simulated the fault probability variance using a normal distribution with mean $1/n$ and the desired variance, where n is the total number of faults achievable under the corresponding fault model. Figures 4a and 4b summarize the simulation observations over a wide range of fault distribution

Fig. 4: Number of Fault Attacks per Faulty Ciphertext vs Variance of Fault Probability Distribution



(a) Adversary has perfect control over target byte



(b) Adversary has no control over target byte

variances, in both scenarios. These observations show that with increase in bias of the fault distribution, the number of fault injections that are required per faulty ciphertext drops rapidly. This in turn confirms our hypothesis that increasing the bias of the fault model enhances that fault collision probability.

VII. Does Masking Protect against Biased Fault Attacks?

Biased fault attacks such as the DFIA tend to combine principles of DPA with active fault analysis to identify the secret key. In literature, one of the most popular and effective countermeasures against side channel analysis techniques such as the DPA is masking. In this section, we explore the security of masked implementations of the time and hardware redundancy countermeasures against biased fault attacks. Masking involves concealing any intermediate value related with the key with a mask m, where m is randomly chosen and varies from encryption to encryption. The intermediate value v is transformed into its randomized counterpart $v_m = v * m$ such that the value of m is not known to the adversary. A special subset of masking is the *boolean masking* in which the operation $*$ is the XOR operation. In the following discussion, we focus on countermeasures that incorporate boolean masking at the algorithmic level. Masking is a very popular countermeasure technique against power analysis, since masking with a random value is assumed to destroy any correlation that the power consumption of the device has with the actual key-dependent data.

A. The Masked Countermeasure Implementation

Figure 5 schemetically describes the basic boolean masking mechanism, which may then be applied to both the time and hardware redundancy countermeasures. The key-dependent input to each round is masked with a random mask value. In order to conceal the power leakage from both registers R_0 and R_1, the content of both these registers needs to be masked. Let ${S^r}_K$ be the unmasked fault free cipher state after r using key K, and let m_0^r and m_1^r be the mask values for the original

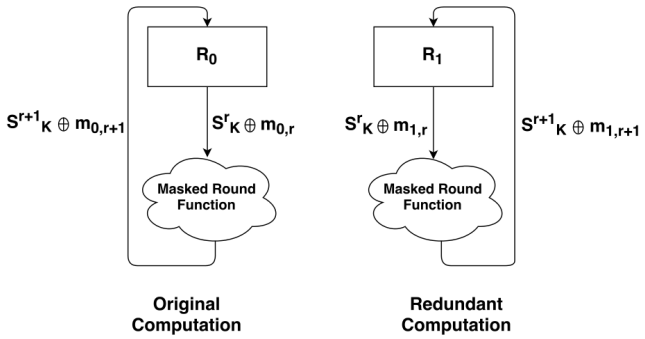


Fig. 5: A Schemetic for the Masked Countermeasure

and redundant computations after round r, respectively. By round r, we of course refer to the round r of the original AES algorithm. We denote the corresponding masked states for R_0 and R_1 by $\hat{S}^r_{0,K}$ and $\hat{S}^r_{1,K}$ respectively. Since boolean masking is used, we have $\hat{S}^r_{0,K} = {S^r}_K \oplus m^r_0$ and $\hat{S}^r_{1,K} = {S^r}_K \oplus m^r_1$. Thus, for fault free computations, the following invariance must be satisfied for each round r.

$$\hat{S}^r_{0,K} \oplus m^r_0 = \hat{S}^r_{1,K} \oplus m^r_1 \tag{8}$$

In fact, this is the very condition that is used by the countermeasure to detect the occurrence of a fault. We note that the very definition of masking demands that the generation of the mask value corresponding to each round be independent of the register content in that round.

B. Biased Fault Attack on the Masked Countermeasure Implementation

We now demonstrate that masking fails to thwart biased fault attacks on the countermeasure schemes. We first point out that our proposed biased fault attack on the AES countermeasures has two major phases:

1) Bypassing the detection step after fault injection by injecting identical faults in the original and redundant rounds
2) Distinguish th correct key hypothesis by exploiting the underlying bias of the faulty cipher state

Suppose that the adversary introduces faults f_0 and f_1 in the registers R_0 and R_1 after the original and redundant computations corresponding to round r of the original AES algorithm. The algorithm fails to detect the fault if the invariance 8 holds even after the fault injection. We now investigate the scenario when this may happen.

$$\begin{aligned} &(\hat{S}^r_{0,K} \oplus f_0) \oplus m^r_0 = (\hat{S}^r_{1,K} \oplus f_1) \oplus m^r_1 \\ &\Rightarrow (\hat{S}^r_{0,K} \oplus m^r_0) \oplus f_0 = (\hat{S}^r_{1,K} \oplus m^r_1) \oplus f_1 \\ &\Rightarrow {S^r}_K \oplus f_0 = {S^r}_K \oplus f_1 \\ &\Rightarrow f_0 = f_1 \end{aligned} \tag{9}$$

Thus, even in the presence of masking, the adversary could still inject the same fault in the original and redundant computations of the masked scheme, and bypass the detection step. Moreover, the biased nature of the underlying fault model implies that the adversary could still inject the same fault in both the original and redundant rounds with a high probability. Thus, masking has effectively no impact on the probability that the adversary successfully bypasses fault detection.

The other half of the attack is also valid on the masked implementation of the countermeasure schemes. As in our previous example, suppose that the adversary introduces the fault f_0 in the original computation corresponding to round r, and has bypassed the detection step by introducing the same fault in the redundant computation as well. The corresponding faulty cipher state is given by $(\hat{S}^r_{0,K} \oplus f_0) = (\hat{S}^r_K \oplus f) \oplus m^r_0$. Since m^r_0 is computed independent of the cipher state R_0 after each round r, the introduction of the fault does not alter the mask values generated in the subsequent rounds of the algorithm. Correspondingly, the computation proceeds as if the correct value after round r was $\hat{S}^r_K \oplus f$ instead of $\hat{S}^r_K$. This in turn implies that a correct key hypothesis will yield the biased faulty state, while all other key hypotheses will yield random states, and the adversary can use an appropriate distinguisher function (such as HW or SEI) to identify the correct hypothesis.

Remark: A major reason as to why masking fails in countering biased fault attacks is that it *does not transform the fault space for the original and redundant rounds*. This allows the adversary to exploit the biased nature of the underlying model and introduce the same fault in the original and redundant rounds. In the next section, we explore a countermeasure scheme that tries to thwart biased fault attacks by transforming the fault space. Essentially, the idea is to ensure that injecting the same random fault f in the registers R_0 and R_1 would not amount to an equivalent fault injection in the original and redundant computation rounds. This reduces the success probability of the attack considerably.

VIII. Countering Biased Faults : Transfomation of Fault Space

In this section we present a countermeasure strategy based on fault space transformation to prevent biased fault attacks on the time and hardware redundancy countermeasure schemes. The basic idea is to prevent the adversary from being able to exploit the underlying bias in the fault model to inject the same fault in both the state registers R_0 and R_1. One such strategy is to have different encodings for the state registers R_0 and R_1 in each state of the FSM. This would automatically imply that injecting the same random fault f_i in the registers R_0 and R_1 would not amount to an equivalent fault injection in the original and redundant computation rounds. It is important to note that the presence of the detection step necessitates the existence of a bijective mapping between the state spaces for the registers; otherwise the comparison of the register values is not possible. This in turn implies that there is an equivalent bijective mapping between the fault spaces $\mathcal{F}_0$ and $\mathcal{F}_1$ for the two registers, that is for each fault $f_i \in \mathcal{F}_0$ there is an equivalent fault $f_j \in \mathcal{F}_1$ such that $f_i \equiv f_j$. Thus, for the first phase of the attack on the updated FSM, the adversary must inject equivalent faults in the registers R_0 and R_1. We now present this idea formally.

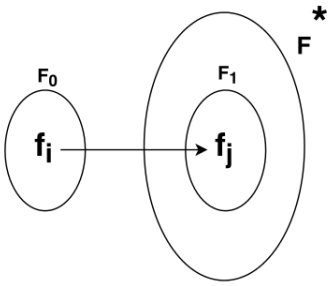


Fig. 6: Fault Space Transformation

A. The Motivation: Transforming the Fault Space

As discussed in section III the adversary must use a precise cum biased fault model in order to be able to induce the same fault in both the original and redundant computations corresponding to a single round of AES. We stress the importance of fault model precision here to improve the success probability of the attack. If the fault space is too large, the probability of fault collision reduces in practical set-ups. To validate this, we performed experimental studies on real life implementations of time and hardware redundancies on a SPARTAN-3A FPGA. The faults were introduced using set-up violations achieved via clock glitches at various frequencies. Figure 7 demonstrates that as the fault precision reduces (that is the fault spreads across multiple bytes), the probability of fault collision reduces. In fact, beyond 16 affected bits, it becomes practically infeasible to even achieve a single occurrence of fault collision. The mathematical intuition behind this could be explained as follows. If the fault space $\mathcal{F}$ consists of a total of n possible faults, the maximum variance of the fault probability distribution followed by the corresponding fault model is given by $Var_{max} = \frac{1}{n} - \frac{1}{n^2}$ which is a monotonically decreasing function for $n > 2$. Since the variance is also a measure for the bias of the fault model, it is amply clear that *less precise fault models are inherently less biased than precise fault models*.

The above discussion thus leads to the conclusion that multi-byte faults cannot be used practically for attacking even naive time and hardware redundancy countermeasure implementations. Single byte fault models are the adversary's best options if she has to have a feasible chance of extracting the key. In our proposed countermeasure scheme against such biased fault attacks, we aim to exploit this fact by ensuring that *single byte faults in R_0 are mapped to equivalent multiple byte faults in R_1*. Thus a smaller fault space $\mathcal{F}_0$ is mapped to an equivalent fault space $\mathcal{F}_1$, which in turn is a subspace of a much larger fault space $\mathcal{F}^*$, as demonstrated in Figure 6. This significantly lowers the probability that the adversary can inject equivalent faults in both R_0 and R_1 in the augmented state machine setting. It is easy to see that the reverse scenario in terms of fault space mapping is not possible, that is, a larger fault space can never be mapped to a smaller or a more precise one. Since the mapping between the fault spaces is a bijection, they will always have the same cardinality. Moreover, using low cost fault injection techniques such as clock glitches, it is practically infeasible for the adversary to target a fixed subset of multiple byte faults on R_0 that are mapped to single byte faults in R_1. Thus transformation of the fault space significantly reduces the success probability of random fault injections. We now prove this formally in the following discussion for both uniform and biased fault model settings. We first state an important assumption that is used in the formal analysis.

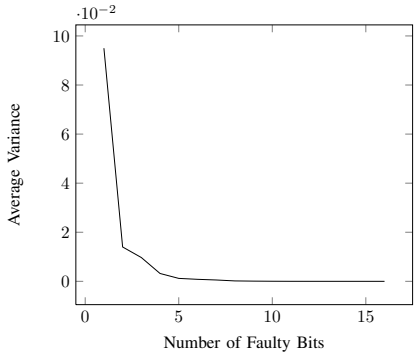
Assumption 1**:** The adversary can guarantee the occurrence of a fault in the larger fault space $\mathcal{F}^*$ but not in the subspace $\mathcal{F}_1$.

This is a fairly reasonable assumption and we present a small example here to justify it. Suppose the chosen fault space mapping maps all single byte faults to a specific subspace of four byte faults. Then the adversary can use a specific clock glitch frequency to inject four byte faults in the redundant computation with a reasonably high probability. However, using the same fault injection technique, she cannot in any way enhance the probability of occurrence of specifically those four byte faults that are part of the smaller subspace under this mapping. These faults would still have the same probability of occurrence as they had in the larger fault space comprising of all four byte faults.

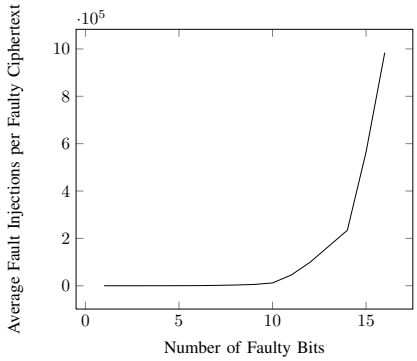
Let $W : (0,1)^{128} \rightarrow (0,1)^{128}$ be a bijective mapping such that $R_1 = W(R_0)$ under fault free operation of both the time and hardware redundancy countermeasures. Under uniform fault assumption, each fault $f_i \in \mathcal{F}_0$ has probability of occurrence $\frac{1}{|\mathcal{F}_0|}$ and each fault $f_j \in \mathcal{F}_1$ has probability of occurrence $\frac{1}{|\mathcal{F}^*|}$, as per the aforementioned assumption. Hence, probability of fault collision between two random fault injections $\hat{f}_0$ and $\hat{f}_1$ is given by $\tilde{p} = \sum_{i=1}^{|\mathcal{F}_0|} Pr[\hat{f}_0 = f_i, \hat{f}_1 = W(f_i)] = \frac{1}{|\mathcal{F}^*|}$ which is a very small value for a large fault space $\mathcal{F}^*$. *Note that we set* $pr[\hat{f}_1 = W(f_i)] = \frac{1}{|\mathcal{F}^*|}$. This is in accordance with the assumption stated above.

The analysis for the biased scenario is much more interesting. We may assume that the probability distribution of faults in the fault spaces $\mathcal{F}_0$ and $\mathcal{F}^*$ are independent and have no correlation. This is a reasonable assumption because in a practical set-up the distribution patterns of faults under different fault models such as single byte and multi byte fault models are in general not correlated. Since $\mathcal{F}_2$ is a subset of $\mathcal{F}^*$ determined by the transformation W over which the adversary has no control, it is safe to assume that the probability distributions of faults in the fault spaces $\mathcal{F}_0$ and $\mathcal{F}_1$ are also independent. Also, the choice of the transformation function W is not known under the adversary's control. Any chosen mapping W maps $\mathcal{F}_0$ to a subset $\mathcal{F}_1$ of $\mathcal{F}^*$ such that $|\mathcal{F}_1| = |\mathcal{F}_0|$. There are $\binom{|\mathcal{F}^*|}{|\mathcal{F}_0|}$ such subspaces. A particular fault $f_j \in \mathcal{F}^*$ occurs in $\binom{|\mathcal{F}^*|-1}{|\mathcal{F}_1|-1}$ of thee subspaces. Thus, given a random fault $f_i \in \mathcal{F}_0$ and a random fault $f_j \in \mathcal{F}^*$, the expectation of $Pr[f_j = W(f_i)])$ over all possible choices of W (assuming the adversary has no control over W)is given

Fig. 7: Effect of Fault Precision



(a) Variance v/s Number of Faulty Bits



(b) Attack Efficiency v/s Number of Faulty Bits

as follows:

$$\begin{aligned}\mathbb{E}(Pr[f_j = W(f_i)]) &= \frac{\binom{|\mathcal{F}^*|-1}{|\mathcal{F}_0|-1}}{\binom{|\mathcal{F}^*|}{|\mathcal{F}_0|}|\mathcal{F}_0|} \\ &= \frac{1}{|\mathcal{F}^*|}\end{aligned} \tag{10}$$

Let p_j be the probability of occurrence of the fault $f_j \in \mathcal{F}^*$. Given the adversary has perfect knowledge of first fault injection $\hat{f}_0 = f_i \in \mathcal{F}_0$, and using Assumption 1, we have the following expected probability of fault collision.

$$\begin{aligned}\mathbb{E}(Pr[\hat{f}_1 = W(f_i)]) &= \sum_{j=1}^{|\mathcal{F}^*|} \mathbb{E}(Pr[f_j = W(f_i)]Pr[\hat{f}_1 = f_j]) \\ &= \sum_{j=1}^{|\mathcal{F}^*|} \mathbb{E}(Pr[f_j = W(f_i)])p_j \\ &= \frac{1}{|\mathcal{F}^*|}\sum_{j=1}^{|\mathcal{F}^*|} p_j \\ &= \frac{1}{|\mathcal{F}^*|}\end{aligned} \tag{11}$$

Finally, the expected probability of fault collision $\tilde{p}$ between two random fault injections $\hat{f}_0$ and $\hat{f}_1$ in a biased set up is given by the following equation.

$$\begin{aligned}\mathbb{E}(\tilde{p}) &= \mathbb{E}(\sum_{i=1}^{|\mathcal{F}_0|} Pr[\hat{f}_0 = f_i, \hat{f}_1 = W(f_i)]) \\ &= \sum_{i=1}^{|\mathcal{F}_0|} \mathbb{E}(Pr[\hat{f}_0 = f_i, \hat{f}_1 = W(f_i)]) \\ &= \sum_{i=1}^{|\mathcal{F}_0|} \mathbb{E}(Pr[\hat{f}_0 = f_i]Pr[\hat{f}_1 = W(f_i)]) \\ &= \sum_{i=1}^{|\mathcal{F}_0|} \mathbb{E}(Pr[\hat{f}_0 = f_i])\mathbb{E}(Pr[\hat{f}_1 = W(f_i)]) \\ &= \sum_{i=1}^{|\mathcal{F}_0|} \mathbb{E}(Pr[\hat{f}_0 = f_i])(\frac{1}{|\mathcal{F}^*|}) \\ &= \frac{1}{|\mathcal{F}^*|}\sum_{i=1}^{|\mathcal{F}_0|} \mathbb{E}(Pr[\hat{f}_0 = f_i]) \\ &= \frac{1}{|\mathcal{F}^*|}\mathbb{E}(\sum_{i=1}^{|\mathcal{F}_0|} Pr[\hat{f}_0 = f_i]) \\ &= \frac{1}{|\mathcal{F}^*|}\end{aligned} \tag{12}$$

The fact that the two probability distributions are independent is used in line 3 of the derivation. Thus, even though the fault models are individually biased, the fact they are mutually independent causes the expected collision probability to remain the same as in the unbiased scenario. In a real life attack scenario, the chosen transformation W will be known to the attacker. However, the mathematical formulation tells us that there exists such transformations where the probability of fault collision is even worse that the unbiased scenario. Thus, to summarize, the chosen transformation must satisfy the following condition:

1) The fault spaces $\mathcal{F}_0$ and $\mathcal{F}_1$ should have highly uncorrelated probability distributions.
2) The expected probability of fault collisions should be low. Intuitively, in a biased fault model scenario, this implies that the most probable faults should not get mapped to the most probable faults in the larger fault space $\mathcal{F}^*$.

We note that both of these conditions are expected to be satisfied in a practical scenario for most choices of W. Thus, changing the state encoding to transform the chosen adversarial fault space $\mathcal{F}_0$ to a large target fault space $\mathcal{F}^*$ would reduce the success probability of the adversary even in a biased scenario. Our next step is to formally present the augmented FSM framework that incorporates this countermeasure scheme. We also discuss good choices of transformation functions that help achieve our desired goal of reducing the fault collision probability.

Figures 8a and 8b represent the modified time and hardware redundancy countermeasure schemes that incorporate the state encoding transformation. It is easy to see the equivalence between the fault classes for the original and redundant computations with this modified scheme. Consider a fault free state of computation during an intermediate round of the computation. Let the content of the register R_0 be x. The content of register R_1 would therefore be $W(x)$. Now suppose the adversary injects equivalent faults f_0 and f_1 in the registers R_0 and R_1 respectively. We now have the following relationship between the faults f_0 and f_1.

$$\begin{aligned} x \oplus f_0 \quad &= W^{-1}(W(x) \oplus f_1) \\ \Rightarrow W(x) \oplus f_1 &= W(x) \oplus W(f_0) \\ \Rightarrow f_1 &= W(f_0) \end{aligned} \tag{13}$$

Thus the fault space has been transformed under the mapping W.

B. Choice of the Transformation Function - Using MDS Matrices

We now look at a possible strategy for designing the transformation function W that would map a restricted fault space $\mathcal{F}_0$ for the register R_0 to a subspace $\mathcal{F}_1$ of a much larger fault space $\mathcal{F}^*$ for the state register R_1. We propose the use of *Maximum Distance Separable* (MDS) matrices [27] for W. An MDS matrix is a matrix representing a function with special diffusion properties and has many useful applications in cryptography, especially in designing multipermutations to prevent cryptanalysis [52]. Formally, an $m_2 \times m_1$ matrix A over a finite field $\mathbb{K}$ is an MDS matrix if it is the transformation matrix of a linear transformation $f(x) = Ax$ from $\mathbb{K}^{m_1} to \mathbb{K}^{m_2}$ such that no two different $(m_1 + m_2)$ tuples of the form $(x, f(x))$ coincide in m_1 or more components. Equivalently, the set of all $(m_1 + m_2)$ tuples $(x, f(x))$ is an MDS code,that is, a linear code of dimension m_1, length $m_1 + m_2$ and minimal distance $m_2 + 1$, that reaches the *singleton bound*. MDS matrices are used in a number of block ciphers such as AES and Twofish [48] as well as state-of-the art lightweight ciphers. The property of the MDS matrices that is most appealing in the context of our preceding discussion on fault space transformation, is that they provide *perfect diffusion* [27]. *For an MDS mapping from K^{m_1} to K^{m_2} changing t components of the input changes at least $m_1 - t + 1$ components of the output.*

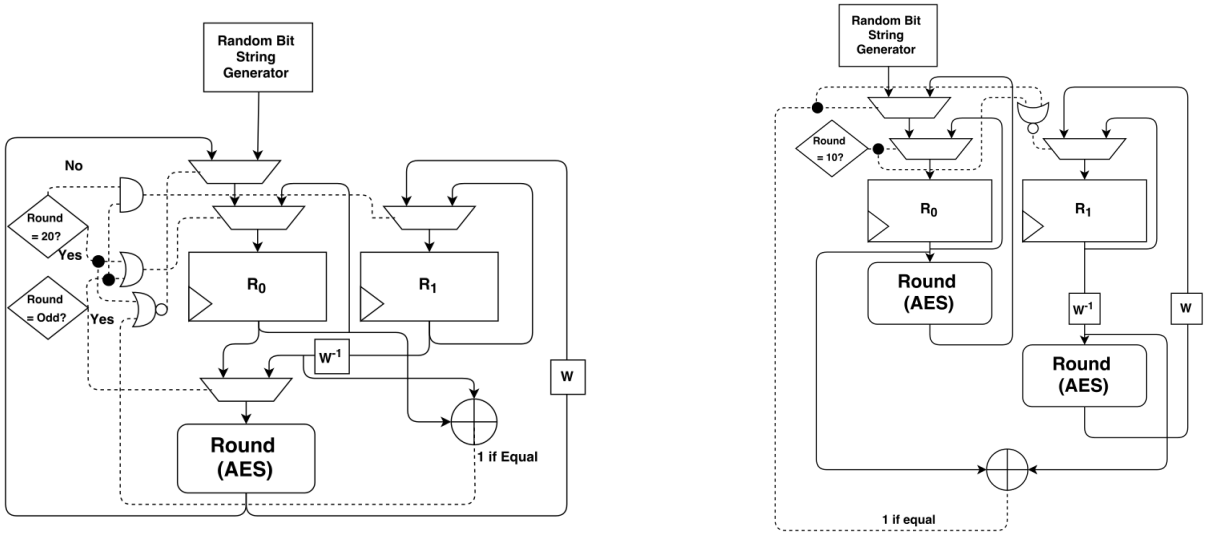
We now investigate the usefulness of the diffusion property of MDS matrices with respect to our fault space transformation motive. We refer in the following discussion to the modified countermeasure schemes in Figures 8a and 8b respectively. Suppose that the linear transformation W is a $m_2 \times m_1$ MDS mapping over a field $\mathbb{K}$ from $\mathbb{K}^{m_1} to \mathbb{K}^{m_2}$. Let the adversary injects a t byte fault f_0 in the register R_0, and let f_1 be the corresponding fault to be injected in the register R_1 so that the countermeasure fails to detect the fault injection. By equation 13, $f_1 = W(f_0)$. By the MDS diffusion property, the t byte fault f_0 is mapped to an at least a $m_2 - t + 1$ byte fault f_1. For the special case of a single byte fault, the transformed fault space comprises of faults that affect at least m_2 bytes of the output. Thus the precision of the output fault space $\mathcal{F}_1$ is much lower, making it very difficult to have the random fault generator $Rand_{\mathcal{F}^*}$ generate the desired fault with even reasonable probability. Note that we use $Rand_{\mathcal{F}^*}$ instead of $Rand_{\mathcal{F}_1}$ because the adversary cannot specifically generate the faults in $\mathcal{F}_1$ in a practical scenario. Thus using MDS matrices for the fault space mapping significantly reduces the attack probability on the time and hardware redundancy countermeasures. The question is which MDS matrix to choose. A possible choice is to use the Rjindael MDS matrix used in the MixColumns operation for AES-128.

1) Using the Rjindael MixColumns: The Rjindael MixColumns operation used in AES consists of multiplying a input vector of length 4 by a 4×4 MDS matrix in the finite field $\mathbf{GF}(2^8)$. The matrix is presented below for reference. Each column is treated as a polynomial over $\mathbf{GF}(2^8)$ and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 0x03x^3 + x^2 + x + 0x02$.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

The MixColumns operation takes four bytes as input and produces 4 bytes as output, where *each input byte affects all four output bytes*. This implies that if a single byte of the input vector is changed, all 4 bytes of the output are affected. From the point of view of fault injections, if the adversary were to inject a single byte fault in the input vector, the MixColumns would diffuse the fault across all four bytes of the output vector. Thus essentially, the fault model transformation takes place from a single byte fault to a four byte fault. Since fault models beyond single byte faults are of no practical significance to the adversary for attacking the time and hardware redundancy countermeasures due to lack of precision, it is sufficient to consider the diffusion property of MixColumns for single byte faults. Thus, in accordance with our previous discussion $\mathcal{F}_0$ is the set of all single byte faults, $\mathcal{F}^*$ is the set of all 4 byte faults and $\mathcal{F}_1$ is the image of $\mathcal{F}_0$ in $\mathcal{F}^*$ under the MixColumns mapping. Even if the adversary were to know precisely which byte of R_0 was affected by the injected fault (and the corresponding bytes of R_1 that would need to be affected for an equivalent fault), the size of $\mathcal{F}^*$ is too huge for the adversary to be able to precisely introduce only those faults that are in $\mathcal{F}_1$. This is immediately apparent from the ratio of the sizes of the fault spaces $\mathcal{F}_1$ and $\mathcal{F}^*$ given by $\frac{255}{2^32-4.2^24-6.2^16-4.2^8-1} \approx 6.032 \times 10^{-8}$. Moreover, it has

Fig. 8: The Augmented Concurrent Error Detection Techniques



(a) Time Redundancy

(b) Hardware Redundancy

been practically observed in our experiments that the variance Var of the probability distribution of the fault model reduces as the fault precision reduces, as demonstrated in Figure 7b. Thus even if the fault model is biased, the size of the fault space plays a major role in making fault collisions practically infeasible.

C. A Common Countermeasure against both Classical and Biased Fault Attacks

Current literature focuses on two major varieties of fault attacks against block ciphers - classical fault attacks such as the DFA that assume uniform fault models, and more recent fault attacks such as the DFIA and the FDE that use biased fault models. Classical DFA is successfully thwarted by concurrent error detection (CED) techniques, that use redundancies to try and detect the fault. As demonstrated in Section V, CEDs are weakened in the presence of biased fault models due to the enhanced fault collision probability. Our proposed countermeasure technique, on the other hand, provides a way to modify CEDs to successfully thwarts both classical as well as biased fault attacks. The argument for this claim is fairly straight forward. In classical DFA, the adversary requires at least one pair of faulty and fault-free ciphertexts. There is no assumption made about the underlying bias of the fault model; classical concurrent error detection techniques successfully thwart DFA under the assumption of uniform fault models. Our proposed countermeasure scheme relaxes this assumption by allowing the adversarial fault model to be biased. Since the fault space is transformed for the redundant computation in the CED, the underlying bias of the original fault model fails to ensure that the adversary is able to introduce equivalent faults in the original and redundant rounds with high probability. The same argument applies for biased fault attacks such as the DFIA and FDE as well. In both DFIA and FDE, the adversary attempts to exploit the underlying bias of the fault model to try and recover the key. However, as already discussed, our proposed countermeasure nullifies the effect of the bias in the fault model by fault space transformation, and hence thwarts both these attack schemes. This scheme, to the best of our knowledge, is the first to counter any fault attack that targets the cipher state, irrespective of the underlying fault model. Moreover, since no other fault attack technique that formidably compromises the security of block ciphers by targeting the cipher state has been reported in literature, this countermeasure technique seems to provide comprehensive coverage against state of the art fault attacks. It is important to note that we focus here precisely on those fault attacks that involve injecting faults in the cipher state. Faults that change the execution sequence via techniques such as instruction skips are outside the scope of this work, and are hence not included in this discussion.

D. Impact of the additional MixColumns operation on the classical security of the cipher

In the proposed countermeasure scheme, the redundant computation computing to each round of AES has been augmented by an additional MixColumns operation. It is natural to question if this disturbs the classical security of the cipher against linear and differential cryptanalysis. However, it is important to note that the *redundant round output is never made visible to the adversary and is used only for internally detecting the presence of a fault*. The final output is the output of the original computation corresponding to each round, which has the same SPN structure as AES itself. The addition of the MixColumns operation in the redundant round thus provides no differential or linear trail that the adversary could potentially exploit to gain information about the key. Consequently, the security of the cipher against classical cryptanalysis is not compromised.

IX. Experimental Results

In this section, we present experimental results to validate the security of our proposed countermeasure scheme. All

experiments have been conducted on a Spartan 3A FPGA, on a SASEBO GII platform. The implementation is a register-transfer level Verilog definition of AES with each round implemented using a original and a redundant round. For the time redundant implementation, each round is repeated twice, and thus, a total of 20 rounds are needed for the computation. The hardware redundant implementation, on the other hand, duplicates the original AES hardware, and runs the two in parallel for 10 rounds. The countermeasure implementations have already been described in Figures 8a and 8b. The plaintext and key are randomly chosen 128 bit values. If the output of original and redundant round of computations is different, i.e., if a fault is detected by the countermeasure, the state register is immediately randomized.

The experimental section is divided into two broad parts. The first part demonstrates the proposed fault attack is indeed feasible on time and hardware redundant implementations of AES. The second part shows the effect of introducing the fault space transformation on the number of fault injections required per faulty ciphertext.

A. Practical Demonstration of the Attack

In this section we present the practical results of our proposed fault attack in Section V. The attack procedure is identical for both the time and hardware redundancy countermeasure schemes.

1) Attack on Round-8:: A total of 4 bytes of the AES state were affected one by one after the anti-penultimate AddRoundKey operation, since each byte of the faulty state can be guessed by hypothesizing 4 bytes of the Round 10 key K_{10}. Again, the external clk_{fast} was increased gradually from 125.3 MHz to 126.4 MHz to achieve the for different fault models. Once sufficient number of faulty ciphertexts had been collected for each of the 4 bytes, the entire key was deciphered using the appropriate Squared Euclidean Imbalance computation for each byte for all the key hypotheses.

2) Attack on Round-9:: Each of the 16 bytes of the AES state were affected one by one after the penultimate AddRoundKey operation to guess the 16 bytes of the Round 10 key K_{10}. The external clk_{fast} was increased gradually from 125.3 MHz to 126.4 MHz to achieve the four different fault models. Once sufficient number of faulty ciphertexts had been collected for each byte, the entire key was deciphered using the appropriate Hamming Weight computation for each byte for all the key hypotheses.

B. Fault Location Precision

We performed 2 types of attacks - Type-1 in which the adversary has perfect control over the byte in which the fault is to be introduced and Type-2 in which the adversary only knows that the fault injected is a single byte fault without any knowledge of the byte affected. The second type of experiments demands much lesser control over the actual fault injection, but is weaker as observed in the experimental results, and demands a significantly larger number of fault injections. For the first type, only the target byte should be affected by the clock glitch while in the second, the entire AES state should be subjected to the clock glitch. We describe the set up changes to be made for either scenario in greater detail. Suppose that the adversary wishes to affect only byte w of the AES state. She can achieve this precision by modifying the fault injection set up slightly to allow clk_{fast} to affect only byte w while all other bytes are driven by clk_{slow}. This ensures that in the event of a clock glitch, only byte w is affected. This is illustrated in Figure 9. Type-2 is the normal fault injection scenario where all bytes are allowed to be affected by clk_{fast}.

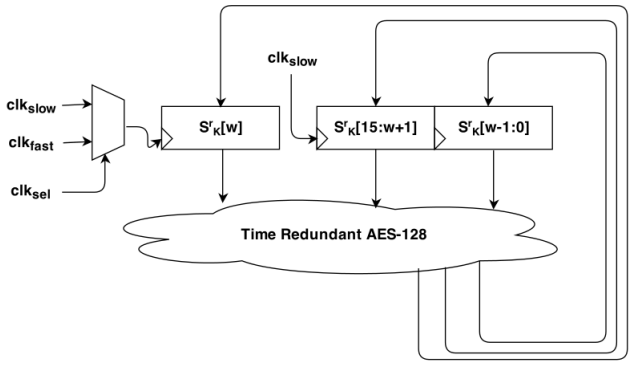


Fig. 9: Modified Fault Injection Setup: adversary has control over affected byte

For each scenario, we repeated the experiment 100 times, with the same randomly chosen key and the randomly chosen plaintext and took the average values for the number of faulty ciphertexts as well as the number of fault injections required to recover the key as well. Tables VIIIand IX demonstrates the number of faulty ciphertexts and the number of fault attacks required for recovering the entire key under the attack on rounds 8 and 9, for both the scenarios where the adversary has and does not have control over the fault location. The variance of fault distribution presented for each model was experimentally observed. In both tables, we compare the experimentally required number of fault injections with the expected number of fault injections according to the simulation. It is evident that the experimentally obtained data corroborates the simulation results very well, thus confirming the hypothesis that with more bias, our proposed fault attack can break the time redundancy countermeasure with very less number of fault injections, as compared to unbiased faults.

C. The Fault Space Transformation

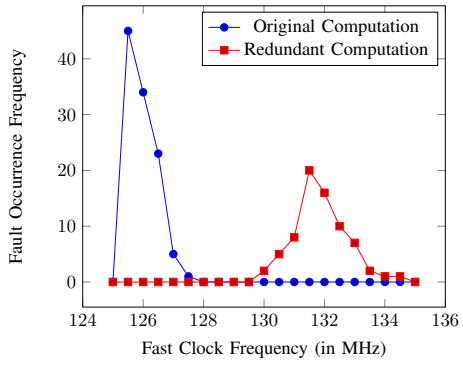
In this section, we focus on the effect of introducing the MixColumns operation in the redundant computation for each round, on the efficiency of the biased fault attack. The results are presented in Figures 10 and 11 for 512 samples obtained at different frequencies for both the time and hardware implementations of AES-128. Quite evidently, for each fault model, the transformation not only causes the frequency ranges for equivalent faults in the original and redundant computations to be drastically different, but also affects the occurrence probability of various faults to be different. The phenomenon could be easily explained as follows. The transformation of the fault space due to the additional MixColumns operation in the redundant computation causes all single byte faults in the

TABLE VIII: Experimental Results : Attack on Time Redundancy

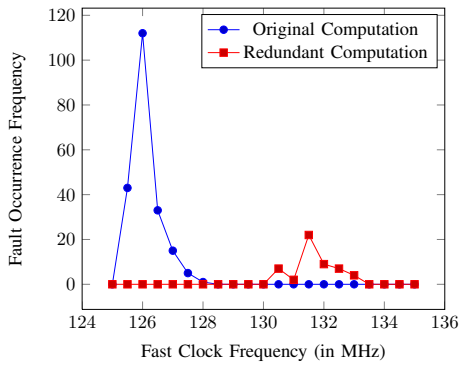
Round	Fault Model	Fault Variance		N_C	N_F(simulation)		N_F(experimental)	
		Type-1	Type-2		Type-1	Type-2	Type-1	Type-2
8	SBU	9.5×10^{-2}	3.6×10^{-3}	304.75	340.48	647.52	387.67	687.91
	SBDBU	1.4×10^{-2}	9.2×10^{-4}	625.12	1456.25	1506.25	1448.45	1652.30
	SBTBU	9.7×10^{-3}	4.9×10^{-4}	1020.49	1815.60	2315.40	1974.86	2395.83
	SBQBU	3.2×10^{-3}	5.9×10^{-5}	1878.55	7868.82	28038.54	8003.14	30201.41
9	SBU	9.2×10^{-2}	3.5×10^{-3}	304.24	385.88	603.11	387.98	632.71
	SBDBU	8.8×10^{-2}	7.9×10^{-4}	624.65	641.18	1487.36	647.82	1556.69
	SBTBU	8.1×10^{-2}	6.7×10^{-4}	832.32	873.56	2054.00	878.23	2489.25
	SBQBU	7.5×10^{-2}	3.5×10^{-5}	1328.22	1788.84	17239.10	1809.25	20145.66

TABLE IX: Experimental Results : Attack on Hardware Redundancy

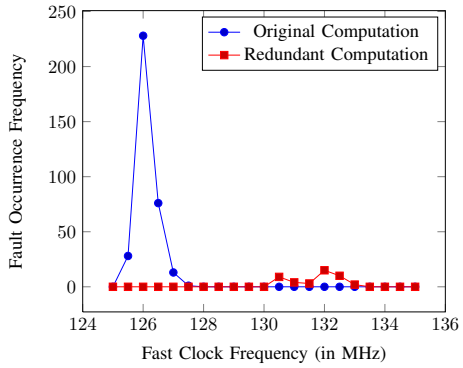
Round	Fault Model	Fault Variance		N_C	N_F(simulation)		N_F(experimental)	
		Type-1	Type-2		Type-1	Type-2	Type-1	Type-2
8	SBU	1.1×10^{-1}	2.5×10^{-3}	300.25	336.75	715.25	323.19	693.81
	SBDBU	9.4×10^{-2}	1.3×10^{-3}	651.10	1425.68	1386.30	1455.37	1498.53
	SBTBU	5.6×10^{-3}	6.2×10^{-4}	989.80	1857.35	2245.40	1824.57	2168.68
	SBQBU	4.5×10^{-3}	3.9×10^{-5}	1723.96	7535.65	32489.35	7503.24	35582.15
9	SBU	9.5×10^{-2}	1.9×10^{-3}	304.24	390.40	601.23	377.38	598.71
	SBDBU	7.7×10^{-2}	8.3×10^{-4}	618.75	646.88	1488.25	664.25	1605.79
	SBTBU	7.6×10^{-2}	2.8×10^{-4}	882.85	891.69	2007.84	828.98	2145.36
	SBQBU	3.4×10^{-2}	5.6×10^{-5}	1299.35	1850.61	25532.45	1913.34	25220.50



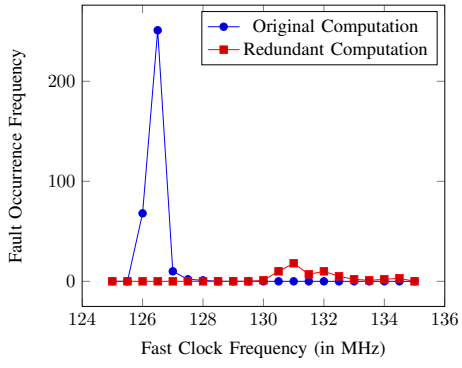
(a) Fault Space Transformation : SBU



(b) Fault Space Transformation : SBDBU



(c) Fault Space Transformation : SBTBU



(d) Fault Space Transformation : SBQBU

Fig. 10: Effect of Fault Space Transformation on the Time Redundancy Countermeasure

original computation to be mapped to a certain subset of the space of four byte faults in the redundant computation. These faults therefore have a different frequency range of occurrence as compared to the single byte faults. Moreover, due to the inherent bias in the fault model, the probability distribution of four byte faults is non-uniform. As is apparent from the fault frequency plots, the target subset of four byte faults have very low probability of occurrence.

One might argue that in the time redundant set-up, he adversary could still introduce equivalent faults by injecting two clock glitches with two different frequencies. We note however that the required difference between the glitch frequencies in fairly low (in the range of 4 MHz), and it would require extremely high precision fault injection equipment to inject faults at precisely these two frequency ranges. Further, even if the adversary were to hypothetically manage the fault injection, the probability that the desired equivalent fault would be injected is still very low. With respect to hardware redundancy, injecting faults with two different frequencies is only possible if the two copies of the hardware were to be supplied external clocks from different sources. However, since in this discussion, we assume that the clock supply to both the circuits is the same, this attack is not possible.

X. Conclusions

In this work, we have proposed a common countermeasure against classical and biased fault attacks. Our proposed countermeasure scheme combines the traditional principle of redundancy (which is found to be successful against uniform fault attacks) with that of fault space transformation to counter both variety of fault attacks. The work first proposes a formal quantification of the bias of a fault model using the variance of the fault probability distribution. It then shows that although traditional concurrent error detection (CED) techniques ensure security against uniform fault models, they are significantly weakened in the presence of practically achievable biased fault models. The work then establishes that even standard countermeasure techniques against side channel analysis such as masking fail to thwart biased fault attacks. Finally, the concept of fault space transformation is introduced as a possible countermeasure technique against biased fault attacks. The idea is to ensure that the adversary cannot introduce identical faults in the original and redundant computations as she did earlier to bypass the fault detection step. This is done by introducing an additional linear operation in the redundant computation corresponding to each round, which means that the adversary must now inject equivalent faults albeit in different fault spaces to beat the detection step. This in turn makes biased fault attacks infeasible, while also countering traditional fault attacks such as DFA. The effectiveness of our proposed countermeasure has been validated via simulations and real life experiments on a Spartan 3A FPGA on a SASEBO GII board. To the best of our knowledge, this is the first countermeasure scheme to provide security against both differential as well as side-channel based fault attacks.

References

[1] Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? In: IOLTS. pp. 235–239 (2010)

[2] Agoyan, M., Dutertre, J.M., Naccache, D., Robisson, B., Tria, A.: When Clocks Fail: On Critical Paths and Clock Faults. Smart Card Research and Advanced Application pp. 182–193 (2010)

[3] Amiel, F., Clavier, C., Tunstall, M.: Fault analysis of dpa-resistant algorithms. In: Fault diagnosis and tolerance in cryptography, pp. 223–236. Springer (2006)

[4] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE 94(2), 370–382 (2006)

[5] Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE 100(11), 3056–3076 (2012)

[6] Barenghi, A., Hocquet, C., Bol, D., Standaert, F.X., Regazzoni, F., Koren, I.: Exploring the feasibility of low cost fault injection attacks on sub-threshold devices through an example of a 65nm aes implementation. In: RFID. Security and Privacy, pp. 48–60. Springer (2012)

[7] Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. Computers, IEEE Transactions on 52(4), 492–505 (2003)

[8] Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. dvances in Cryptology CRYPTO'97,Springer pp. 513–525 (1997)

[9] Blömer, J., Seifert, J.P.: Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In: Wright, R.N. (ed.) Financial Cryptography, Lecture Notes in Computer Science, vol. 2742, pp. 162–181. Springer (2003)

[10] Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT 1997, Lecture Notes in Computer Science, vol. 1233, pp. 37–51. Springer (1997)

[11] Canivet, G., Clédière, J., Ferron, J.B., Valette, F., Renaudin, M., Leveugle, R.: Detailed analyses of single laser shot effects in the configuration of a virtex-ii fpga. In: On-Line Testing Symposium, 2008. IOLTS'08. 14th IEEE International. pp. 289–294. IEEE (2008)

[12] Canivet, G., Maistri, P., Leveugle, R., Clédière, J., Valette, F., Renaudin, M.: Glitch and laser fault attacks onto a secure aes implementation on a sram-based fpga. Journal of Cryptology 24(2), 247–268 (2011)

[13] Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of aes. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on. pp. 7–15. IEEE (2012)

[14] Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of aes. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on. pp. 7–15. IEEE (2012)

[15] Demming, R., Duffy, D.J.: Introduction to the Boost C++ Libraries; Volume I-Foundations. Datasim Education BV (2010)

[16] Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on aes. In: Applied Cryptography and Network Security. pp. 293–306. Springer (2003)

[17] Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault Attacks on AES with Faulty Ciphertexts Only. In: Fischer, W., Schmidt, J.M. (eds.) Fault Diagnosis and Tolerance in Cryptography – FDTC 2013, pp. 108–118. IEEE Computer Society (2013)

[18] Fuhr, T., Jaulmes, E., Lomne, V., Thillard, A.: Fault Attacks on AES with Faulty Ciphertexts Only. 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography(FDTC).IEEE pp. 108–118 (2013)

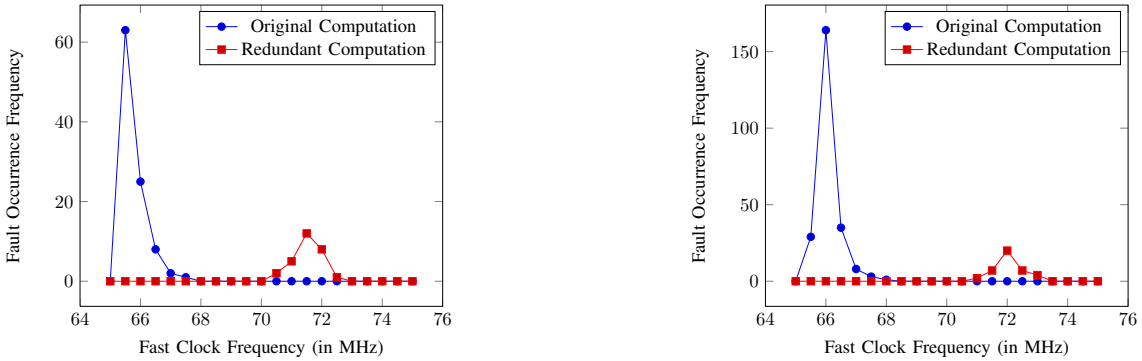
[19] Ghalaty, N., Yuce, B., Taha, M., Schaumont, p.: Differential Fault Intensity Analysis. 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography(FDTC).IEEE (2014)

[20] Giraud, C.: DFA on AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) Advanced Encryption Standard – AES, Lecture Notes in Computer Science, vol. 3373, pp. 27–41. Springer (2005)

[21] Giraud, C., Thiebeauld, H.: A survey on fault attacks. In: Smart Card Research and Advanced Applications VI, pp. 159–176. Springer (2004)

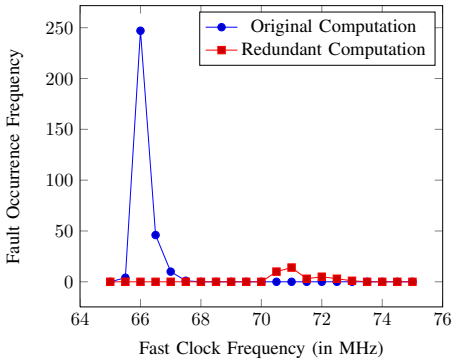
[22] Guo, X., Karri, R.: Invariance-based concurrent error detection for advanced encryption standard. In: Proceedings of the 49th Annual Design Automation Conference. pp. 573–578. ACM (2012)

[23] Guo, X., Karri, R.: Recomputing with permuted operands: A concurrent error detection approach. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 32(10), 1595–1608 (2013)

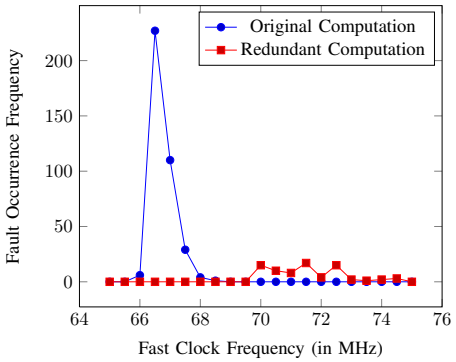


(a) Fault Space Transformation : SBU

(b) Fault Space Transformation : SBDBU



(c) Fault Space Transformation : SBTBU



(d) Fault Space Transformation : SBQBU

Fig. 11: Effect of Fault Space Transformation on the Hardware Redundancy Countermeasure

[24] Guo, X., Mukhopadhyay, D., Jin, C., Karri, R.: Security analysis of concurrent error detection against differential fault analysis. Journal of Cryptographic Engineering pp. 1–17 (2014)

[25] Hemme, L.: A Differential Fault Attack against Early Rounds of (triple-)DES. Cryptographic Hardware and Embedded Systems, CHES 2004, Springer pp. 254–267 (2004)

[26] Joye, M., Manet, P., Rigaud, J.B.: Strengthening hardware aes implementations against fault attacks. IET Information Security 1(3), 106–110 (2007)

[27] Junod, P., Vaudenay, S.: Perfect diffusion primitives for block ciphers. In: Selected Areas in Cryptography. pp. 84–99. Springer (2005)

[28] Kaminsky, A., Kurdziel, M., Radziszowski, S.: An overview of cryptanalysis research for the advanced encryption standard. In: MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010. pp. 1310–1316. IEEE (2010)

[29] Karaklajic, D., Schmidt, J.M., Verbauwhede, I.: Hardware designer's guide to fault attacks. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 21(12), 2295–2306 (2013)

[30] Karpovsky, M., Kulikowski, K.J., Taubin, A.: Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In: Dependable Systems and Networks, 2004 International Conference on. pp. 93–101. IEEE (2004)

[31] Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 21(12), 1509–1517 (2002)

[32] Khelil, F., Hamdi, M., Guilley, S., Danger, J.L., Selmane, N.: Fault analysis attack on an fpga aes implementation. In: New Technologies, Mobility and Security, 2008. NTMS'08. pp. 1–5. IEEE (2008)

[33] Kim, C.H.: Differential fault analysis against aes-192 and aes-256 with minimal faults. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on. pp. 3–9. IEEE (2010)

[34] Kim, H.: Differential Fault Analysis against AES-192 and AES-256 with Minimal Faults. 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography(FDTC),IEEE pp. 3–9 (2010)

[35] Kim, H.: Improved Differential Fault Analysis on AES Key Schedule. IEEE Transactions on Information Forensics and Security 7(1), 41–50 (2012)

[36] Lashermes, R., Reymond, G., Dutertre, J.M., Fournier, J., Robisson, B., Tria, A.: A dfa on aes based on the entropy of error distributions. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on. pp. 34–43. IEEE (2012)

[37] Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault sensitivity analysis. In: Cryptographic Hardware and Embedded Systems-CHES 2010, pp. 320–334. Springer (2010)

[38] Maistri, P., Leveugle, R.: Double-Data-Rate Computation as a Countermeasure against Fault Analysis. IEEE Transactions on Computers 57(11), 1528–1539 (2008)

[39] Malkin, T., Standaert, F., Yung, M.: A Comparative Cost/Security Analysis of Fault Attack Countermeasures. 2005 Workshop on Fault Diagnosis and Tolerance in Cryptography(FDTC),IEEE pp. 109–123 (2005)

[40] Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A generalized method of differential fault attack against aes cryptosystem. In: Cryptographic Hardware and Embedded Systems-CHES 2006, pp. 91–100. Springer (2006)

[41] Mozaffari-Kermani, M., Reyhani-Masoleh, A.: Concurrent structure-independent fault detection schemes for the advanced encryption standard. Computers, IEEE Transactions on 59(5), 608–622 (2010)

[42] Mukhopadhyay, D.: An Improved Fault Based Attack of the Advanced Encryption Standard. In: Preneel, B. (ed.) Progress in Cryptology – AFRICACRYPT 2009, Lecture Notes in Computer Science, vol. 5580, pp. 421–434. Springer (2009)

[43] Piret, G., Quisquater, J.J.: A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2003, Lecture Notes in Computer Science, vol. 2779, pp. 77–88. Springer (2003)

[44] Piret, G., Quisquater, J.J.: A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. Cryptographic Hardware and Embedded Systems, CHES 2003, Springer pp. 77–88 (2003)

[45] Rivain, M.: Differential Fault Analysis on DES Middle Rounds. Clavier and Gaj[8] pp. 457–469

[46] Saha, D., Mukhopadhyay, D., Chowdhury, D.R.: A diagonal fault attack

on the advanced encryption standard. IACR Cryptology ePrint Archive 2009, 581 (2009)
[47] Satoh, A., Sugawara, T., Homma, N., Aoki, T.: High-performance concurrent error detection scheme for aes hardware. In: Cryptographic Hardware and Embedded Systems–CHES 2008, pp. 100–112. Springer (2008)
[48] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: The Twofish encryption algorithm: a 128-bit block cipher. John Wiley & Sons, Inc. (1999)
[49] Selmane, N., Guilley, S., Danger, J.L.: Practical setup time violation attacks on aes. In: Dependable Computing Conference, 2008. EDCC 2008. Seventh European. pp. 91–96. IEEE (2008)
[50] Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication, pp. 224–233. Springer (2011)
[51] Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. Information Security Theory and Practice.Security and Privacy of Mobile Devices in Wireless Communication, Springer pp. 224–233 (2011)
[52] Vaudenay, S.: On the need for multipermutations: Cryptanalysis of md4 and safer. In: Fast Software Encryption. pp. 286–297. Springer (1995)
[53] Wu, K., Karri, R., Kuznetsov, G., Goessel, M.: Low cost concurrent error detection for the advanced encryption standard. In: Test Conference, 2004. Proceedings. ITC 2004. International. pp. 1242–1248. IEEE (2004)