UNIVERSITY OF CALIFORNIA SAN DIEGO

Using Blinking to Mitigate Passive Side Channel Attacks and Fault Attacks

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Jeremy Blackstone

Committee in charge:

Professor Ryan Kastner, Chair
Professor Sean Gao
Professor Truong Nguyen
Professor Lawrence Saul
Professor Geoff Voelker

2021

The Dissertation of Jeremy Blackstone is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

DEDICATION

To my wife Abigail Blackstone for reading and giving feedback on research papers regarding topics she is unfamiliar with and being the inspiration to finish them.

EPIGRAPH

Failure is simply the opportunity to begin again, this time more intelligently.

Henry Ford

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Ryan Kastner for his support as the chair of my committee. Through multiple drafts, his guidance has proved to be invaluable.

Chapter 3, ”Power Analysis Mitigation” is based on ”Using Stalls to Minimize Information Leakage”. It is coauthored with Alric Althoff, Scott Davidson, Dustin Richmond, Michael Taylor and Ryan Kastner and is currently being prepared for submission for publication of the material. The dissertation author was the primary author of this chapter.

Chapter 4, ”Fault Analysis Mitigation” is based on ”Using Intermittent Isolation To Mitigate Fault Attacks”. It is coauthored with Alric Althoff, Scott Davidson, Dustin Richmond, Michael Taylor and Ryan Kastner and is currently being prepared for submission for publication of the material. The dissertation author was the primary author of this chapter.

Chapter 5, ”EM Analysis Mitigation” is based on ”iSTELLAR: intermittent Signature aTenuation Embedded CRYPTO with Low-Level metAl Routing”. It is coauthored with Debayan Das, Shreyas Sen and Ryan Kastner and is currently being prepared for submission for publication of the material. The dissertation author was the primary author of this chapter.

VITA

2014	B. S. in Systems and Computer Science, Howard University, Washington, DC
2015	M. S. in Systems and Computer Science, Howard University, Washington, DC
2021	Ph. D. in Computer Science, University of California, San Diego

ABSTRACT OF THE DISSERTATION

Using Blinking to Mitigate Passive Side Channel Attacks and Fault Attacks

by

Jeremy Blackstone

Doctor of Philosophy in Computer Science

University of California San Diego, 2021

Professor Ryan Kastner, Chair

Ignoring security concerns when building digital hardware allows for malicious parties to take advantage of vulnerabilities to gain access to secret information and manipulate systems. This is unacceptable because of the disastrous results of attackers compromising consumer products such as cell phones, smart cards and automobiles. To this end, researchers have developed numerous mathematically secure cryptographic algorithms.

Unfortunately, side channel analysis (SCA) attacks bypass these algorithms by monitoring the effects of the algorithm on a physical platform through power consumption, electromagnetic emanations (EM), or subjecting it to fault injection. These effects are referred to as side channels. By analyzing side channels, an attacker is able to discover sensitive information, e.g., extracting

the secret key from a cryptographic algorithm.

It is shown in [3] that retrieving side channel information is not uniform. Some portions of the execution reveal a large amount of information to an adversary while other portions reveal little to no information to the adversary. However, most SCA countermeasures incur larger than necessary overhead by protecting all portions of the computation. One way to reduce overhead is through a methodology called blinking. Blinking identifies the most critical points in time for a cryptographic computation and performs isolation to prevent an adversary from observing or modifying any information. This thesis proposes using blinking in a variety of different scenarios and provides analysis so hardware designers can make informed decisions on how to balance the performance, area overhead, power consumption and security.

First, this thesis provides analysis on using blinking to mitigate power analysis attacks with an on-chip capacitor. Next, this thesis shows how the same on-chip capacitor can be used to protect against differential fault analysis, fault sensitivity analysis, biased fault analysis and combined fault analysis. Finally, this thesis demonstrates how blinking can be used to attenuate EM analysis.

Chapter 1

Background

1.1 Cryptographic Algorithms

Cryptographic algorithms are essential to protecting integrity and confidentiality for data. These algorithms keep data secure using hardware specifically designed to execute them in order to perform operations as quickly as possible[8, 45]. Many products including smart cards, mobile devices and TV set-top boxes implement symmetric encryption algorithms such as the Advanced Encryption Standard(AES).

Symmetric encryption hinges on the idea that a secret key can introduce uncertainty into a encryption algorithm[135]. Sharing the secret key only with authorized users allows for them to access the data while disallowing access to unauthorized users. Cryptography focuses on ensuring that the computational effort of determining this secret key using a brute force search is exponentially dependent on the key size to the point that it is not feasible for an adversary to accomplish it using modern computing resources.

Side channel attacks allow an adversary to use information from the algorithm's physical implementation on hardware to extract secret data and assist in finding the key by reducing the key space. Side channel attacks are important because they undermine the algorithmic complexity required to keep the secret key secure. The key space is important because it is the set keys that will be used in an exhaustive search for the secret key.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 1.1. AES's S-box

1.1.1 AES

The Advanced Encryption Standard(AES) is a cryptographic algorithm which takes a 128 bit plaintext and encrypts it using a secret key [116]. It is composed of iterations called rounds and the number of rounds corresponds to the key length. If the secret key is 128 bits, there are 10 rounds, if it is 192 bits there are 12 rounds and if it is 256 bits there are 14 rounds. Each round is composed of 4 layers: Byte Substitution, ShiftRows, MixColumns and Key Addition.

When computing AES, the plaintext is first exored with the secret key and sent to the Byte Substitution layer. In this layer, each byte is substituted based on a lookup table called an S-Box as shown in figure 1.1. The first 4 bits of the plaintext are used to determine the column and the last 4 bits are used to determine the row. For example, if the input byte is d7 it will be replaced substituted with the byte 0e.

Next, each byte is considered an element in a 4 x 4 state matrix and sent to the ShiftRows

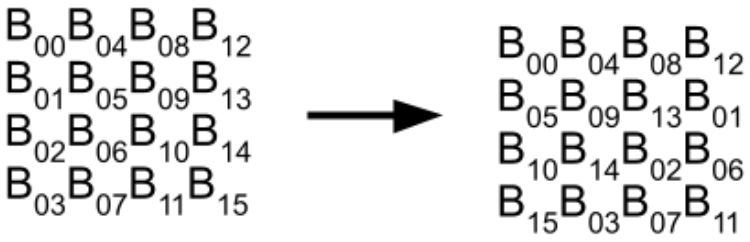


Figure 1.2. Shift Rows layer

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

Figure 1.3. Mix Columns layer

layer.

In this layer, the first row of the matrix remains unchanged, the second row is shifted to the left 1 byte, the third row is shifted to the left 2 bytes and the fourth row is shifted to the left 3 bytes as shown in Figure 1.2.

After the ShiftRows layer, the MixColumns layer does matrix multiplication between the current state matrix(B_i) and the matrix shown in Figure 1.3 to produce a new state matrix (C_i).

Finally, in the Key Addition layer, each bit is exored with a 16 byte variation of the secret key called a round key. Round keys are generated by a key scheduling algorithm as shown in Figure 1.4. In this algorithm, first the initial key is divided into words and can be considered round key 0. Next, the last word of round key 0 is sent to a RotWord function which shifts the positions of bytes in the word and a SubWord function which sends the word into an S-box. This value is exored with a constant value called RCON to generate the first word of round key 1. The second word is generated by exoring the first word of the current round key and the second word

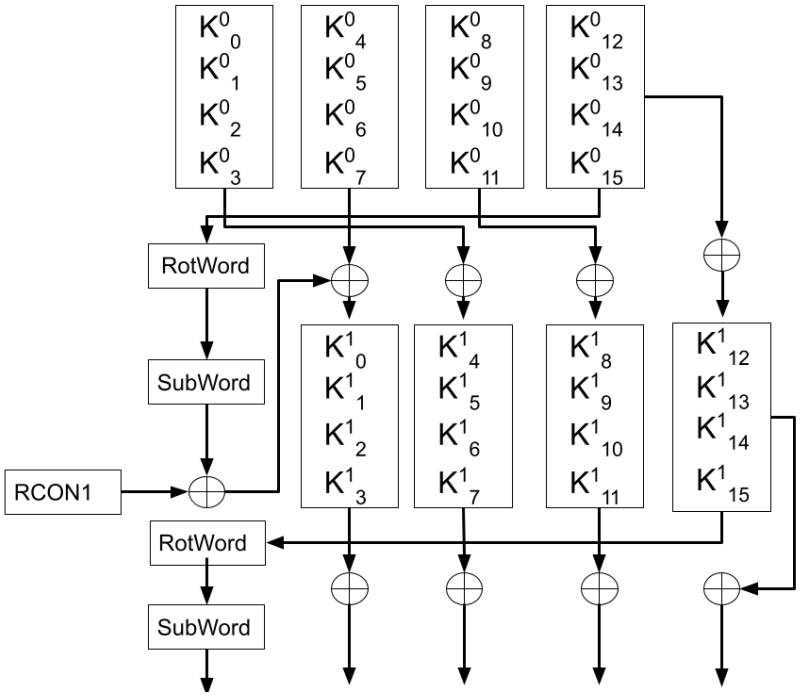


Figure 1.4. Key Scheduling Algorithm

from round key 0. Each subsequent word is generated by exoring the preceding word from the current round with the parallel word from the previous round. The last word of round key 1 is sent to the RotWord and SubWord functions and this process continues until all 10 round keys have been generated.

1.1.2 PRESENT

While AES is the preferred algorithm of choice for most applications, it is not best suited for highly resource constrained devices such as RFID tags and sensor networks [16]. To address this problem, PRESENT has been proposed as an ultralightweight block cipher.

PRESENT can be implemented on a 64-bit plaintext with either 80-bit or 128-bit keys. Like AES, PRESENT is also composed of rounds, but it is always 31 rounds regardless of the key length. Each round consists of an AddRoundKey layer, S-box layer and permutation layer.

In the AddRoundKey layer, each bit is exored with a 64 bit variation of the secret key

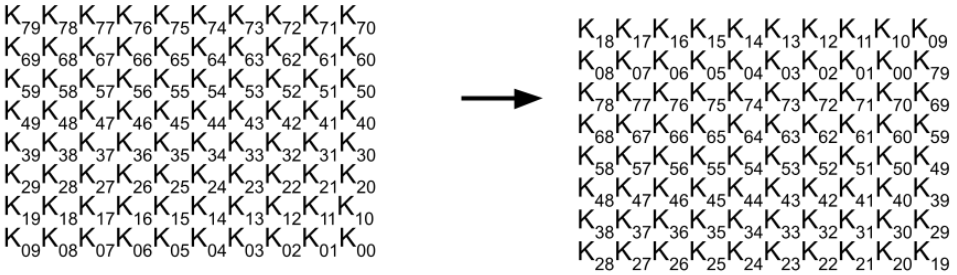


Figure 1.5. PRESENT key shift operation

Table 1.1. PRESENT's S-Box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

called a round key. Round keys are generated by a key scheduling algorithm as shown in Figure 1.4.

While PRESENT can have an 80bit or 128-bit key, we focus on an example with an 80-bit key. In this algorithm, the 64 leftmost bits of the key (K) are chosen as the initial round key (K^0). Next, the key is shifted left by 61 positions as shown in figure 1.5 and table 1.1. Afterwards, the four leftmost bits are substituted through PRESENT's S-Box as shown in equation 1.1. Finally, bits $K_19K_18K_17K_16K_15$ are exored with the least significant bit of the round counter as shown in equation 1.2 where i is the round number.

$$K_{79}K_{78}K_{77}K_{76} = S(K_{79}K_{78}K_{77}K_{76}) \tag{1.1}$$

$$K_{19}K_{18}K_{17}K_{16}K_{15} = K_{19}K_{18}K_{17}K_{16}K_{15} \oplus i_0 \tag{1.2}$$

In the S-Box layer, each nibble is substituted based on PRESENT's S-box as shown in table 1.1.

In the permutation layer, each bit in the current state is moved to a new position. This is shown in table 1.2 where i is the initial position of the bit and P(i) is the new position of the bit.

Table 1.2. PRESENT's permutation layer places each bit i in new position P(i)

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

1.2 Passive Side Channel Attacks

1.2.1 Power Analysis

Microprocessors are composed of logical gates which behave differently based on the data and instructions they are given as input[5]. This is because power is applied or removed from transistors to hold values or perform an operation on values. Therefore, power consumption contains information about a device's operations because differing behaviors require differing amounts of power to perform. Power analysis attacks correlate power consumption measurements collected during cryptographic computations to a device's operation and secret information[77]. A number of power attacks have been used to reveal the secret key in a variety of scenarios [96, 138, 77, 19, 61, 140, 92, 26, 97, 78, 15, 118, 90, 94]

Simple Power Analysis

Simple Power Analysis(SPA) is the most basic class of power attacks. These attacks simply observe a power trace, collection of power measurements taken over a period of time, and attempt to derive information from it[77]. SPA uses variations in power to determine sequences of instructions. As an example, an adversary could collect the power measurements for a round of AES and distinguish which cycles are performing Byte Substitution, ShiftRows, MixColumns

and Key Addition [127].

Differential Power Analysis

While SPA is useful for identifying properties of a cryptogrphic algorithm, in many cases, simply observing a power trace is not sufficient to recover the entire secret key. Differential Power Analysis (DPA) solves this problem by analyzing the differences between power traces. Furthermore, is capable of launching a successful attack even when there is a significant amount of noise in the power measurement. The attacker simply needs to acquire more power measurements to develop a reasonable signal to noise ratio.

Classical DPA approaches take a univariate approach by choosing a single point in an algorithm's execution, collect a number of traces and divides them into two buckets [87]. The traces are divided into buckets using a selection function which based on the assumption that a device consumes more power changing a bit from 0 to 1 than changing a bit from 1 to 0. Thus, power traces with greater power consumption will be placed in one bucket and power traces with lesser power consumption will be placed in the other.

An example of DPA is shown in figure 1.6. The adversary begins by running an encryption algorithm a number of times using a hypothetical key guess [87]. Next, they collect power traces for a bit in the algorithm's execution that is directly dependent on a subset of the secret key(subkey). After this, they divide the traces into 2 buckets based on the value of the bit dependent on the subkey. The adversary calculates the mean of each bucket and repeats this process using different values for the subkey. They determine which subkey is correct by finding which value results in the largest difference between means for its two buckets. The adversary repeats this process until they are able to recover all the bits of the secret key.

More recent DPA approaches take a multivariate approach by comparing multiple points on a trace to one another rather than choosing a single point [127]. This trend arose because previous approaches consider measurements of power consumption at different points in time to be independent from one another. This means that if the power consumption of point 1 changes,

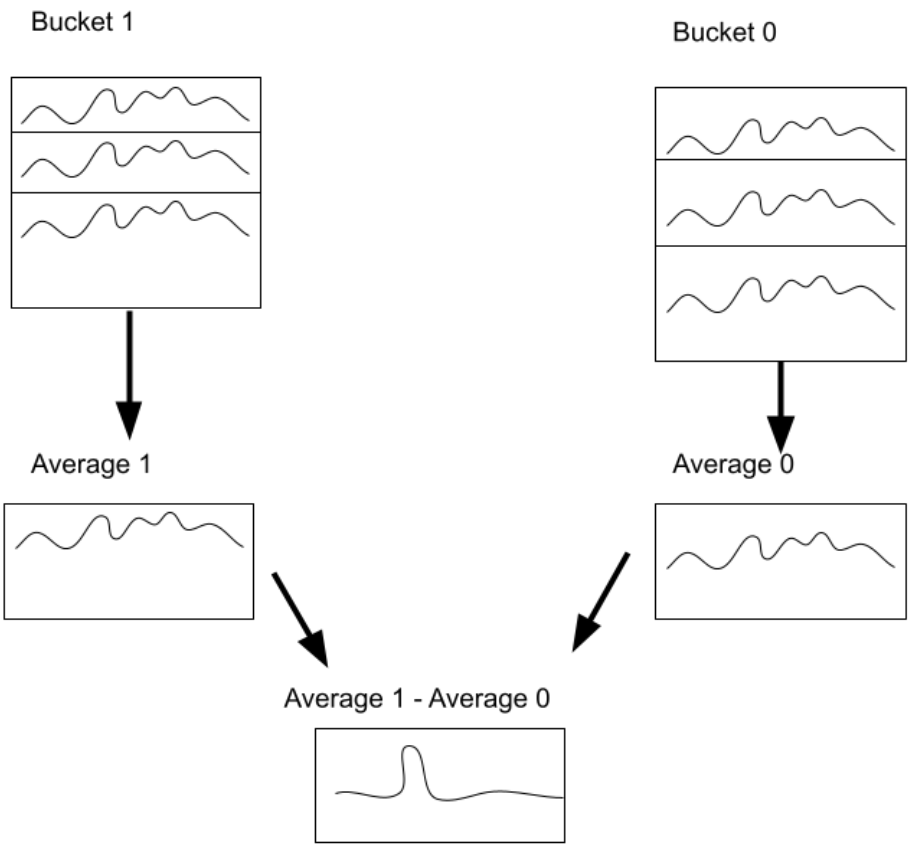


Figure 1.6. Differential Power Analysis(DPA) example

there is no guarantee that the power consumption of point 2 will change. However, in the case of encryption, it is very possible for one point in an algorithm's execution to directly affect another point in the algorithm's execution. Therefore, it is is possible for an adversary to acquire secret information by analyzing how different points in an algorithm change with respect to each other.

One way to launch a multivariate DPA attack is by developing a multivariate Gaussian distribution [127]. A Gaussian distribution is a way to determine the likelihood of a measurement being equal to specific value. If the measurements within a bucket are normally distributed, it is highly likely that a bucket assumes the correct value for a bit and if the measurements within a bucket are not normally distributed it is highly unlikely that the bucket assumes the correct value for a bit. Furthermore, if we consider multiple points in a trace, the variance of each individual

point and the variance between each 2 sets of points we are able to distinguish subkey guesses in the presence of greater noise. This is because there are more requirements a subkey guess must satisfy in order to achieve high likelihood value.

Correlation Power Analysis

While DPA compares power traces to other power traces, Correlation Power Analysis (CPA) develops a power model for the implementation of a cryptographic algorithm [19] and compares power traces to the devloped model. The CPA power consumption model stems from the fact that the power consumption of the device should have a linear correlation with the Hamming Weight of a reference state dependent on a subkey key. The attacker is able to determine the subkey by observing which of their subkey guesses has the highest correlation to the power consumption model.

One way to calculate the correlation between a power model and the actual power consumption is by using the Pearson correlation coefficient equation [87]. The Pearson correlation coefficient equation operates by calculating whether there is a linear correlation between two sets of values. For example, if the Hamming Weight of a reference state increases as the measured power consumption increases, there will be a strong positive correlation between Hamming Weight and power consumption. The adversary determines which subkey is correct by finding which value results in the strongest correlation between Hamming Weight and power consumption.

Template Power Analysis

While DPA and CPA overcome noise by collecting more trace, Template Power Analysis(TPA) overcomes noise by profiling the power consumption of a device and comparing a few power traces to the profile [87]. The TPA power consumption model uses a large number of traces using different potential plaintexts and keys on a device identical to the device they intend to attack. Afterwards, the adversary collects a small number of power traces using the actual

key and determines which subkey guess is correct by determining whether an observed power consumption value is similar to an power consumption value from the profile

1.2.2 Electromagnetic Analysis Attacks

A number of devices generate electromagnetic(EM) radiation at unintended frequencies as a consequence of their internal operations [55]. This occurs because EM waves are generated from electrical currents changing over time and the frequency, amplitude and phase of these waves are dependent on how the electrical currents vary [42]. By analyzing these EM waves, an adversary is able to acquire information about the instructions that generated them. A number of EM attacks have been used to reveal the secret key in a variety of scenarios [55, 68, 114, 9, 33, 11, 24, 22, 21, 23, 20, 25, 46, 50, 52, 53, 54, 66, 69, 70, 71, 73, 79, 93, 98, 111, 115, 124, 133, 132, 136, 142, 141, 143, 149, 42, 43, 139, 41, 47, 63, 95, 126, 83, 147, 144, 39, 29, 112, 100, 102, 101, 103, 91, 84, 1, 148, 137, 91]

Simple Electromagnetic Analysis (SEMA)

One way to acquire information from electromagnetic emanation is through Simple Electromagnetic Analysis (SEMA) [139, 52, 53, 66]. One way to launch this attack is by visually inspecting the EM emanations of the computed instructions. This is effective because toggling a bit consumes one bit of power and leaving a bit unchanged consumes no power because this variation in power consumption is reflected in the resulting EM emanations. Without noise, variations in the frequency of the EM emanations can map directly to the executed instructions allowing an adversary to acquire the secret key to a cryptographic algorithm in a with a single EM trace. However, many implementations are very noisy and this form of analysis is not viable.

Differential Electromagnetic Analysis (DEMA)

One way to acquire information from EM emanation in presence of noise is through Differential Electromagnetic Analysis (DEMA) [55]. Just as DPA analyzes the differences between power traces DEMA analyzes the differences between EM traces. As a result, it is

possible to launch a successful attack even if there is a significant amount of noise by collecting more EM measurements.

One way to implement DEMA is by calculating the hamming distance between the value of register before an instruction and the value of a register after an instruction based on the EM emanations they generate [55]. An example of this is shown in figure 1.7. First, the adversary identifies a set of input data bytes, and all possible key bytes. Next, each input data byte is fed into the device running the encryption algorithm with the real key and EM traces are collected for each input value before and after the step which performs an XOR operation between the input value and the secret key. The XOR values are recorded in a matrix referred to as the *real matrix* where I^j is input byte j and R_b is the difference in EM traces at bit b before and after the step which performs an XOR operation between the input value and the secret key. After this, they perform XOR operations between each hypothetical key and the input bytes. These XOR values are recorded in a matrix referred to as the *hypothetical matrix* where I^j is input byte j, k^p is key hypothesis p and H^j, p is the result performing an XOR operation between the input value and the key hypothesis. Finally, the adversary determines which key hypothesis has the highest correlation between the real matrix and the hypothetical matrix for an input value as indicated by the red circles.

Template Electromagnetic Analysis (TEMA)

Another way to acquire information from EM emanation in presence of noise is through Template Electromagnetic Analysis (TEMA) [139]. Just as TPA uses a large number of traces to profile the power consumption of a device, TEMA uses a large number of traces to profile the EM emanations for a device. When performing TEMA an adversary collects a large number of traces using different potential plaintexts and keys on a device identical to the device they intend to attack, then collects a small number of EM traces using the actual key. They are able to and determine which subkey guess is correct by determining whether the observed EM value is similar to an EM value from the profile.

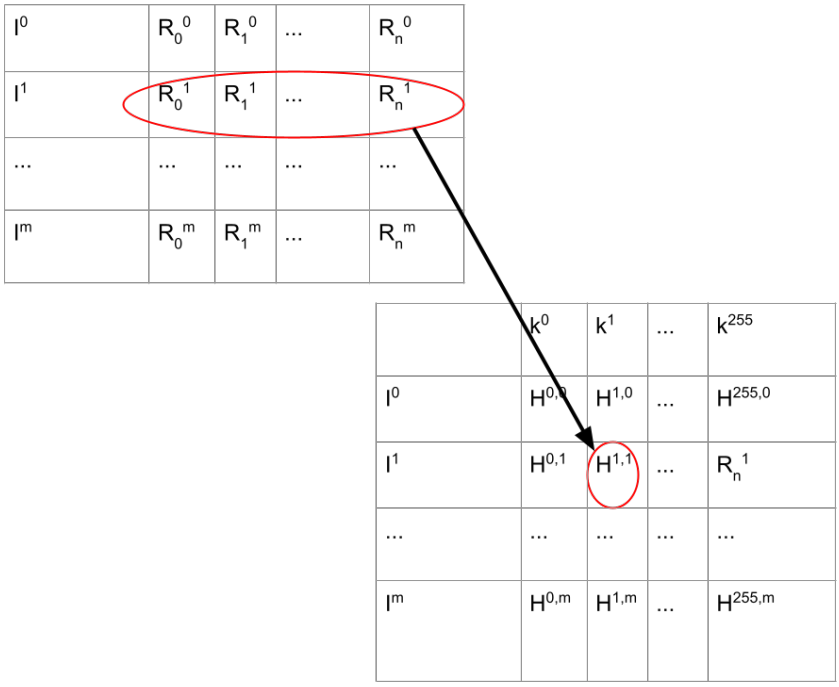


Figure 1.7. Differential Electromagnetic Analysis(DEMA) example

1.3 Fault Analysis Attacks

Fault attacks effectively extract the information a cryptographic algorithm protects by analyzing the effects of carefully placed bit flips. By analyzing the effects of bit flips, adversaries are able to acquire secret key bits and reveal sensitive information. A number of EM attacks have been used to reveal the secret key in a variety of scenarios [31, 17, 10, 48, 28, 72, 129, 106, 13, 105, 85, 56, 60, 49, 80, 57, 67, 86, 14, 40, 120, 108, 131, 62, 44, 107, 18, 5, 32, 130, 146, 76, 6, 8, 7, 134, 58]

1.3.1 Differential Fault Analysis

Adversaries are able to determine a secret key with correct and faulty ciphertext pairs using a technique called Differential Fault Analysis (DFA) [14]. In DFA, the adversary analyzes the differences between faulty ciphertexts and their corresponding correct ciphertexts to reveal secret information. The adversary begins by running AES and getting the correct ciphertext.

Next, they run AES again using the same plaintext and injecting a fault. Some attacks inject faults during the initial AK step and others injecting faults on later rounds.

In attacks injecting faults during the initial AK step, the adversary simply observes whether their fault affects the output. However, in attacks injecting faults on later rounds, the adversary uses the fact that the diffusion steps of the algorithm spread faults to solve for multiple key bytes at once. AES spreads the effects of the fault to every column with each SR step and spreads the effects of the fault to each row within a column in different multiples with each MC step. Finally, the adversary solves for key bytes by determining which key guess byte causes the faulty ciphertext and its corresponding correct ciphertext to differ by the same amount ($Y^*(X'_1 \oplus X_1) = Z^*(X'_2 \oplus X_2)$).

The foundation for all fault analysis attacks on AES was a Differential Fault Analysis(DFA) proposed by Giraud in 2004[62]. The first step in DFA is to simply run the AES algorithm normally. The attacker starts with a plaintext P. After each round of AES an intermediate state S^i occurs where i is the round which caused S. Finally, after all 10 rounds of AES-128 the algorithm outputs the ciphertext C. Next, the attacker uses the same plaintext and runs the AES algorithm again. However, this time she induces a fault at the end of round 9. This results in a faulty intermediate state S'^9 and subsequently a faulty ciphertext because the remainder of the algorithm is operating on the wrong value of S^i. In order to perform analysis, the attacker seeks to use the differences between the faulty and correct ciphertexts to determine the correct value for S^9. After this, the attacker will use the value of S^9 to derive the value of the 10th round key. Finally, she is able to use the inverse key scheduling algorithm to acquire the entire 128 bit AES key.

The first step in analysis is to compute the difference between C and C'[62] as shown in Figure 1.8 . Assuming a fault was only induced in a single byte, this will cause a one byte difference between C and C'. We will use the subscript j to note the byte which has been affected by a fault and se_j to note the differences caused as a result of the fault. Now that the attacker knows the byte the fault has affected, she will try to work backwards to determine the conditions

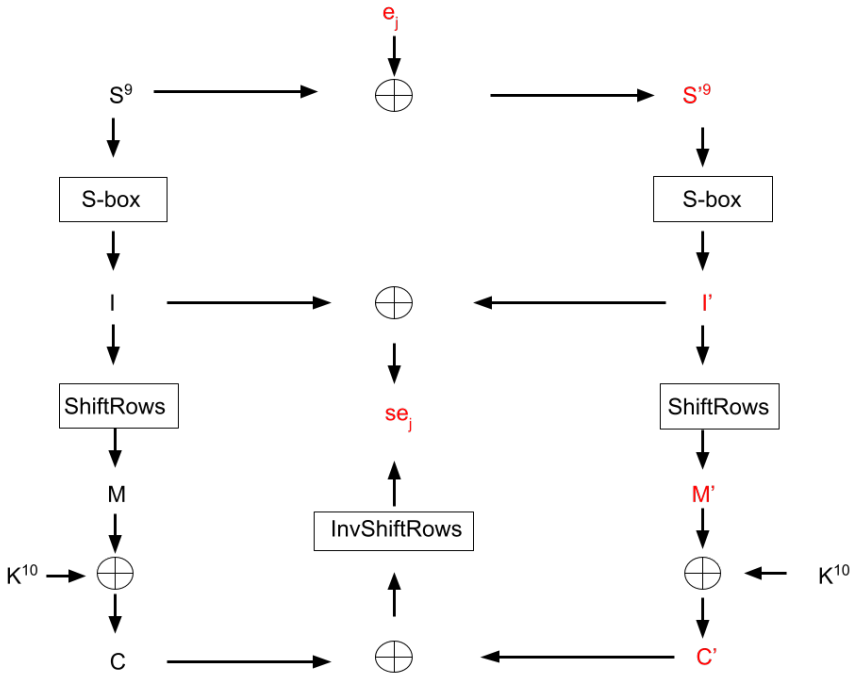


Figure 1.8. Difference between ciphertexts

under which the fault was induced. The value of S^9 can be derived from the ciphertext by exoring it with the 10th round key and performing inverse ShiftRow and Byte Substitution operations. Using these derivations, it can be shown that the only difference between the faulty and correct ciphertext(C and C‘) should be the difference between them after the Byte Substitution step in the final round(I_j and I'_j). Since the same value for K^{10} is used to exor both M and M’ the difference between them does not change even though the values differ from C and C‘. Furthermore, the Inverse ShiftRows and ShiftRows operations change the positions of bytes within the state matrix but do not change the value of any of the bytes.

It is important that the difference between I and I‘ is the same as the difference between C and C‘ because the attacker does not know the value of K^{10} but can derive its value using (1.3) once she has the value of S^9.

$$C = ISR(ISB(S^9)) + K^{10} \tag{1.3}$$

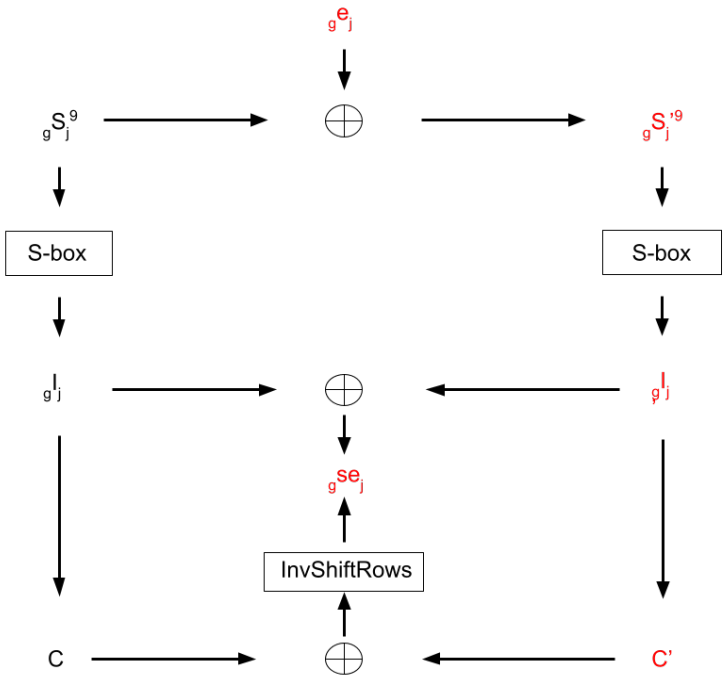


Figure 1.9. DFA key recovery diagram

To find the right value for S^9, the attacker will use the inverse ShiftRows operation on both the correct and faulty cipher texts to align them with their corresponding bytes at the end of round 9 and guess values for the intermediate state byte as well as the fault induced as shown in Figure 1.9.

$$P = 0110101111000001 \tag{1.4a}$$

$$S^9 = 10111011 \tag{1.4b}$$

$$S'^9 = 10111001 \tag{1.4c}$$

$$C = 0011101000010001 \tag{1.4d}$$

$$C' = \mathbf{1000\ 0110}00010001 \tag{1.4e}$$

$$C \oplus C' = \mathbf{1}0\mathbf{1111}0000000000 \tag{1.4f}$$

$$j = 0 \tag{1.4g}$$

$$se_j = \mathbf{1}0\mathbf{1111}00 \tag{1.4h}$$

$$S^9_{j,g} = 10111011 \tag{1.4i}$$

$$e_{j,g} = 0000001\mathbf{0} \tag{1.4j}$$

$$I_{j,g} = 11101100 \tag{1.4k}$$

$$S'^9_{j,g} = 10111001 \tag{1.4l}$$

$$I'_{j,g} = \mathbf{0101\ 0110} \tag{1.4m}$$

$$I_{j,g} \oplus I'_{j,g} = \mathbf{1}0\mathbf{1111}00 = se_j = C_j \oplus C'_j \tag{1.4n}$$

$$\tag{1.4o}$$

We will consider the 1 bit fault in the affected byte e_j, and we will consider the byte affected by that fault S'^9_j. If the attacker guesses the right value for S^9_j and e_j then the difference between the faulty and correct ciphertext byte(C_j and C'_j) will be the difference between them after the Byte Substitution step in the final round(I_j and I'_j). There are 2^8 possibilities for S^9_j, and 2^8 possibilities for a bit fault within a byte and we must do this process for all 16 bytes giving a time complexity of 2^{20} for this algorithm.

The equations in 1.4 provide an example of DFA calculations. In this example the first

byte of the plaintext is 0x6b in hexadecimal representation and by the end of round 9 it has been changed to 0xbb.

If the second bit from the right is flipped from 1 to 0 then the faulty intermediate state byte is 0xb9. After computing the final round, the first byte is 0x3a for the correct ciphertext and 0x86 for the faulty ciphertext. By exoring the two ciphertexts, the attacker will find the value of the byte difference is 0xbc and confirm that its position is in byte 0 and all other bytes have a byte difference of 0. If the attacker correctly guesses that S_j^9 is 0xbb and that the fault occurred at the second bit from the right then she can compute that $S`_j^9$ is 0xb9. After this, she places both into the S-box and discovers that $I_j$ is ea and $I`_j$ is 0x56. By these intermediate values the attacker discovers that its byte difference is also bc and confirms that she has chosen the correct value for S_j^9. Using the correct value for S_j^9 she can perform the steps to reach state M and exor this with the correct ciphertext to acquire a byte of the 10th round key.

In 2006, a theoretical DFA attack emerged which utilized faults that randomly changed the value of a single byte[107].

In this attack, first the attacker runs AES to produce the correct ciphertext. Next, he runs AES again inducing a fault before the MixColumns step of the ninth round. The first step in analysis is to compute the difference between C and $C`$. Assuming a fault was only induced in the first column, this will cause 4 byte differences between C and $C`$ as shown in Figure 1.10. The shaded regions are the bytes that have been affected by the fault. Just as in [62], the difference between I and $I`$ should be the same as the difference between C and $C`$ for each byte. Using the same steps outlined in [62], the attacker can compute bytes of the 10th round key four at a time rather than one at a time reducing the number of ciphertexts that are necessary.

In 2008, it was shown that DFA can circumvent masking which was widely used as a countermeasure for side channel attacks[18]. In masking, every time the device prepares to compute AES, a random number is generated and referred to as a mask. This mask is exored with the plaintext and all of the linear operations that are applied to the plaintext(ShiftRows, MixColumns, AddRoundkey) are also applied to the mask. One important difference masking

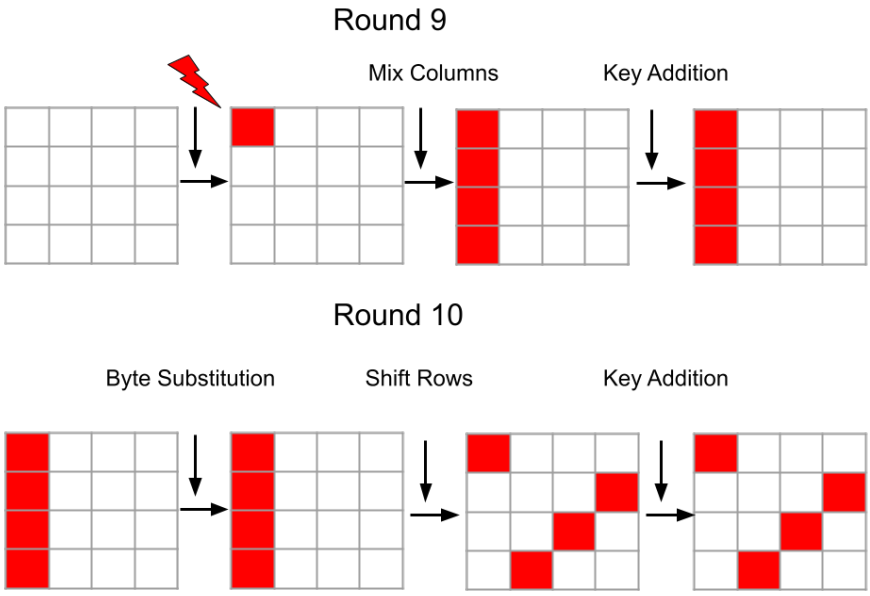


Figure 1.10. Generalized DFA

makes on the algorithm is that the S-box must be dynamically changed for each mask. The S-box must be changed so that the correct intermediate value masked with a random value(u) will produce the correct output with a corresponding random value (v). At the end of the encryption, the resulting ciphertext is exored with the mask again to remove its effects and provide the correct ciphertext.

This is shown in Figure 1.11.

The randomness provided by the mask eliminates the correlation between the secret key and side channels necessary to launch side channel attacks. However, adding randomness does not disrupt the relationship between the correct and faulty ciphertext necessary for DFA. As in [62] the attacker is still able to compute the difference between C and *C*‘ and show that this is the same difference between I and *I*‘ as shown in the diagram below. By repeating the same steps as in [62] the attacker is able to recover the secret key.

In 2009, a DFA attack was discovered capable of breaking AES-128 with only 1 fault.

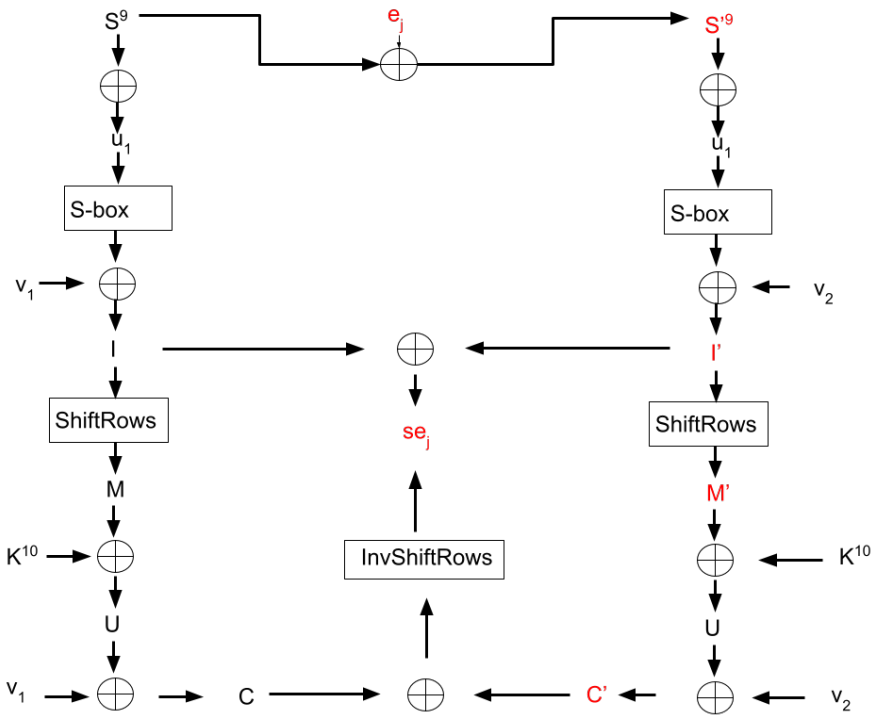


Figure 1.11. Maksed DFA diagram

The attack exploited a 1 byte fault before the MixColumns layer of the eighth round[146].

In this attack, first the attacker runs AES to produce the correct ciphertext. Next, he runs AES again inducing a fault at the end of round 7 and determines the effect the fault will have on the resulting ciphertext as shown in Figure 1.12. Each Byte Substitution and AddRoundKey step affects the value of the faulty byte, each ShiftRows step spreads the effects of the fault to every column and each MixColumns step spreads the effects of the fault to each row within a column in different multiples. After MixColumns, the value of the fault effect is equivalent in rows 1 and 2, row 0 has twice this value and row 3 has three times this value. The initial value of the induced fault will be denoted as f, f' will represent the fault value after the 8th round Byte Substitution, F1, F2, F3 and F4 will represent the fault values in the shaded bytes after the 9th round Byte Substitution and A1 through A16 will represent the fault values in each byte after the 10th round Byte Substitution. Following this, the attacker will make guesses for bytes of the 10th round key. Using these key guesses, he will perform Key Addition then inverse ShiftRows

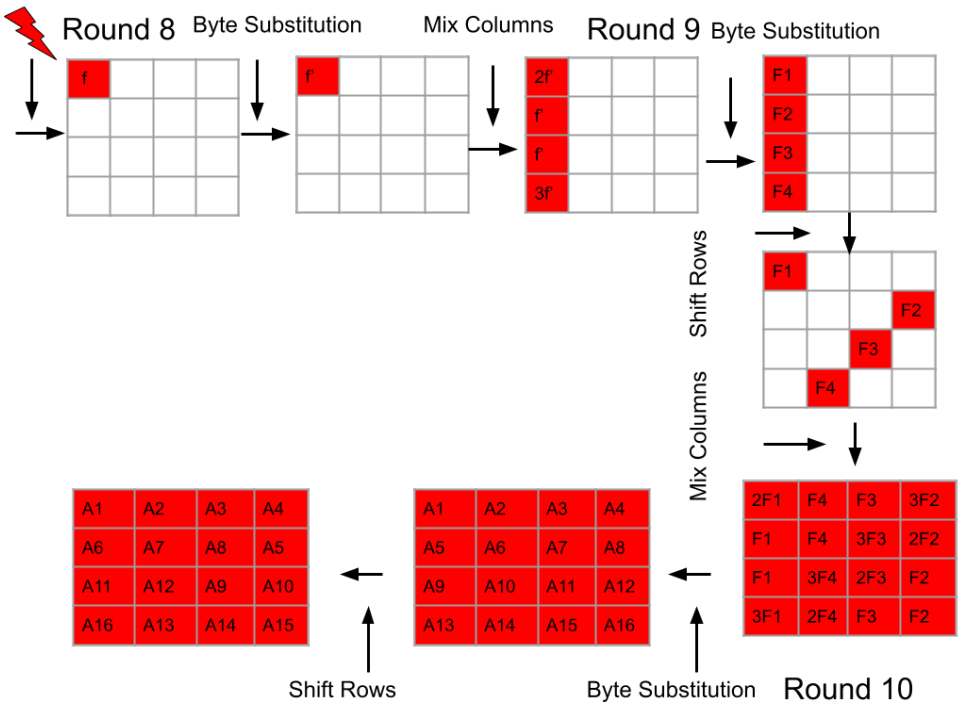


Figure 1.12. Single Fault DFA fault propagation

and Byte Substitution operations to derive the state matrices at the end of round 9. If the correct key guesses were selected then the following systems of equations will be satisfied.

$$S_0^9 + S'^9_0 = 2(S_1^9 + S'^9_1) \tag{1.5}$$

$$S_1^9 + S'^9_1 = S_2^9 + S'^9_2 \tag{1.6}$$

$$S_1^9 + S'^9_1 = 3(S_3^9 + S'^9_3) \tag{1.7}$$

$$S_4^9 + S'^9_4 = S_5^9 + S'^9_5 \tag{1.8}$$

$$S_5^9 + S'^9_5 = 3(S_6^9 + S'^9_6) \tag{1.9}$$

$$S_5^9 + S'^9_5 = 2(S_7^9 + S'^9_7) \tag{1.10}$$

$$S_8^9 + S'^9_8 = 3(S_9^9 + S'^9_9) \tag{1.11}$$

$$S_8^9 + S'^9_8 = 2(S_1^9 0 + S'^9_1 0) \tag{1.12}$$

$$S_8^9 + S'^9_8 = S_1^9 1 + S'^9_1 1 \tag{1.13}$$

$$S_1^9 4 + S'^9_1 4 = 3(S_1^9 2 + S'^9_1 2) \tag{1.14}$$

$$S_1^9 4 + S'^9_1 4 = 2(S_1^9 3 + S'^9_1 3) \tag{1.15}$$

$$S_1^9 4 + S'^9_1 4 = S_1^9 5 + S'^9_1 5 \tag{1.16}$$

If not, he makes another key guess and repeats this process.

$$S_0^9 = ISB(x8 + k8) \tag{1.17}$$

$$S'^9_0 = ISB(x8 + A5 + k8) \tag{1.18}$$

$$S_1^9 = ISB(x11 + k11) \tag{1.19}$$

$$S'^9_1 = ISB(x11 + A9 + k11) \tag{1.20}$$

In 2011, Meet in the Middle(MiM) attacks developed in response to protection schemes limited to the last few rounds of AES[40]. In a MiM attack, first the attacker runs AES to produce the correct ciphertext. Next, she runs AES again inducing a fault at the end of round 6. As in [146], each Byte Substitution and AddRoundKey step affects the value of the faulty byte, each ShiftRows step spreads the effects of the fault to every column and each MixColumns step spreads the effects of the fault to each row within a column in different multiples.

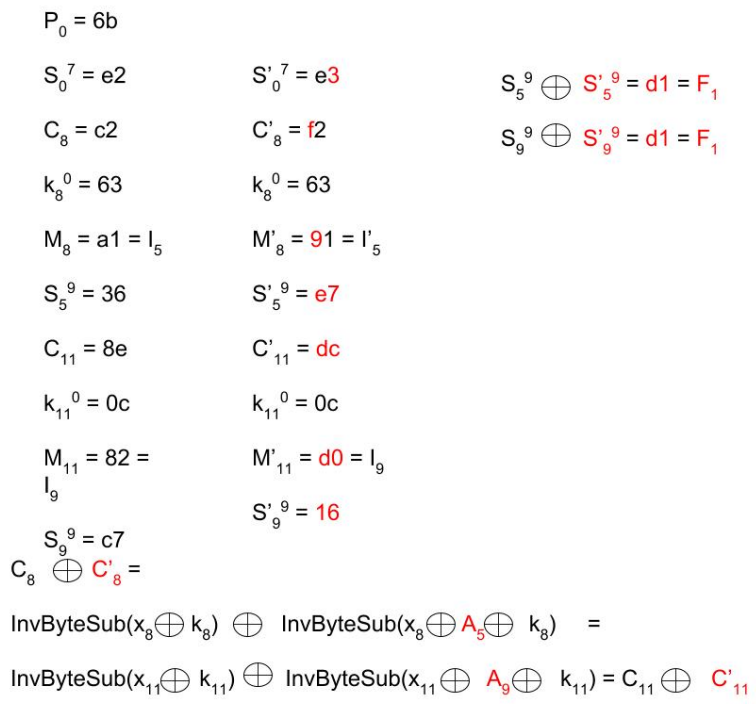


Figure 1.13. Single Fault DFA

$$S_0^8 + S'^8_0 = S_1^8 + S'^8_1 \tag{1.21}$$

$$S_0^8 + S'^8_0 = 3(S_2^8 + S'^8_2) \tag{1.22}$$

$$S_0^8 + S'^8_0 = 2(S_3^8 + S'^8_3) \tag{1.23}$$

After obtaining the faulty ciphertext, the attacker will attempt to derive the values for the first 4 bytes of S^8 and S'^8. She needs the values for 4 bytes of the 10th round key to reach guesses for the first byte of S^9 and S'^9 and one byte of the round key for the 9th round to reach guesses for the first byte of S^8 and S'^8. The attacker generates all possible values for these 5 key bytes and stores these values as well as the difference between the first byte of S^8 and S'^8 for 10 pairs of correct and faulty ciphertexts in a list. Afterwards, she repeats this process to store the possible key values and differences for the second byte of S^8 and S'^8 in a list. The attacker

then merges these lists and looks for collisions between the lists to determine which key guesses satisfy the equation (1.21). Afterwards, she repeats the process to satisfy (1.22) and (1.23).

1.3.2 Fault Sensitivity Analysis

Adversaries are able to determine a secret key without the values of ciphertexts using a technique called Fault Sensitivity Analysis (FSA) [13]. The adversary begins by running AES and getting the correct ciphertext. Next, they run AES repeatedly using the same plaintext and increasing the clock frequency at each iteration. After each iteration, they compare the resulting ciphertext to the correct ciphertext to determine whether or not a fault was injected. If there is a difference between the ciphertexts, the adversary records the clock frequency. After this, they are able to acquire the secret key byte-by-byte using statistical analysis because the clock frequency where faults begin to occur is data-dependent.

In 2010, Fault Sensitivity Analysis(FSA) was introduced as a fault analysis method that does not require the value of faulty ciphertexts[13]. Instead, FSA detects characteristics of a fault injection method that violate setup delay(increased clock frequency or reduced voltage) and treats them as an additional side channel.

In FSA, the attacker begins by running AES and getting the correct ciphertext. Afterwards, he runs AES repeatedly using the same plaintext increasing the fault injection intensity at each iteration. After each iteration, he compares the resulting ciphertext to the correct ciphertext to determine whether or not a fault was induced. If there is a difference between the ciphertexts, the fault intensity is recorded as the critical fault intensity. The attacker then repeats this for numerous plaintexts to get multiple critical fault intensities.

In order to launch an FSA attack, an attacker must understand how to use Hamming Weight(HW) to determine how close a circuit will be to the maximum timing delay. HW calculates the number of non-zero elements in a string. It is useful to the attacker because the critical fault intensity is highly correlated to the number of 0's in the signals sent to an S-box. This correlation exists because the value of inputs influences the maximum timing delay for an

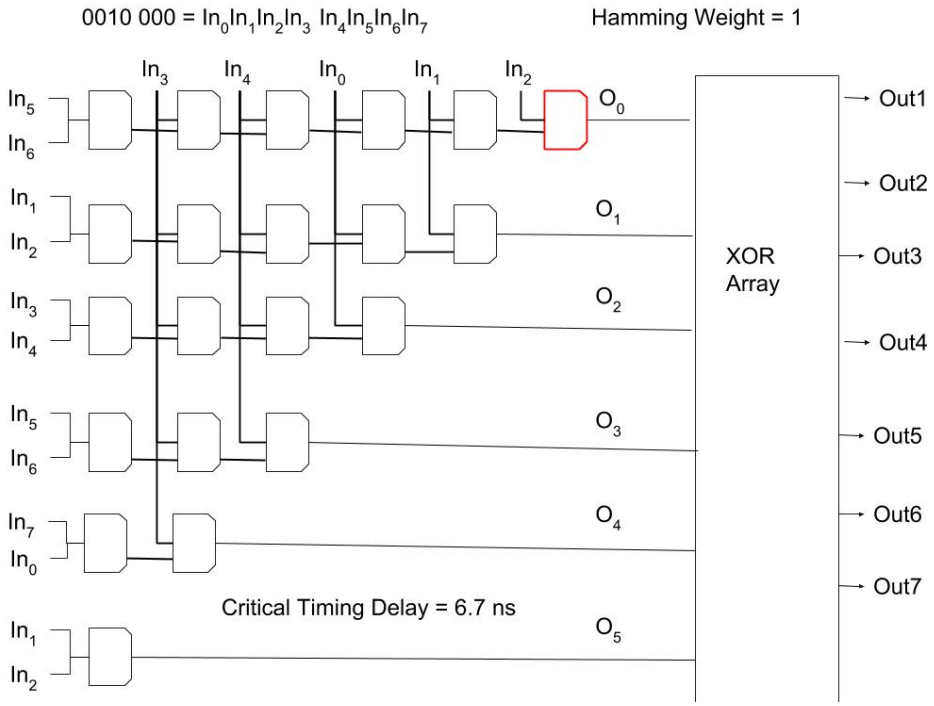


Figure 1.14. FSA example part 1

S-box. The maximum timing delay determines the threshold of timing delays that can occur without causing a setup time violation. The extent to which a circuit approaches the maximum timing delay depends on the values of the input signals. If the first input to an AND gate is a 0 then the output of the circuit is determined faster because both inputs must be 1 in order to output a 1. Since most S-box implementations include AND gates, each 0 in the input signal has the probability of decreasing the timing delay for the circuit and reduces the chance of a setup time violation making fault injection methods such as clock and voltage glitching more difficult.

As a result, the critical fault intensities for inputs with higher HW will be lower. An example is shown in Figure 1.14. A PPRM-1 S-box consists of the AND gate array shown and an XOR array. If the S-box is given the input byte 0x10 the HW will be 1 and there will only be 1 gate where the first input is 1. In all other gates the first input is 0 so they can be determined faster and the critical timing delay will be 6.7ns.

As a result, the critical fault intensities for inputs with higher HW will be lower. An

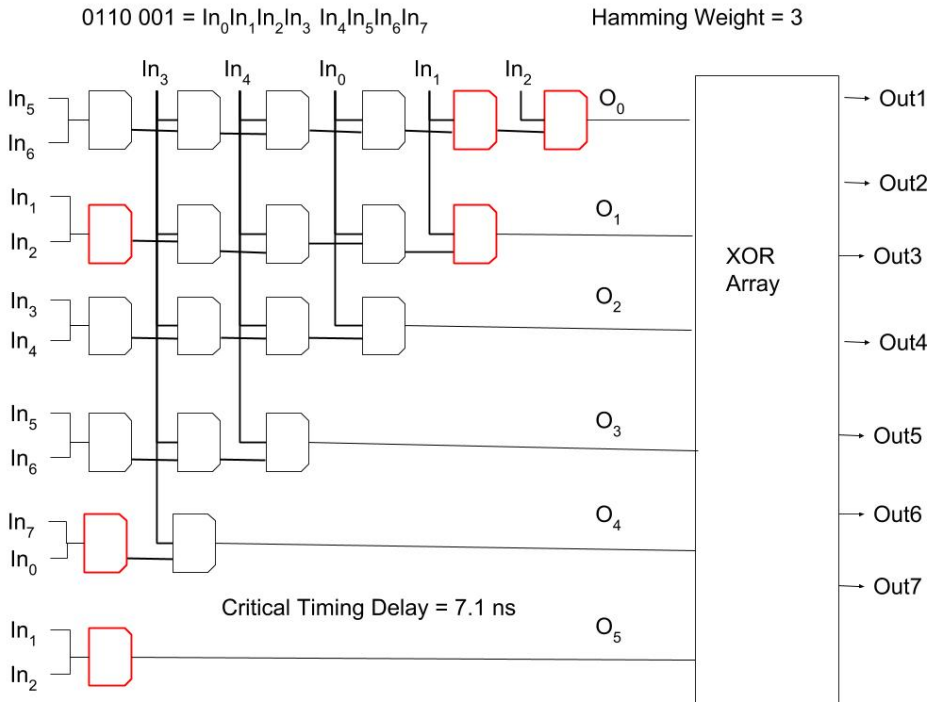


Figure 1.15. FSA example part 2

example is shown in Figure 1.14. A PPRM-1 S-box consists of the AND gate array shown and an XOR array. If the S-box is given the input byte 0x10 the HW will be 1 and there will only be 1 gate where the first input is 1. In all other gates the first input is 0 so they can be determined faster and the critical timing delay will be 6.7ns.

However, when the input byte is 0x61 the HW will be 3 and there will be 6 gates where the first input is 1 as in Figure 1.15. Since fewer gates have their first input as 0, the critical timing delay will increase to 7.1ns.

To find the key, first the attacker generates a guess of each byte of the key. Next, he exors the key guesses with each corresponding byte of the ciphertext and performs an inverse Byte Substitution operation on them. After this, he uses the Hamming Weight(HW) of this value to predict a guess value for the critical fault intensity.

The attacker repeats the same steps for each ciphertext using the same key guesses and calculates Pearson's correlation coefficient between the actual critical fault intensity and the

guessed critical fault intensity to determine the correlation for each key guess. The attacker repeats this process for each possible value for the key guess for each of the bytes and the key guess for each byte that has the highest correlation is the correct key guess for that byte. Since the values of the faulty ciphertexts are not required, countermeasures that stop execution or provide unusable output are not effective. However, masking which is not effective against DFA, is a useful countermeasure for FSA since FSA is data dependent.

1.3.3 Biased Fault Analysis

Adversaries are able to determine a secret key without using correct ciphertexts with a technique called Biased Fault Analysis (BFA) [56]. The adversary begins by running AES repeatedly with the same plaintext injecting different faults. Next, they will collect the resulting faulty ciphertexts and use key byte hypotheses to recover the state values where the fault was injected. Finally, the adversary distinguishes the correct key hypothesis from incorrect hypotheses by statistically analyzing the faulty state values. This type of analysis works because the correct key hypothesis will produce faulty states with similar state values while incorrect hypotheses will produce faulty states with large random differences.

In 2014, Differential Fault Intensity Analysis(DFIA) emerged as a new method of fault analysis[57]. DFIA uses faulty ciphertexts to derive an intermediate value and selects the most likely key hypothesis based on the fact that the wrong key hypothesis results in large random changes when faults are induced.

In DFIA, the attacker does not need the correct ciphertext so he runs the AES algorithm repeatedly with same plaintext inducing different faults. We will denote a specific fault injection with the subscript q. A specific fault injection will produce a faulty intermediate state $S`_q$ and subsequently $C`_q$. Hamming Distance(HD) gives a count of the difference between bit strings. Therefore, if the same number of bits are flipped in each computation, each faulty intermediate state will have the same HD with the correct intermediate state and the faulty intermediate states will be close to each other in HD. For example, assuming a 1 bit fault, S would have an HD of 1

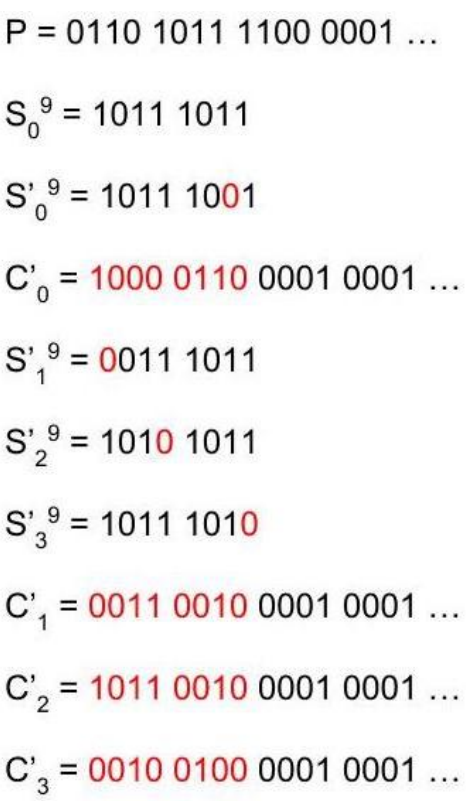


Figure 1.16. DFIA attack part 1

with each S'_q. Furthermore, each S'_q would have an HD of 2 with each other $S'q$.

To perform the analysis, first the attacker will generate faulty ciphertexts. Next, he will make a guess for a byte of the 10th round key. Using this key guess, he will perform Key Addition then inverse ShiftRows and Byte Substitution operations to derive a faulty state. Afterwards, the attacker calculates the HD between each faulty state and every other faulty state to determine the cumulative HD for that key guess. The minimum cumulative HD is the most likely guess for the correct key byte. The attacker then repeats this process for each key byte. Once all of the key bytes for the 10th round key have been acquired, the attacker is able to use the inverse key scheduling algorithm to acquire the entire 128 bit AES key. An example of this is shown in Figure 1.16

In this example the first byte of the plaintext is 0x6b in hexadecimal representation and by round 9 it has been changed to 0xbb. If the second bit from the right is flipped from 1 to 0

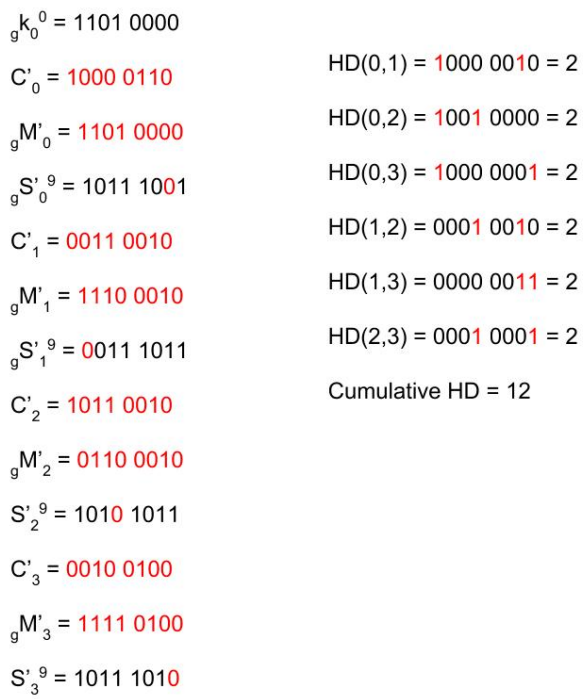


Figure 1.17. DFIA attack part 2

then the faulty intermediate state byte 0xb9. After computing the final round the first byte is 0x86 for the faulty ciphertext. If the leftmost bit is flipped the faulty state is 0x3b, if the fourth bit from the left is flipped the faulty byte is 0xab and if the rightmost bit is flipped the faulty state is 0xba. The resulting ciphertexts bytes are 0x32, 0xb2, and 0x24.

If the attacker correctly guesses that the first key byte is 0xd0 he will find that the intermediate states are 0x56, 0xe2, 0x62 and 0xf4 when the key byte is xored with each faulty ciphertext byte. After performing the inverse ShiftRows and Byte Substitution operations he will recover the faulty states bytes from the 9th round. The HD between each faulty state and every other faulty state is 2 giving a cumulative HD of 12 as shown in Figure 1.17.

1.3.4 Combined Fault Analysis

Adversaries are able to determine a secret key by combining fault analysis with power analysis in a technique called Combined Fault Analysis (CFA) [32]. The adversary begins by

running AES repeatedly with the same plaintext injecting different faults. Next, they will collect the resulting power consumption curves. Finally, the adversary correlates the power consumption curves to secret key hypotheses and uses statistical analysis to determine which key hypothesis is correct.

1.4 Threat Model

For all SCA, our experimental results target the attack of cryptographic algorithms running on a microcontroller. We assume the attacker is aware of when the computation of the cryptographic algorithm begins with microsecond level precision (e.g., by using simple power analysis (SPA)) [77].

For passive SCA, we assume the attacker has physical possession of the target device, is able to run security critical programs with arbitrary inputs, and is capable of collecting detailed power or EM traces. We make no assumptions about the equipment the attacker uses to collect traces.

For active SCA, we assume the attacker has physical possession of the target device, is able to run security critical programs with arbitrary inputs, and is capable of collecting faulty ciphertexts. We assume the adversary uses clock, glitching, voltage glitching or electromagnetic interference to inject faults because these techniques are non-invasive and require less than $3,000 worth of equipment making them well within the means of a single, motivated adversary [8]. We assume the adversary does not use focused light beams, lasers, or focused ion beams to inject faults because the fault injection techniques are too invasive and/or expensive to be practical. We assume an adversary does not use overheating to inject faults because this technique is not precise enough to inject faults during critical points in an algorithm's execution. We do not consider a decapsulation scenario because this requires an uncommon technical skill and is a highly invasive technique which leaves tamper evidence and could damage the chip.

1.5 Passive Side Channel Countermeasures

Some techniques add active equalization circuitry to diminish power variations during execution and keep the power supply constant [109, 110]. Other techniques use signal attenuation hardware to reduce the power cost of noise injection [37] or use a suppression circuit to reduce low frequency power variations and a low-pass filter to reduce high frequency power variations [128]. There are some ideas which provide internal power sources which an adversary cannot modify e.g., a charge-pump circuit using on-chip capacitors [119] and a switched capacitor circuit to isolate an AES core from the power supply line [145]. The disadvantage to these works is that they were not applied selectively in order to allow designers to make tradeoffs between performance, area, and security. While computational blinking [3] also identifies non-uniformity in information leakage, it implements a switched capacitor circuit rather than using signal attenuation hardware. As a result, applying the different techniques intermittently has different accommodations and requires different constraints.

1.6 Fault Analysis Countermeasures

1.6.1 Masking

Masking increases the amount of noise by adding randomization to eliminate the correlation between the secret key and intermediate values [13]. It XORs plaintexts and secret keys with a random value which changes each time the algorithm is executed. Masking can be used to mitigate FSA attacks because their method of recovering the secret key is the most data-dependent. DFA, BFA and CFA attacks are not data-dependent because they simply rely on the correct and fault ciphertexts to differ by the same amount and researchers have practically demonstrated their effectiveness on masked implementations of AES [18, 32, 117].

1.6.2 Time Redundancy

Time redundancy performs a step, round or the entire algorithm multiple times to check whether or not both executions match [88]. If a fault is detected, the system can stop execution so the adversary is not able to acquire faulty ciphertetxs.

Time redundancy can be used to mitigate DFA attacks because they are able to detect the differences between correct and faulty ciphertexts during the algorithm's execution and prevent the adversary from acquiring faulty ciphertexts [88]. It does not prevent BFA attacks because they do not require a correct ciphertext [117]. This creates the opportunity for an adversary to inject the same faults on both the original and redundant step to force them to match and bypass the error detection step. Time redundancy does not mitigate FSA or CFA attacks because they do not require the value of faulty ciphertexts; their analysis depends on the faults themselves rather than their effect on the final ciphertext [13, 32].

1.6.3 Error Detection Codes

Error detection codes use a check bit to determine whether or not a fault has occurred [88]. While error detection codes incur low performance and area costs, they have lower fault coverage than time redundancy. We do not compare error detection codes to isolation in our analysis because in many cases they are not able to detect faults well enough to protect against a reasonably motivated adversary.

1.6.4 CAMFAS

CAMFAS enables operation duplication using single instruction multiple data (SIMD) extensions of microprocessors [27]. SIMD extensions operate on wider registers to complete multiple operations in parallel. SIMD is usually used to exploit data parallelism available in a cipher through mapping the algorithm's statements into vector statements manually. Instead of running the original instruction and it's copy sequentially, CAMFAS vectorizes the instructions and packs them into a SIMD register for execution. Error checking is performed on the vectorized

instruction for fault detection.

CAMFAS can be used to mitigate DFA and BFA attacks by executing the original instruction and its redundant copy in parallel which prevents the adversary from injecting the same fault on both. CAMFAS does not work on FSA or CFA because they do not require the value of faulty ciphertexts [13, 32].

1.7 Contributions

The major contributions of this thesis are:

- Computational blinking is a programmable technique that switches between on-chip and off-chip energy sources to mitigate power information leakage. This thesis analyzes how clock speed and on-chip capacitance affect blinking and proposes stalling the processor to avoid information leakage while the on-chip energy source recharges. This analysis can be used to help designers determine the level of security they are able to afford based on area and performance costs

- Faults are purposefully injected by manipulating the clock signal or voltage fed to a chip. This thesis proposes mitigating this process by selectively isolating the computation when adversaries must inject faults to successfully launch fault attacks. We develop a representative sample of fault attacks according to difficulty in terms of effort for the adversary, to allow for designers to make tradeoffs between performance and security. This analysis can be used to help designers determine the level of security they are able to afford based on area and performance costs

- Signature aTtenuation Embedded CRYPTO with Low-Level metAl Routing (STELLAR) was proposed to mitigate power and EM-based attacks, but incurs 50% power overhead. This thesis reduces power overhead by operating STELLAR intermittently utilizing an intelligent scheduling algorithm. The proposed scheduling algorithm determines the

optimal locations during the crypto operation to turn STELLAR ON, and reduces power overhead compared to the normal STELLAR operation, while eliminating power and EM information leakage. This analysis can be used to help designers determine the level of security they are able to afford based on area costs

Chapter 2

Introduction

The idea of computational blinking derives from the physiological act of blinking in humans. Humans blink 15-20 times per minute for a duration between 100-400 ms. Thus, they spend between 2.5-13.3% of their waking time with their eyes closed due to blinking. Furthermore, sections of our brain their brains are actually momentarily "powered off" during each blink. Yet, they are rarely even aware of these near continuous interruptions. These spontaneous blinks occur at natural breakpoints when attention is least needed, e.g., during a pause when listening to a speaker or at a scene change in a video. In the same way, computational blinking attempts to turn mitigation strategies on when an encryption algorithm is vulnerable to side channel attacks and turn them off when an encryption algorithm is least vulnerable to side channel attacks. We achieve this through a workflow with a security phase and a design exploration phase. In the security phase, the designer considers an encryption algorithm such as AES or PRESENT, implements it on a device and uses a metric to determine which points in the algorithm's execution are most vulnerable to a side channel attack. We then use this metric to rank the vulnerability of the execution cycle by cycle. In the design exploration phase, we determine the best times to turn a blinking mitigation strategy on or off by considering when it is most beneficial to utilize the mitigation technique, how much overhead a hardware designer is willing to sacrifice for security, and the constraints for when it is possible to turn a mitigation strategy on or off

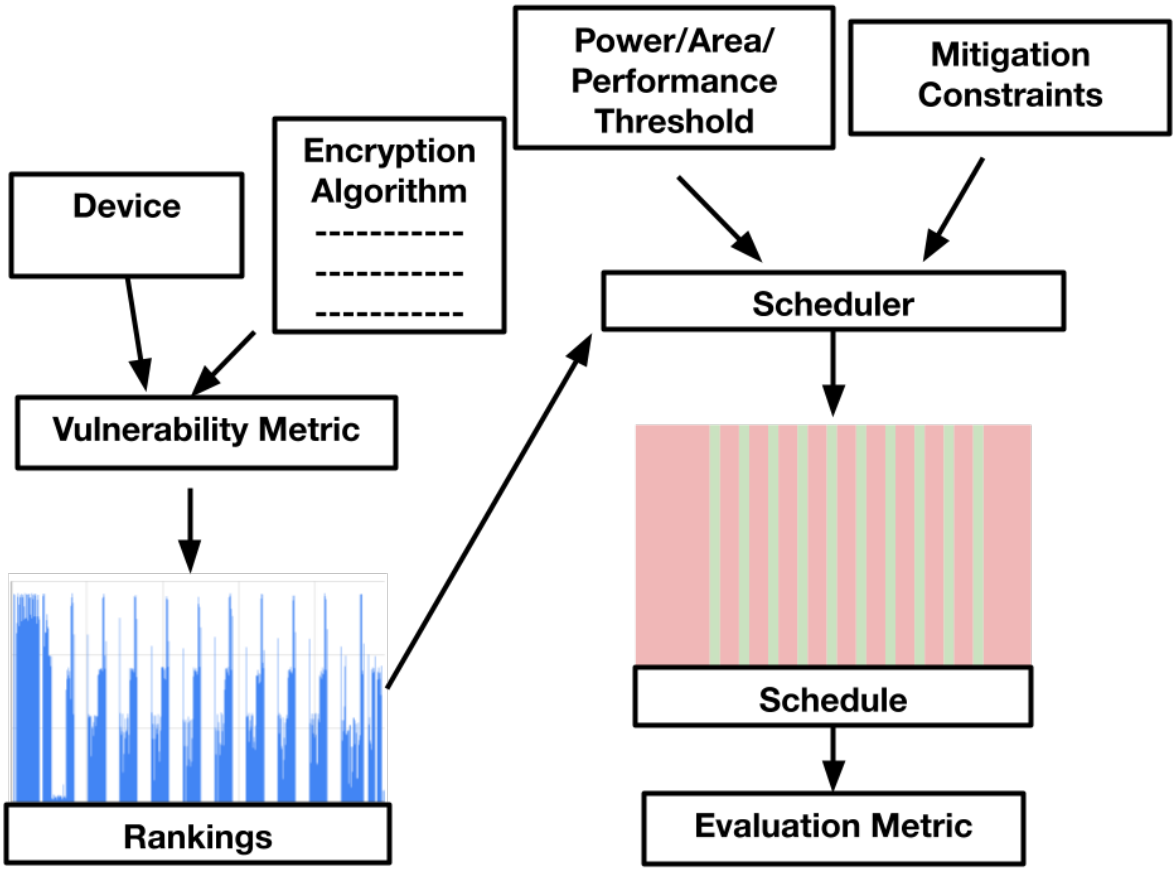


Figure 2.1. In the security phase, uses a vulnerability metric to determine which points in the algorithm's execution are most vulnerable to a side channel attack cycle by cycle. In the design exploration phase, the vulnerability rankings from the security phase, power, performance or area thresholds defined by the designer and the the proposed mitigation strategy's constraints are used as inputs to a scheduling algorithm that determines the best times to turn the mitigation strategy on or off. Afterwards, an evaluation metric is used to evaluate the effectiveness of the mitigation strategy.

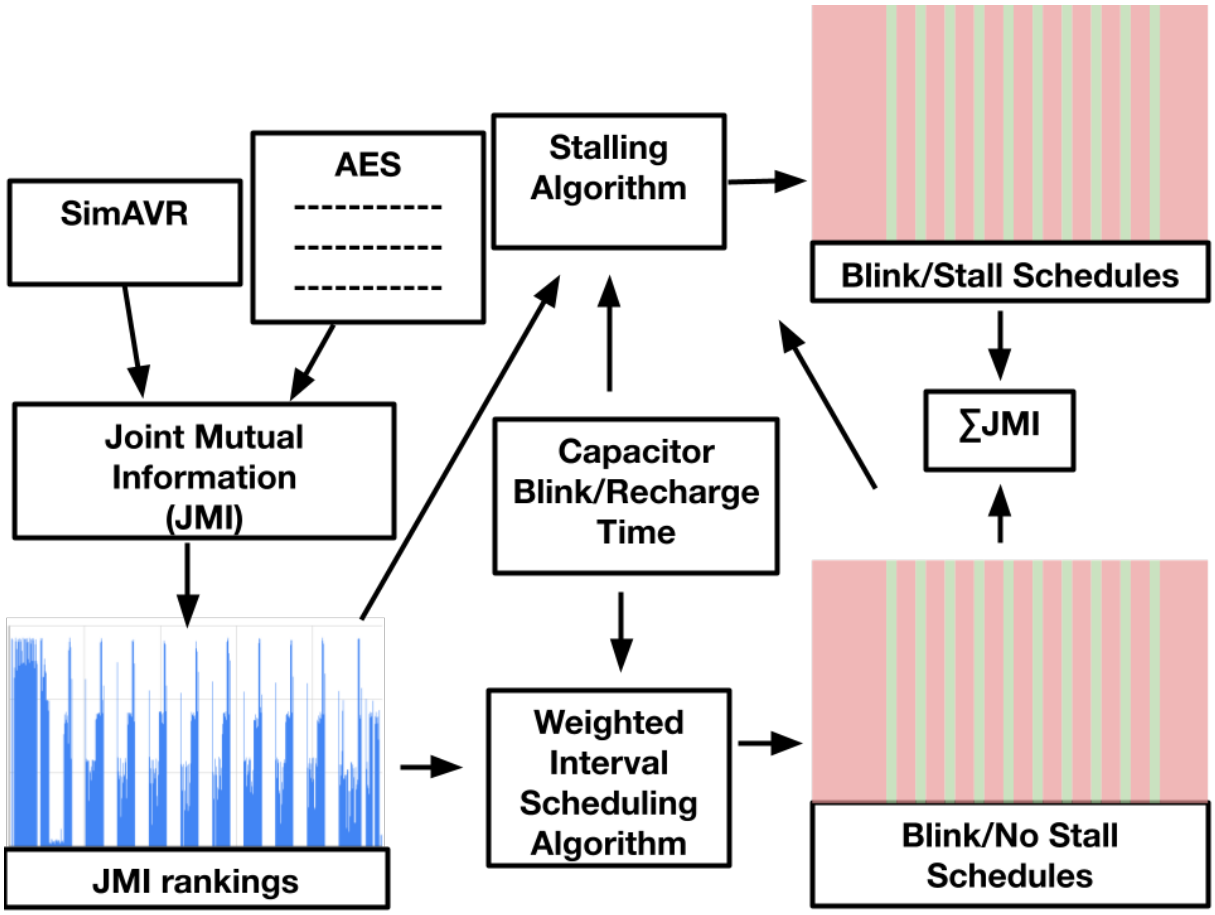


Figure 2.2. The stalling algorithm improves upon the blinking algorithm by making tradeoffs among the stalling parameters (security, performance and area) and isolating portions of an algorithm's execution left unprotected while an on-chip energy source recharges in three phases. In the security phase we generate joint mutual information (JMI) rankings to determine which important parts the algorithm's execution reveal information about the key. In the first part of the design exploration phase, we take the JMI rankings from the security phase and the blink and recharge times of the on-chip capacitor as inputs to a weighted interval scheduling algorithm. This algorithm generates blinking schedules which do not use stalling to determine when the capacitor should be turned on(shown in red) and when it should be turned off(shown in green). In the second part of the design exploration phase, we take the remaining JMI rankings after implementing our initial blinking schedule as well as a performance threshold as inputs to a stalling algorithm which extends the initial blinking schedule by giving the capacitor more time and isolate more computations. Afterwards, we measure the effectiveness of the new blinking schedule by calculating the sum of the remaining JMI rankings.

Chapter 3

Power Analysis Mitigation

A number of system-level physical countermeasures have been proposed to thwart these attacks [128, 109, 110, 119, 37, 145]. Typically, attackers target a particular point in time, e.g., one that leaks significant information about the key and is easy to subsequently derive the bits of the key. The amount of useful information about a key varies radically across the encryption computation [2]. Thus, techniques that protect all portions of the computation incur larger than necessary overhead. Furthermore, software implementations require flexible and programmable techniques to protect cryptographic implementations whose code can be updated to include more advanced mitigation techniques like masking [34, 12, 113, 64].

Computational blinking addresses these issues by making the on-chip energy source programmable and using it to isolate the computation at the most critical points in time [3]. One can then use a smaller, less expensive energy source to provide isolation. During a blink, the computation is fully powered from an on-chip energy source and draws no energy from the off-chip power supply. This eliminates the power side channel. However, the on-chip source stores a finite amount of energy and thus can only power the computation for a fixed amount of time. When the energy is depleted, the on-chip power source must be recharged before it can be used again.

A fundamental challenge is deciding when to perform the blinking. We attempt to position blinks in a manner that minimizes information leakage to develop a blink schedule: the

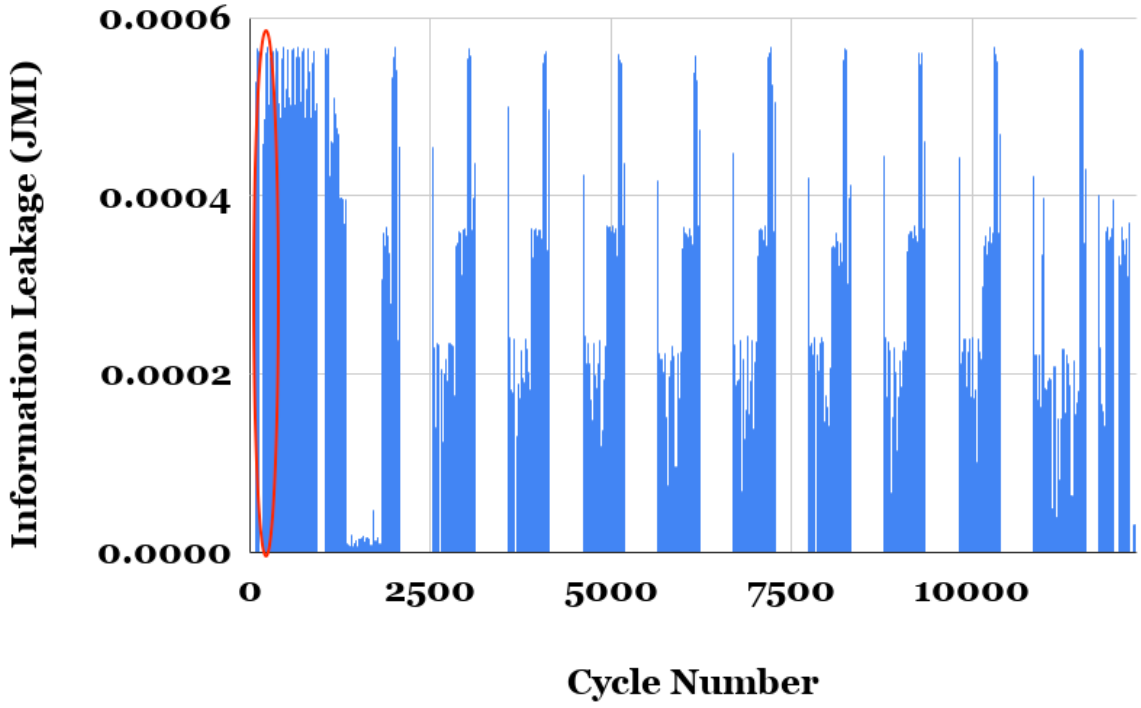


Figure 3.1. The joint mutual information (JMI) of a subset of a power trace collected from an AES-128 implementation from DPA Contest v4.2 [30]. The x-axis shows the time (cycle number) and the y-axis the JMI, which corresponds to the amount of information leakage. Cycles that have larger JMI reveal more information about the key.

positions of a set of blinks. Althoff et al. [3] developed an optimal blink scheduling algorithm under the assumption that the execution time is constant. However, even with an optimal schedule, information leaks during the recharge times. We propose stalling the processor during these recharge times and using another blinked computation to avoid information leaks. Although stalling increases the number of instructions by adding noops, we are able to develop blinking schedules which improve security without sacrificing performance by placing stalls strategically. We achieve this by utilizing stalling to allow blinked computations during times with higher leakage that normally occur during recharge times.

We can further increase security by increasing the amount of on-chip capacitance. This allows the on-chip capacitor bank to power computations for a longer amount of time before it must recharge and makes it possible to isolate more instructions with a single blink. However,

increasing on-chip capacitance incurs additional area costs. In this work, we explore the tradeoffs among the stalling parameters (security, performance and area) to evaluate blinking and stalling schedules as shown in Figure 2.2.

3.1 Switched Capacitor

It is possible to use a switched capacitor to use on-chip energy to power a critical computation for a well-defined but very short duration of time. Since this computation does not draw power from an external energy source, it eliminates the off-chip power side channel corresponding to the execution of this computation. We focus on providing a programmable, flexible energy usage model for a general execution core running any software such as encryption, attestation, or a high-assurance piece of code containing sensitive data. This general framework enables programmers and system designers to perform a computational blink and mask intermediate energy usage over a fixed amount of time, either eliminating or greatly reducing the information leakage through this channel

The blinking process is depicted in Figure 3.2. The core begins by executing instructions while connected to the external power supply V_{DD}. When the core reaches a point where its power consumption should be hidden, a blink is initiated and a power control unit disconnects the chip from its main power source by turning off the blink and recharge transistors. As a result, the chip is only connected to the capacitor bank's internal power rails. The core then executes sensitive operations while connected to the on-chip capacitor bank to hide its power consumption from the off-chip power monitoring. Afterwards, the capacitor bank is always discharged to a fixed minimum level (V_{min}) by using a shunt resistor to open the discharge transistor.

Discharging the capacitor before recharging is done to eliminate all residual energy before reconnecting the core and capacitor to the external power supply. Sets of instructions will use different amounts of energy during a blink, but discharging avoids information leakage by ensuring the capacitor is always at V_{min} by the end of a blink. Thus, recharge energy is always

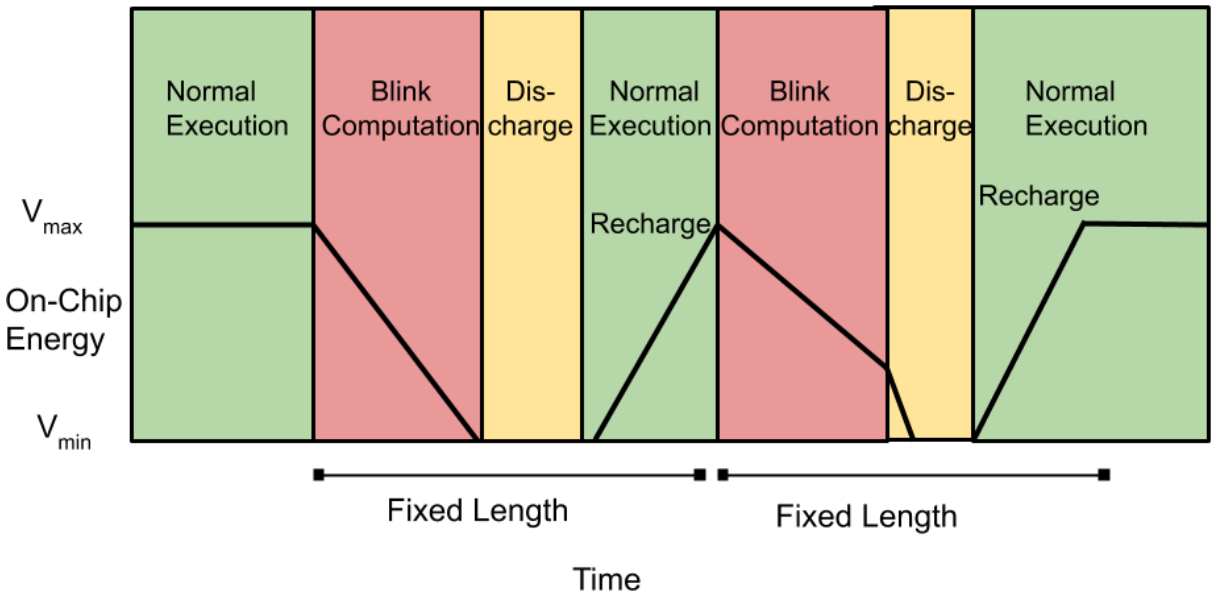


Figure 3.2. Recharge periods force some instructions to be executed using the external power supply, thus leaking information to an attacker. Figure reproduced from [3].

the same regardless of the computation and the amount of energy used on instructions during a blink cannot be determined by an adversary. Furthermore, cycles are spent on a discharge step even if the chip is already at V_{min} to ensure that the blink/discharge/recharge sequence occurs at a fixed length. We always perform the discharge step to avoid introducing a timing channel.

Once a blink is completed, the on-chip capacitors must be fully recharged, because another blink cannot occur until this recharge time is complete. Ordinarily, during the recharge time, the core must operate from the external power supply. This makes its activity vulnerable to a power attack.

3.2 Joint Mutual Information

We use JMI to rank time periods within the traces and these rankings are used to determine when we turn the switched capacitor On/OFF [104, 150]. We argue for the use of joint mutual information(JMI) because it addresses the issue of variable complementarity[68]. As an example, consider a scenario where x1 XOR x2 = y under the assumption that x1 and x2 are statistically independent Boolean variables [2]. In this scenario, the mutual information between x1 and y =0

and the mutual information between x2 and y is 0. However, if we concatenate x1 and x2 ,the mutual information between the concatenation of x1 and x2 and y is ¿ 0 because the information from these Boolean variables completely determines y. Similarly, if a side channel attack security evaluation metric such as a t-test determines the time index of a security-sensitive algorithm has low vulnerability by considering just that index, combining functions or multivariate histograms with a few time indices, it may still be attractive to an adversary if considered alongside additional time indices as demonstrated in [61].

It is not possible to launch an attack if JMI = 0 because if this is true, the measurements with differing secret key hypotheses is always equal[3]. Therefore, it is impossible for an attacker to differentiate between different secret key hypotheses given sets of measurements.

We use joint mutual information (JMI) to rank time periods within the traces and these rankings as well as blink size and recharge size to develop a blinking schedule using a weighted interval scheduling (WIS) algorithm. This algorithm ranks time indices based on joint mutual information (JMI) [104, 150] as shown in equation 3.1.

$$JMI_i = \sum_{j \in B} I(f(t_i, \hat{m}, \hat{s}) \frown f(t_j, \hat{m}, \hat{s}); \hat{s}) \tag{3.1}$$

$a \frown b$ calculates the concatenation between a and b, and f is a function used to represent a power trace. The power trace takes an independently and uniformly random message and secret key from vectors $\hat{m}$ and $\hat{s}$. I(C;D) determines the mutual information for a and b which determines how much we can learn about C based on how it is related to D. It is calculated using equation 3.2 where H(C) is the entropy of C and $H(C|D)$ is the conditional entropy of C given D. B is the set of indices (i) we have chosen to blink.

As indices are added to B they are given a ranking based on their JMI. The index with the greatest mutual information with the key will be ranked highest and would be selected first to be blinked. Next, every time index redundant to this index in the algorithm would be given the same ranking because they would provide an equal amount of information. The algorithm repeats this

process with successive time indices ranked according to how much easier they would make it for an attacker to recover the key. After all of the time indices have been ranked, the JMI scores are normalized so the sum of all JMI rankings is 1.

$$I(C;D) = H(C) - H(C|D) \tag{3.2}$$

After the indices have been ranked, a WIS algorithm takes the amount of time the chip is able to blink and the amount of time necessary to recharge to optimally select which time periods to blink. First, it creates a list of every potential blink and calculates the amount of coverage each blink would provide from the JMI rankings. Next, it determines which blinks can occur in the same schedule by matching blink start points to the nearest potential blink end point. Finally, it selects the blink schedule that minimizes the distance between blinks and maximizes the leakage that is blinked.

After implementing the blinking schedule, we measure its effectiveness by the sum of our remaining JMI rankings as an evaluation metric because JMI provides a measure of multivariate vulnerability. As a multivariate metric, it measures the usefulness of each time region for an attack when combined with information from other time regions. Multivariate security is important because some time regions may be independent from the key individually, but may allow an adversary to recover the secret key easily when combined with additional information from the trace. This type of vulnerability has been exploited in attacks such as [61].

3.3 Stalling

Figure 3.1 provides an example of a scenario where stalling addresses an information leak that blinking is not able to eliminate. It shows information leakage of a software implementation of AES-128 (obtained from the DPA Contest v4.2 [30]) running on an AVR microcontroller. The leakage is computed using JMI; samples of the power trace that have larger JMI values reveal more information about the secret key.

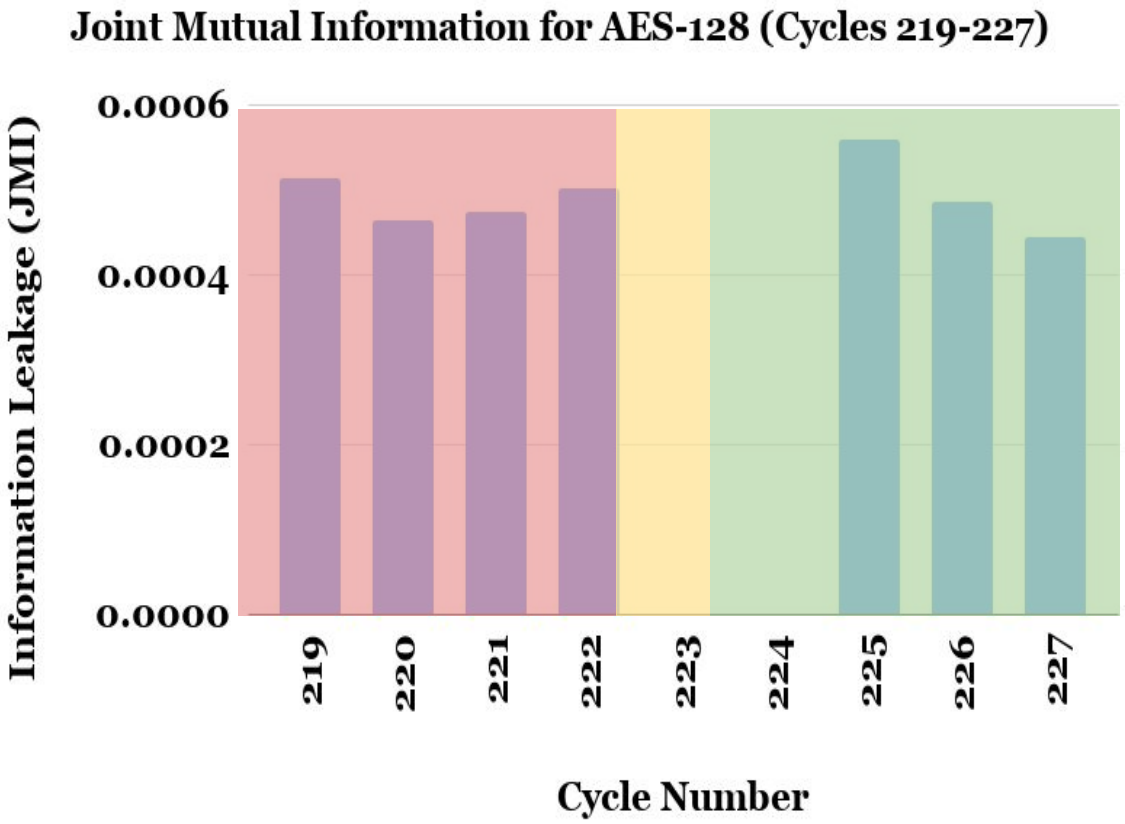


Figure 3.3. By blinking (shown in red), we can hide parts of the trace, which effectively eliminates the leakage, i.e., JMI = 0. But we must recharge (shown in green) after a blink, which leaves those portions of the computation visible to an attacker.

Figure 3.3 performs more detailed analysis on the cycles circled in Figure 3.1 to show an example blinking schedule that uses four cycle blinks. The cycles covered in red represent parts of the power trace hidden by blinking and the cycles covered in green represent parts that are left uncovered due to recharge periods. Cycles 219 to 222 are blinked out because they are the four cycles with the highest cumulative JMI. Unfortunately, after blinking for four cycles it is necessary to spend twelve cycles recharging before it is possible to perform another blink that can hide four cycles. Although we do not need to blink for cycles 223 and 224 because they do not leak any information, cycles 225-227 have high JMI and are left unprotected because blinking them would prevent us from blinking cycles 219 to 222.

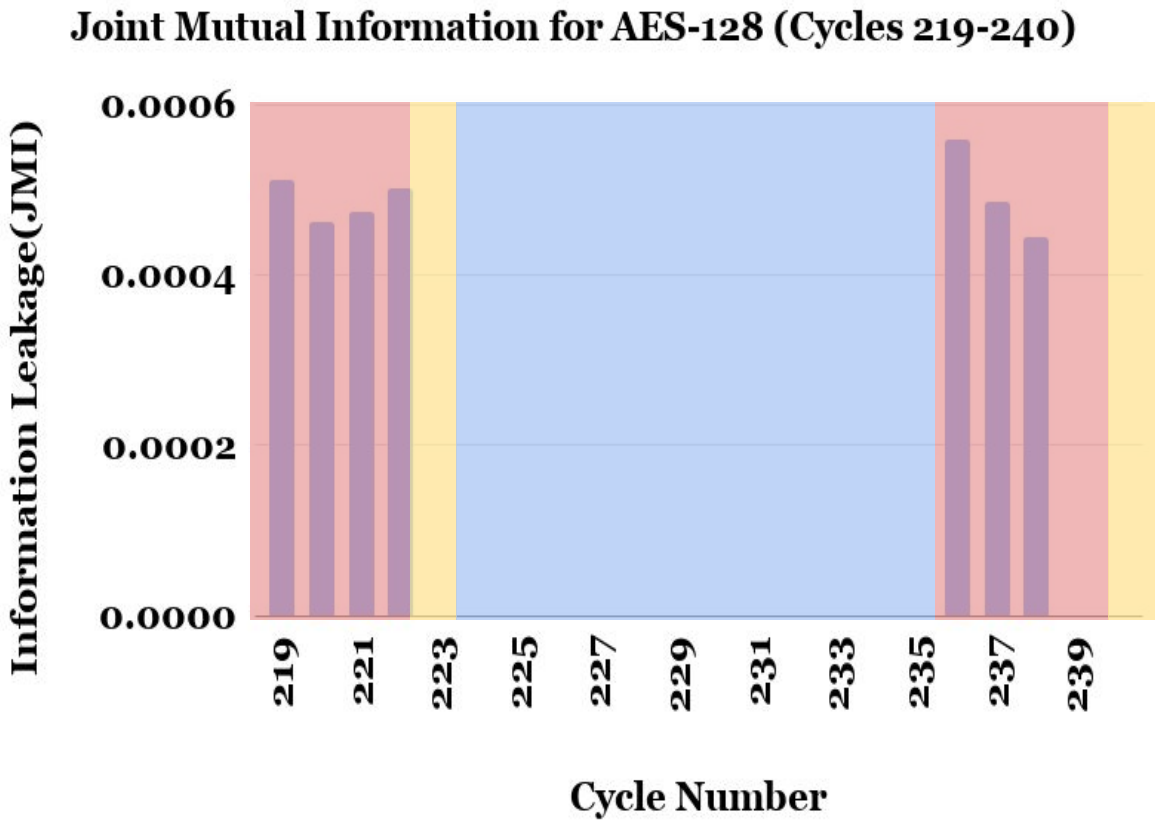


Figure 3.4. By stalling the processor (shown in blue) to recharge, we are able to blink again and hide parts of the trace we missed in our initial blink schedule

Figure 3.4 illustrates how this issue can be resolved by stalling. We are able to blink the leakage from cycles 219 to 222 as well as cycles 225-227 by stalling the processor for ten cycles after cycle 224. We also see that the first blink computation uses all of the on-chip energy while the second blink computation does not. This is because we always include a discharge time although the capacitor is already at V_{min} to ensure the blink/discharge/recharge time for both blinks take sixteen cycles.

3.3.1 Stalling Process

By stalling the core's execution during recharge times, we can effectively perform two blinks in a row, at least with respect to the execution of the core's operation as shown in Figure

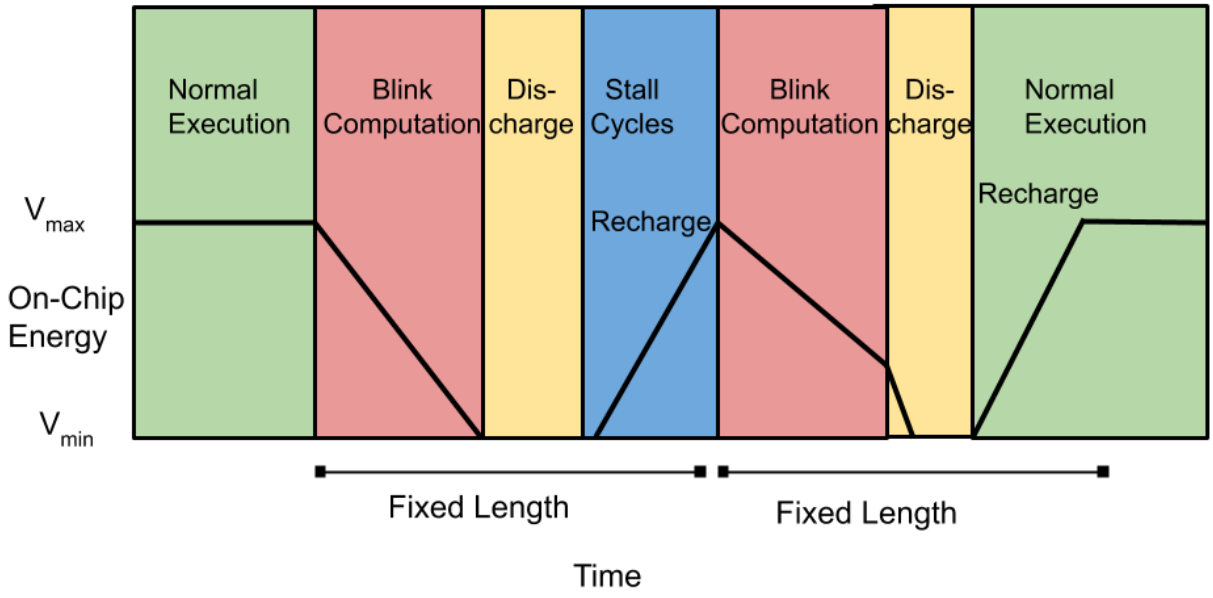


Figure 3.5. By stalling the processor, we are able to blink during more of the algorithm's execution making it more resilient to power side channel attacks.

3.5. We stall during the "blue" time period while performing the recharge. This increases security by enabling us to blink during more of the core's operation time and consequently reducing the amount of information leakage. However, it comes at the cost of decreasing the performance as the core's execution time increases due to stalling.

We perform stalling in a manner that is consistent across *any and all* executions of the cryptographic algorithm to avoid introducing a new timing side channel. The stalling schedule is determined before the execution starts. It is the same regardless of the data that is being computed, and it never changes the time, locations or the number of stalls. **Thus, this does not introduce a timing channel.** If the stalling was done in a dynamic manner or not enforced consistently across every execution of the cryptographic algorithm, it would very likely introduce a timing side channel. This is not the case with our proposed techniques.

3.3.2 Stalling Algorithm

In order to further improve security from blinking, we propose using the stalling algorithm shown in Algorithm 1. This algorithm attempts to address the instructions that the algorithm

from [3] was not able to hide from the attacker. It uses a sliding window to determine which sequence of cycles has the highest leakage according to their JMI rankings. The algorithm schedules a blink for this sequence of cycles and determines the execution time and security of the new power trace. This process repeats until an execution time threshold has been reached.

Algorithm 1. Stalling Schedule

Input:
length-n vector **r** of JMI rankings after blinking,
execution time **E** after blinking and
constants **blinkTime**, **rechargeTime**, **threshold**
Output:
Execution time **E** and **leakage**

1: **procedure** STALL
2: $leakage \leftarrow \sum_{i=1}^{n} r_i$
3: **while** $E < threshold$ **do**
4: **for each** $i \in [n]$ **do**
5: $start \leftarrow i$
6: $end \leftarrow blinkTime$
7: $findMax_i \leftarrow \sum_{j=start}^{n} r_j$
8: $newBlink \leftarrow \max(findMax)$
9: $k \leftarrow newBlink$
10: **while** $k < blinkTime$ **do**
11: $r_k \leftarrow 0$
12: $k \leftarrow k + 1$
13: $E \leftarrow E + rechargeTime$
14: $leakage \leftarrow \sum_{i=1}^{n} r_i$

If security is the highest priority, designers should aim to ensure JMI is 0. They would begin by choosing the optimal blinking schedule for their capacitance value and clock speed, then perform stalling until all cycles with a mutual information value greater than 0 have been blinked. In this case, an adversary is not able to ascertain any information about the secret key from power traces and will not be able to acquire the secret key using power analysis.

If performance is the highest priority, designers could choose not to implement blinking at all. In this case, JMI would be 1 and the entire power trace would be vulnerable to power analysis attacks.

If designers want to balance security and performance, they would begin by choosing the optimal blinking schedule for their capacitance value and clock speed. **If large peaks in mutual information still exist using the optimal blinking schedule due to recharge constraints, designers can perform stalling to eliminate them.**

3.4 Stalling Parameters

The effectiveness of stalling is directly dependent on the amount of time we are able to blink and the amount of time we must recharge after blinking. This is because longer blink times allow us to hide more of the power consumption with each additional blink and longer recharge times increase the performance penalty we incur from stalling. Section 3.4.1 describes our blink and recharge time calculations and Sections 3.4.2, 3.4.3 and 3.4.4 outline the factors which affect our blink and recharge time calculations.

3.4.1 Blink and Recharge Time

Designers will increase security the most by blinking as often as possible. However, this is limited by the maximum blink time and the recharge time. The maximum blink time is the number of cycles that a blink can cover [3] and the recharge time is the number of cycles necessary to recharge the capacitor bank to perform another blink of the same size.

The number of cycles spent performing instructions from the capacitor bank being full to operating at V_{min} is the maximum blink time. We compute the maximum blink time using the per instruction energy (C_L), storage capacitance (C_S), nominal operating voltage (V_{max}), and minimum operating voltage (V_{min}). The equation is shown below:

$$MaximumBlinkTime = \frac{2 * log(\frac{V_{min}}{V_{max}})}{log(1 - \frac{C_L}{C_S})} \tag{3.3}$$

The number of cycles spent recharging the capacitor from V_{min} back to its initial capacitance is the maximum recharge time. Energy is potentially wasted in every blink, as excess

charge in the capacitor must be shunted to avoid leaking information. From our power simulations, the most energy-intensive instructions consume 1.6x the energy of an average instruction. Provisioning for the worst case, we account for wasting 60% capacitance and increase our recharge time accordingly. The equation is shown below:

$$RechargeTime = \frac{2 * log(\frac{V_{min}}{V_{max}})}{log(1 - .6 * \frac{C_L}{C_S * 1.6})} \tag{3.4}$$

For a given capacitor bank size, the recharge resistor can be tuned to achieve the desired recharge time; however reducing the recharge resistor (and thus the recharge time) will cause a larger in-rush current which at a certain point can corrupt the data inside the secure core [75]. For safety, we ensure that the in-rush current is no greater than peak current drawn from the secure core during normal operation.

3.4.2 On-Chip Capacitance

On-chip capacitance impacts security because increasing it allows for longer maximum blinks times at higher clock speeds. By increasing on-chip capacitance, designers can minimize power side channel leakage for much lower performance degradation.

Designers can increase on-chip capacitance by introducing additional on-chip decoupling capacitors [122]. However, introducing more capacitors requires more area. As a result we consider two scenarios with differing amounts of capacitance.

3.4.3 Clock Speed

Clock speed impacts security because it affects the maximum blink time for a capacitance value. Increasing the clock speed reduces the maximum blink time and decreasing it allows for larger maximum blink times. Unfortunately, decreasing the clock speed degrades performance. For example, when operating at 290 MHz, V_{min} is 1.8 V, but when operating at 70 MHz V_{min} is 1 V.

3.4.4 Multiple Blink Times

While having a long maximum blink time is desirable, it is not always necessary or advantageous to use the maximum blink time. In light of this, we also use a half length or quarter length blink times when possible to drain less capacitance and recharge faster. However, in order to perform half length and quarter length blink times, our maximum blink time must be long enough to allow it.

We assume that we must blink for an entire cycle, and thus, we require at least a maximum blink time of two to have a half length blink time and we require at least a maximum blink time of four to have a quarter length blink time. Different blink times do not drain the capacitor equally, but blinks of the same length are discharged to the same level of capacitance. As a result, there can be one, two or three V_{min} thresholds.

3.5 Results

We demonstrate the effectiveness of stalling by comparing optimal blink schedules to stalling schedules. We define our experimental setup in Section 3.5.1, we calculate our maximum blink and recharge times in Section 3.5.2, we describe our stalling algorithm in Section 3.3.2 and we perform design exploration in Section 3.5.3 to find the best stalling schedules.

3.5.1 Experimental Setup

We develop power traces using SimAVR to simulate an Atmel ATmega328 chip[121, 74]. SimAVR is capable of executing binaries compiled by the avr-gcc toolchain as they would be run on an AVR microcontroller and we use it to collect power traces using a Hamming distance leakage model [19]. To perform our evaluation, we collect power traces for 2^{14} experimental plaintext and secret key vectors on an implementation of AES-128 from DPA Contest v4.2 [30]. Under this model, each time point in a trace consists of the difference in Hamming distance between an opcode and its predecessor for different experimental plaintext and secret key vectors

[3]. For our information leakage model we assume that toggling a bit consumes one bit of power and leaving a bit unchanged consumes no power. Our information leakage evaluation is independent of the instruction type or the type of the data.

We consider two different scenarios for capacitance. In our first scenario, we were able to achieve 11 nF of storage capacitance on TSMC 180nm by filling $2.34mm^2$ of the $25mm^2$ die area with decoupling capacitors. Alternatively, in our second scenario we were able to achieve 22 nF of storage capacitance by filling $4.68mm^2$ of the $25mm^2$ die area with decoupling capacitors.

3.5.2 Blink/Recharge Time Calculations

Figure 3.6 shows the maximum blink times and recharge times for clock speeds ranging from 70MHz to 280MHz with 11 nF and 22 nF on-chip capacitance banks. It visually demonstrates how the maximum blink/recharge times increase when the clock speed decreases or the capacitance increases.

We are able to determine the maximum blink time and recharge time for a capacitor bank by selecting a clock speed. Using vector power simulations of a 32-bit RISC-V processor in TSMC 180nm, we found that each instruction consumes an average of 513 pJ which requires 317 picofarads (pF) to store (C_L). When using 11 nF of capacitance, our C_S value is 11 and if we run at 280MHz, our V_{min} value will be 1.75V and our V_{max} value is 1.8 because the nominal voltage for the core in our 32-bit RISC-V processor is 1.8V. This makes the maximum blink time only one cycle and we must recharge for four cycles. If we decrease the clock speed to 70MHz, our V_{min} value will be 1V allowing a maximum blink time of 24 cycles. However, if we blink for 24 cycles we will have to recharge for 68 cycles.

When using 22 nF of capacitance, our C_S value is 22, and if we run at 280MHz our V_{min} value will be 1.75V. This makes the maximum blink time only two cycles and we must recharge for seven cycles. If we decrease the clock speed to 70MHz, our V_{min} value will be 1V. This allows a maximum blink time of 50 cycles. However, if we blink for 50 cycles we will have to recharge for 136 cycles.

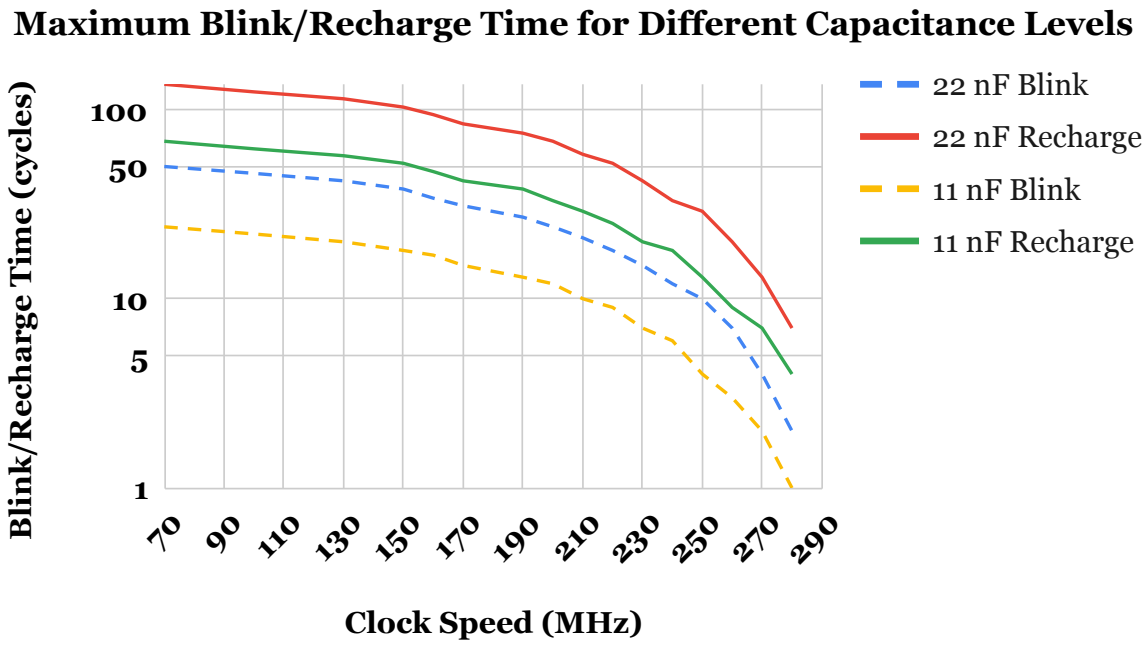


Figure 3.6. Greater capacitance allows larger blink sizes. Higher clock frequencies have stricter requirements on V_{min} and thus decrease the blink and recharge times.

3.5.3 Design Exploration for Stalling

Figure 3.7 outlines how we evaluate our stalling schedules. Each stalling schedule can be divided into three sections: "no blink/no stall", "blink/no stall" and "blink/stall". The "no blink/no stall" section is a single point which shows the normalized execution time(NET) without implementing blinking at a given clock speed; in this case JMI is always 1 and the entire power trace is vulnerable to power analysis attacks. The "blink/no stall (x,y,z)" section is a single point which shows the NET and JMI score after implementing the optimal blinking schedule for blink times x, y, and z without stalling to perform additional blinks. The "blink/stall (x,y,z)" section is a set of points showing the new NETs and JMI scores after stalling to perform additional blinks. When selecting a stalling schedule, we seek to obtain optimal performance or security for the "blink/no stall (x,y,z)" and "blink/stall (x,y,z)" sections. Once the best stalling schedule has been selected, all other potential stalling schedules will be discarded and the schedule will be constant for all executions of the algorithm regardless of the input data to avoid introducing a timing

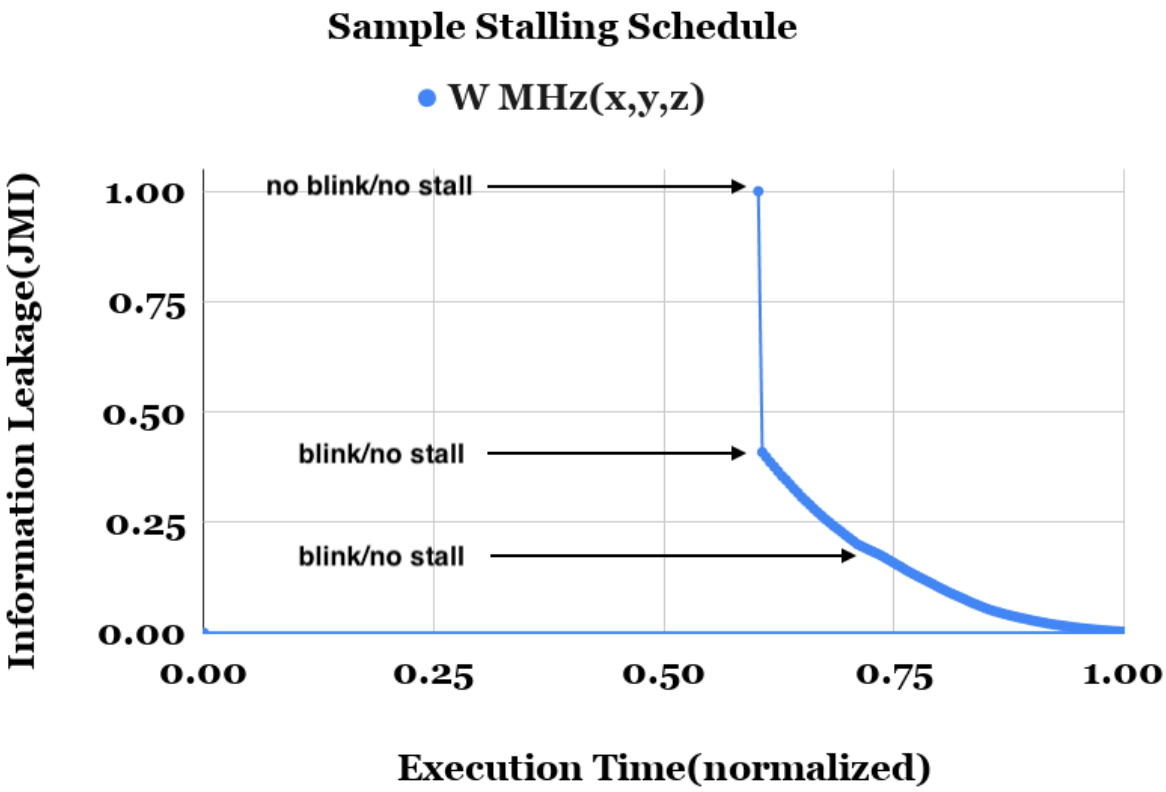


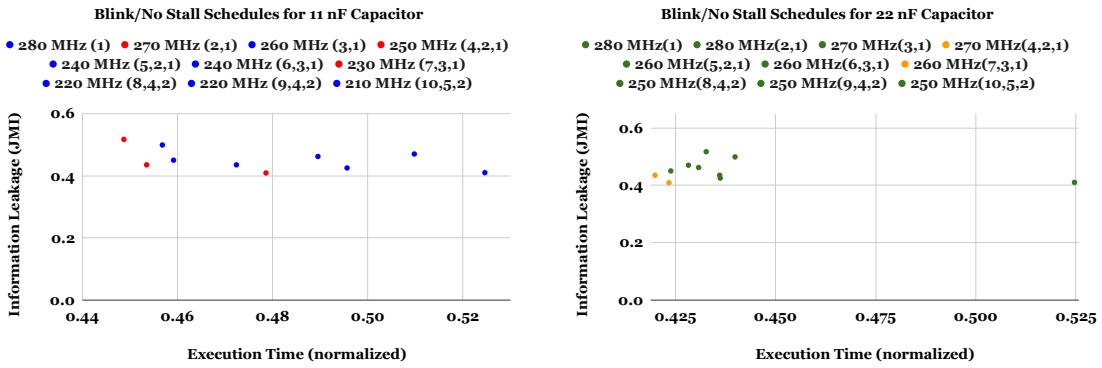
Figure 3.7. Stalling Schedules can be divided into "no blink/no stall", "blink/no stall" and "blink/stall". "no blink/no stall" does not performing blinking at all, "blink/no stall" implements the algorithm in [3] and "blink/no stall" allows us to add more blinks to improve security

channel.

Figure 3.9 shows stalling schedules for both long and short blink times on a 11 nF capacitor bank and a 22 nF capacitor bank. Section 3.5.3 evaluates blink times on a 11 nF capacitor bank, and Section 3.5.3 evaluates short blink times on a 22 nF capacitor bank. We evaluate these stalling schedules in terms of performance and security.

Blink Times with 11 nF Capacitor Bank

Figure 3.8a provides an extreme example of how slowing down clock speed to increase maximum blink time can dramatically degrade performance. "blink/no stall (10,5,2)" uses blink times that are ten, five and two cycles long. Unfortunately, "blink/no stall (10,5,2)" only



(a) "blink/no stall" schedules when using 11 nF capacitor bank

(b) "blink/no stall" schedules when using 22 nF capacitor bank

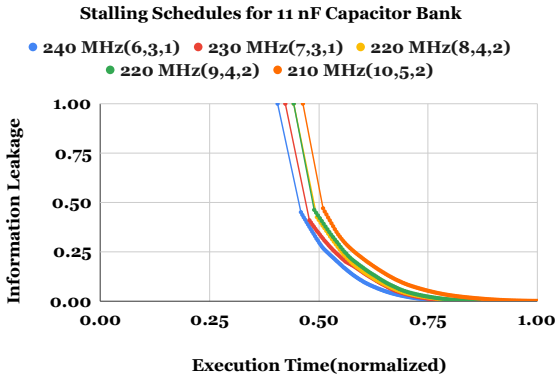
Figure 3.8. The scatter plots above shows "blink/no stall" when using a 11 nF and 22 nF capacitor bank. Non-optimal points at 11 nF are shown in blue. Optimal points at 11 nF are shown in red. Non-optimal points at 22 nF are shown in green. Optimal points at 22 nF are shown in orange. The points are extended into stalling schedules in Figure 3.9

achieves a JMI score of .470 with an NET of .510. In Figure 3.9a we see that it has the highest performance for "blink/no stall (x,y,z)" points on the graph and"blink/stall (10,5,2)" degrades performance even further.

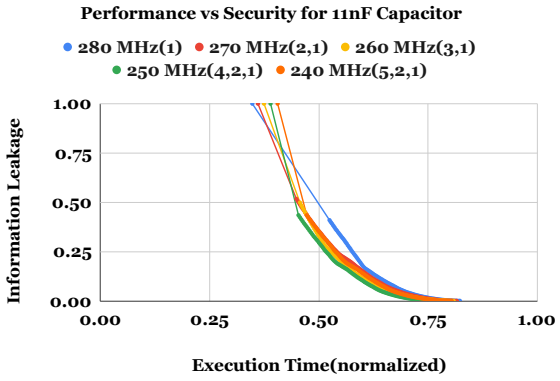
"blink/no stall (9,4,2)" and "blink/no stall (8,4,2) have very similar JMI scores and NETs. "blink/no stall (9,4,2)" achieves a JMI score of .462 with an NET of .490 and"blink/no stall (8,4,2)" achieves a JMI score of .425 with an NET of .496. Although"blink/stall (9,4,2)" and "blink/stall (8,4,2)" are both much better than"blink/stall (10,5,2)", the performance penalties for both far outweigh the security benefits.

In Figure 3.9a we also see how stalling can help designers decide between a blinking schedule which optimizes performance and a blinking schedule which optimizes security. In 3.8a we see "blink/no stall (6,3,1)" optimizes performance with an NET of .459 and achieves a JMI score of .450. Alternatively, "blink/no stall (7,3,1)" optimizes security by achieving a JMI score of .409 and an NET of .479.

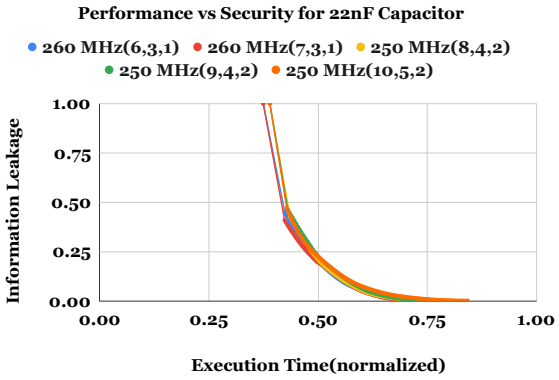
However, when we consider the "blink/stall (x,y,z)" sections of each schedule we see that "blink/stall (6,3,1)" provides better performance and security. If we want to prioritize



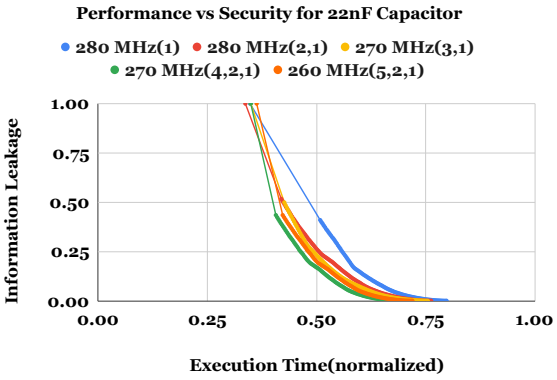
(a) Stalling schedules for long blink times with 11 nF capacitor bank



(b) Stalling schedules for short blink times with 11 nF capacitor bank



(c) Stalling schedules for long blink times with 22 nF capacitor bank



(d) Stalling schedules for short blink times with 22 nF capacitor bank

Figure 3.9. Stalling Schedules

performance, “blink/stall (6,3,1)” has an NET of .472 and achieves a JMI score of .400. If we want to prioritize security, “blink/stall (6,3,1)”achieves a JMI score of 0 with an NET of .810.

Figure 3.8a also provides an extreme example of how limiting the blink time by increasing clock speed can dramatically degrade performance. “blink/no stall (1)” and “blink/stall (1)” can only use blinks that are one cycle long. “blink/no stall (1)” only achieves a JMI score of .410 with an NET of .525. It has the highest performance for “blink/no stall (x,y,z)” points on the graph and“blink/stall (1)” degrades performance even further.

“blink/no stall (5,2,1)”, “blink/no stall (3,1)” and “blink/no stall (3,1)” have very similar JMI scores and executions times. “blink/no stall (5, 2,1)” achieves a JMI score of .435 with an NET of .472, “blink/no stall (3,1)” achieves a JMI score of .499 in with an NET of .457 and “blink/no stall (2,1)” achieves a JMI score of .517 with an NET of .449. Although these stalling schedules are all much better than“blink/stall (1)”, their performance penalties far outweigh the security benefits.

Figure 3.9b repeats the analysis in Figure 3.9a for shorter blink times. “blink/no stall (4,2,1)” optimizes performance by achieving a JMI score of .435 with an NET of .453. “blink/no stall (1)”optimizes security by achieving a JMI score of .410 and with an NET of .525. While “blink/no stall (1)” is worse than “blink/no stall (7,3,1)” with respect to both performance and security, “blink/no stall (4,2,1)” is better than “blink/no stall (6,3,1)’ with respect to both performance and security. Furthermore, when we consider the “blink/stall (x,y,z)” sections of each schedule “blink/no stall (4,2,1)” provides the best performance and security when using a 11 nF capacitor bank. If we want to prioritize performance, “blink/stall (4,2,1)” chieves a JMI score of .394 with an NET of .466. If we want to prioritize security, “blink/stall (4,2,1)” achieves a JMI score of 0 with an NET of .772. These results are summarized in Table 3.1.

Blink Times with 22 nF Capacitor Bank

In Figures 3.9c and 3.9d and we also see how additional capacitance can make it more difficult to determine the best blinking schedule. In Figure 3.8b we see that “blink/no stall (7,3,1)”

Table 3.1. 11 nF Results

Model	Area (mm^2)	ET(normalized)	JMI
Baseline [145]	670	.348	0
no blink/no stall	2.34	.348	1
blink/no stall (2,1) [3]	2.34	*.449*	.517
blink/no stall (7,3,1) [3]	2.34	.479	*.409*
blink/stall (4,2,1)	2.34	**.466**	**.394**
blink/stall (4,2,1)	2.34	.772	0

Table 3.2. 22 nF Results

Model	Area(mm^2)	ET(normalized)	JMI
Baseline [145]	670	.348	0
no blink/no stall	4.68	.348	1
blink/no stall (4,2,1)[3]	4.68	*.420*	.435
blink/no stall (7,3,1)[3]	4.68	.423	*.409*
blink/stall (4,2,1)	4.68	.432	.394
blink/stall (7,3,1)	4.68	**.432**	**.376**
blink/stall (4,2,1)	4.68	.715	0

optimizes performance and security by achieving a JMI score of .409 with an NET of .423. We also see that "blink/no stall (4,2,1)"optimizes performance and "blink/no stall (1)" optimizes security. "blink/no stall (4,2,1)" optimizes performance by achieving a JMI score of .435 with an NET of .420.

When considering the "blink/stall (x,y,z)" sections of each schedule we argue that "blink/stall (7,3,1)" optimizes performance and "blink/stall (4,2,1)" optimizes security. We argue that "blink/stall (7,3,1)" optimizes performance because when comparing the stalling schedules at a lower NET (.432), "blink/stall (7,3,1)" achieves a JMI score of .376 while "blink/stall (4,2,1)" only achieves a JMI score of .394 We argue that "blink/stall (4,2,1)"optimizes security because when comparing the stalling schedules when JMI is 0, "blink/stall (4,2,1)" has an NET of .715 while "blink/stall (7,3,1)" has an NET of .726. These results are summarized in Table 3.2.

3.6 Conclusion

Although the blink scheduling algorithm finds the optimal blink schedule under given parameters, many leakage peaks remain due to recharge constraints. By strategically stalling the processor, we are able to eliminate these leakage peaks without sacrificing performance. Furthermore, we are able to improve security even more with additional capacitance because we are able to isolate more of the power consumption with each additional blink. While the proposed methodology is tailored to power attack mitigation we plan use blinking to address other side channel attacks in future work.

3.7 Acknowledgements

Chapter 3 is coauthored with Alric Althoff, Scott Davidson, Dustin Richmond, Michael Taylor and Ryan Kastner and is currently being prepared for submission for publication of the material. The dissertation author was the primary author of this chapter.

Chapter 4

Fault Analysis Mitigation

Security is becoming a first-class optimization objective along with power, performance area, for modern EDA and chip design. Unfortunately, side channel attacks exploit weaknesses in the physical system performing computations rather than weaknesses in software algorithms to reveal secret information [123]. Passive side channel attacks monitor output from the physical system such as the timing, power consumption, or electromagnetic emanation of the system to acquire secret key bits [5]. Active side channel attacks perturb the physical system and observe the results. Fault attacks are considered active side channel attacks because an adversary must modify the physical state in some way to perform analysis that allows him to derive the secret key. Fault attacks pose a unique threat because they require less data collection than other side channel attacks and are more costly to prevent [76, 88].

Faults are purposefully injected by perturbing the clock frequency (clock glitching), perturbing the power supply (voltage glitching), electromagnetic interference (EMI), focused light beams, lasers, focused ion beams, and overheating [6]. While all of these methods can be used to inject faults, clock glitching and voltage glitching are used most widely because they are low cost and relatively easy to implement.

Boneh, DeMillo, and Lipton proposed the first theoretical fault attack against RSA using the Chinese Remainder Theorem [17]. Since then, a number of attacks have been proposed against a number of different cryptographic algorithms including Elliptic Curve Cryptography

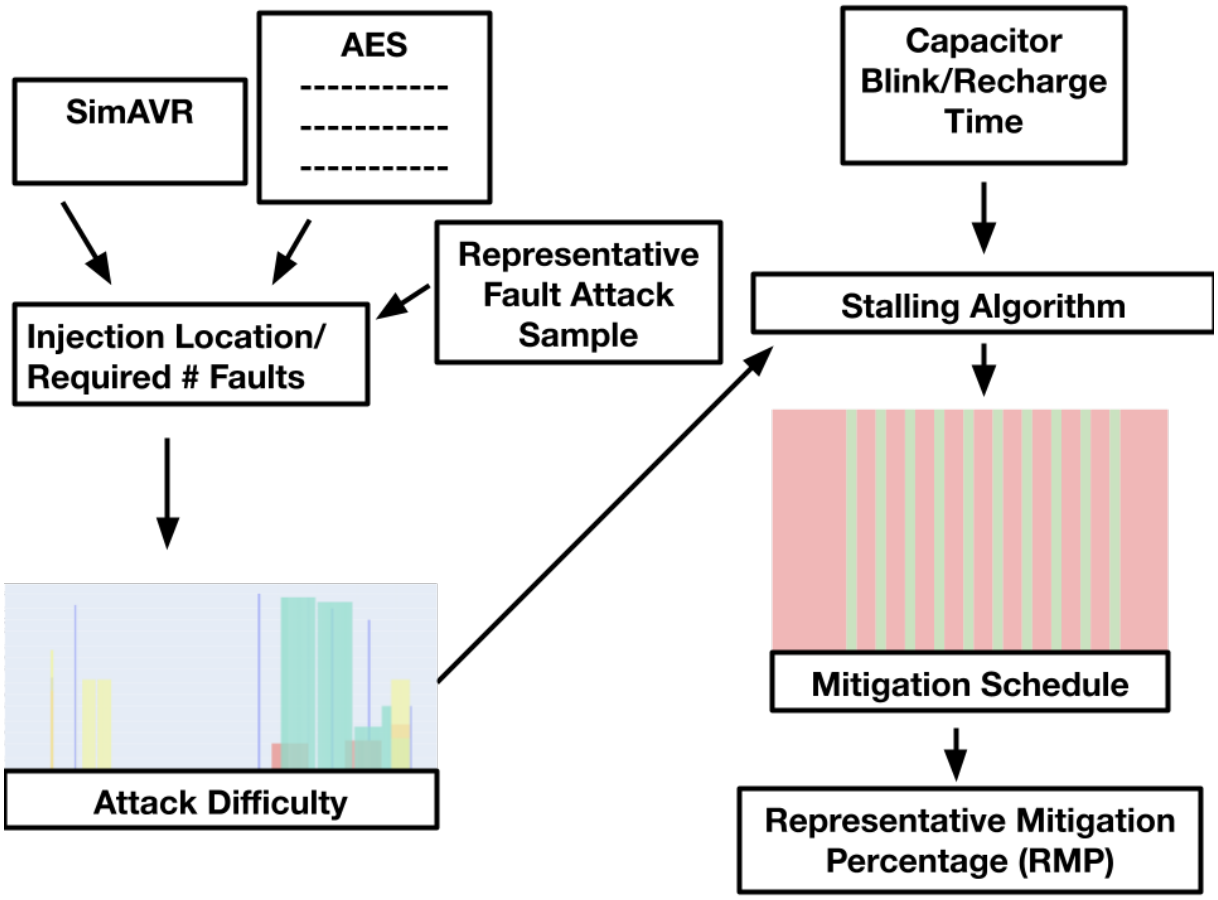


Figure 4.1. Fault analysis mitigation can be performed using an on-chip switched capacitor and ring oscillator to mitigate fault attacks in two stages. In the security phase, we use the injection location and required number of faults for a representative sample of fault attacks as our evaluation metric and rank the difficulty for launching a fault analysis attack cycle by cycle. In the design exploration phase, we take the blink and recharge times of the capacitor as well as a performance threshold as inputs to the stalling algorithm to generate a mitigation schedule which determines when the mitigation strategies should be turned on(shown in red) and when they should be turned off(shown in green). Afterwards, we measure the the effectiveness of the mitigation schedule with representative mitigation percentage(RMP) which determinines how many attacks from our representative list of fault attacks the mitigation schedule is able to protect against

[10, 48, 28], DES [72, 129, 106] and AES[13, 85, 56, 14, 40, 60, 146, 107, 18, 67, 49, 57]. AES has been the subject of extensive study for fault attacks due to its prevalence in devices such as smart cards, servers, FPGAs and TV set-top boxes.

Masking [13], time redundancy [88], and a compiler approach to mitigate fault attacks

(CAMFAS) [27] have been proposed as techniques to mitigate fault attacks. While masking adds randomization to weaken the correlation between the secret key and intermediate values, the other two techniques are simply different ways to detect faults. Time redundancy repeats steps in the algorithm to detect faults and CAMFAS duplicates instructions in larger registers to check for faults.

Blinking [3] has been used to prevent information leakage through power channels by disconnecting secure cores from the main power supply and using an on-chip capacitor for short periods of time . We propose extending this technique by using a chip an on-chip ring oscillator in addition to using an on-chip capacitor. An on-chip ring oscillator would isolate the clock signal to prevent clock glitching and an on-chip capacitor would isolate the power supply to prevent voltage glitching. In this paper, we investigate whether isolation which implements blinking would be effective as a fault attack mitigation strategy.

4.1 Fault Attacks

We choose fault attacks from [13, 85, 56, 14, 40, 60, 146, 107, 18, 67, 49, 57] to develop a representative list of fault attacks with which we will evaluate the security of our fault attack mitigation strategies. Although this is not an exhaustive list of all fault attacks against AES, it does contain a representative list of attacks detailing the main four categories of fault attacks (Fault Sensitivity Analysis (FSA), Differential Fault Analysis (DFA), Biased Fault Analysis (BFA), and Combined Fault Analysis (CFA)). FSA recovers the secret key by analyzing how an adversary manipulates aspects of the physical system to inject faults. DFA precisely derives the secret key using differences generated by faults. BFA performs statistical analysis to distinguish the correct key hypothesis from incorrect key hypotheses. CFA uses fault injection in tandem with passive side channel analysis to reveal the secret key.

Each mitigation technique is able to protect against particular categories of fault attacks. Masking is only able to mitigate FSA attacks, time redundancy is only able to mitigate DFA, and

Table 4.1. The attacks, its type, injection location, and the number of faults to perform the attack. Attacks that require more faults are considered more difficult than those requiring fewer faults.

Attack	Type	Injection Location	# Faults
[14]	DFA	AK0	128
[13]	FSA	AK0	285
[32]	CFA	AK0	1568
[13]	FSA	SB1	32
[130]	CFA	SB1-AK1	256
[56]	FSA	SB6	5×10^4
[40]	DFA	MC6-MC7	5
[56]	BFA	AK6-AK7	4×10^4
[56]	FSA	SB7	3.5×10^4
[146]	DFA	MC7-MC8	1
[67]	BFA	AK7-SB8	2
[56]	BFA	AK7-AK8	3×10^4
[56]	FSA	SB8	2×10^4
[107]	DFA	MC8-MC9	6
[49]	BFA	AK8-AK9	14
[56]	FSA	SB9	1×10^4
[60]	BFA	MC9-SB10	50
[130]	CFA	AK9-SB10	256
[18]	DFA	AK9-SB10	16
[57]	BFA	AK9-SB10	7
[85]	FSA	SB10	50

CAMFAS is only able to mitigate DFA and BFA. However, isolation is able to mitigate all four categories.

In our representative sample of fault attacks, we choose the attack from each injection location that requires the lowest number of faults for each fault attack type because it is the easiest attack of that type to execute. If there are 2 DFA attacks that require faults injected in the same location, then we only consider the attack with the least number of required faults because a single countermeasure could mitigate both of them. Alternatively, if one DFA attack, one FSA attack, one BFA attack, and one CFA attack all require faults injected the same location, then we consider all four attacks because there are different countermeasures to mitigate the different types of attacks.

4.2 Isolation

Blinking provides power isolation to mitigate fault injection through voltage glitching and an on-chip clock generator provides clock isolation to mitigate fault injection through clock glitching. We discuss how an adversary would typically inject faults using an injection method and how isolation would mitigate this process.

4.2.1 Power Isolation

Blinking uses an on-chip energy source to perform sensitive computations rather than an off-chip power supply [3]. It was initially introduced as a countermeasure for power side channel attacks because an adversary cannot learn anything about the value of the computations from the ground, power or any other pins of the chip. We argue that using an on-chip power source would also prevent an adversary from modifying the chip's power supply to inject faults.

Voltage Glitching

Adversaries can perform voltage glitching through power spikes and temporary brown outs [8]. For temporary brown outs an adversary taps into the power supply line of the device

and connects their own power supply unit with reduced feeding voltage for a single clock cycle to increase the setup time for latches in the circuit and slow down the propagation of signals on the bus lines [7]. Alternatively, for spike attacks, an adversary connects the power supply to ground for a single clock cycle to cause a dramatic voltage drop and a drastic increase in power consumption [134]. We argue that in a blinking scenario an adversary would not be able to perform either of these techniques because the chip would be disconnected from the power supply line and ground pins of the chip.

Blinking Power/Area Overhead

Designers can increase on-chip capacitance by introducing additional on-chip decoupling capacitors [122]. However, introducing more capacitors requires more area. For our experiments we were able to achieve 22 nF of storage capacitance by filling $4.68mm^2$ of the $25mm^2$ die area with decoupling capacitors.

Blinking is limited by the fact that having an on-chip source requires recharging between periods of isolation. This prevents designers from isolating portions of an algorithm's execution they would like to protect. Designers can overcome this limitation by stalling the processor while the capacitor recharges, but this incurs an additional performance cost.

4.2.2 Clock Isolation

Adversaries use clock glitching to inject faults by using their own faulty clock signal for a single clock cycle. This forces the next rising clock edge to occur earlier or causing the falling clock edge to occur later [8]. However, it been shown by [4] that it is not possible to perform clock glitching on chips that generate their own clock signal because the adversary is not able to disconnect the clock line from the circuit. Therefore, we propose extending blinking to use an on-chip clock rather than an off-chip clock signal.

A tunable ring oscillator may have advantages as an on-chip clock source rather than a phase-locked loop (PLL) and have been used practically [82, 81, 125, 51]. Although PLLs are

used as on-chip clock sources traditionally, they are sensitive devices (analog PLLs typically outperform digital PLLs significantly) and are tricky to design so most chip-builders who want a PLL would license the device for $10-100K. Off the shelf PLL's generally assume ultra-stable input voltages, which may not be a valid assumption in the presence of faults or when blinking. Furthermore, the PLL requires an external reference clock which could be glitched and thus has the potential to leak information.

Additionally, ring oscillators are much simpler than PLLs. The ring oscillator would use energy from the on-chip capacitor to generate a clock signal and as the voltage drops their behavior is much easier to predict (simply slows down) [89]. Environmental factors such as temperature will affect the clock generators, however the effects will correlate with the main circuitry. The oscillator frequencies are also tunable and designed to have a large range beyond the target frequency.

4.3 Results

4.3.1 Blinking Performance Lower Bound

We determine the lower bound for performance with blinking based on the number of cycles we can isolate per blink (maximum blink time) and how many cycles we must recharge between each blink (recharge time). The calculations are dependent on the clock speed we run the algorithm, so we perform design exploration to determine the optimal clock speed to run the algorithm.

Design Exploration

Using our maximum blink time and recharge time calculations we determine the performance for blink schedules at 270 MHz, 260 MHz, 250 MHz, 240 MHz, 230 MHz and 220 MHz. At 270 MHz we are able to isolate the power for up to four cycles, at 260 MHz we can isolate for up to seven cycles, at 250 MHz we can isolate for up to eight cycles, at 240 MHz we can isolate for up to eleven cycles, at 230 MHz we can isolate for up to twelve cycles and at 220

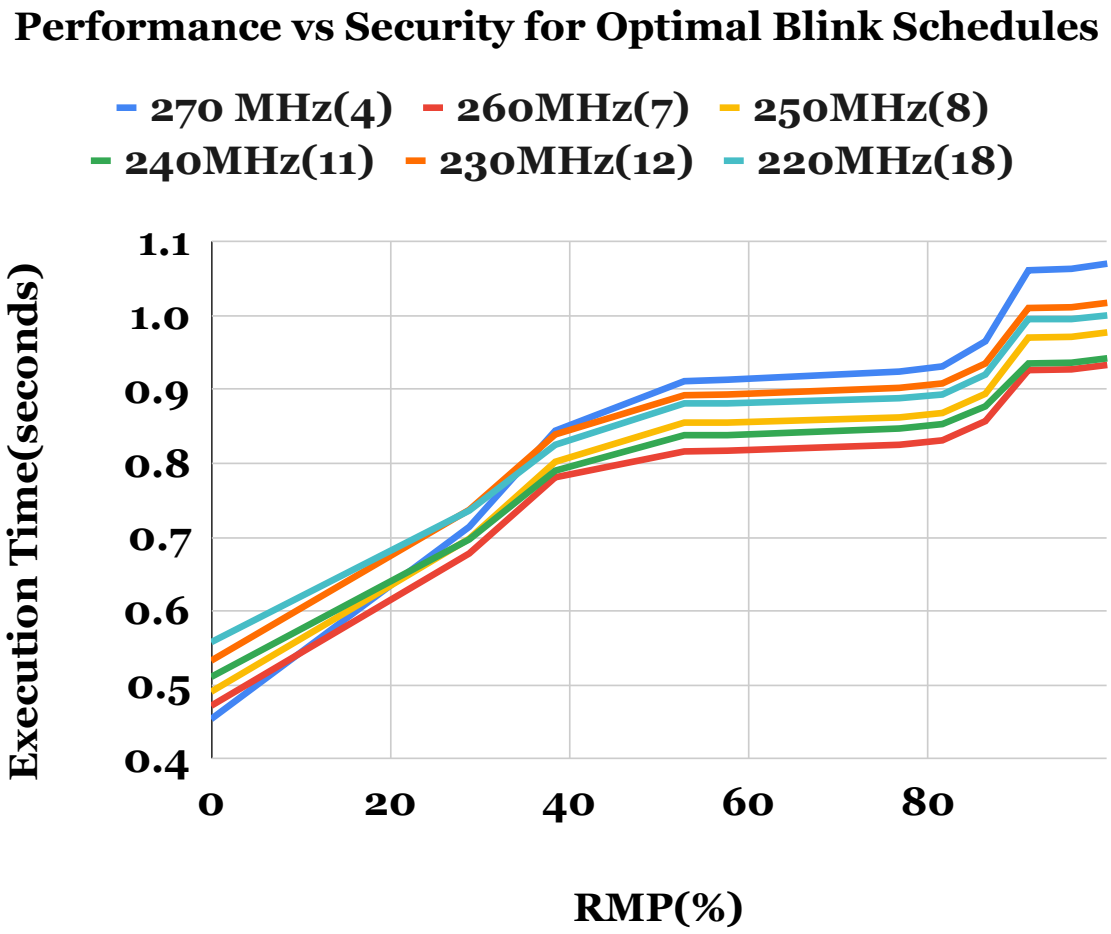


Figure 4.2. The lower bound for performance with blinking is between 220 MHz and 270 MHz

MHz we can isolate for up to eighteen cycles. However, we do not have to perform only blinks using the our maximum blink time because the injection locations are not all multiples of our maximum blink time. As an example, isolating the initial AK step requires 80 cycles. In this scenario we could perform our first blink by isolating the power for 18 cycles. After this, we would have to stall the processor for 52 cycles. Since 62 cycles still remain, we could isolate the power for another 18 cycles and we would have to stall the processor for another 52 cycles. This process would be repeated 2 more times until we reach the final 8 cycles. We could then isolate the power for 8 cycles and we would not have to stall the processor because it is performing

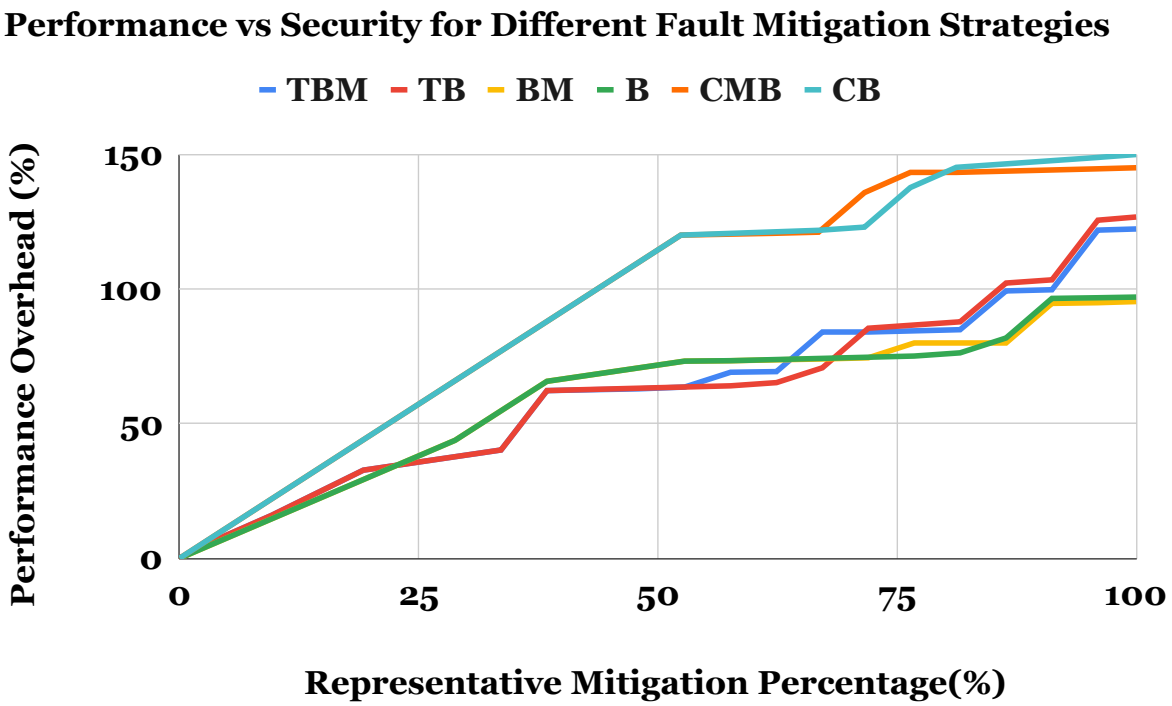


Figure 4.3. Results for Combined Fault Mitigation Strategies

instructions we do not need to isolate.

Figure 4.2 shows our design exploration to find the optimal blink schedule. The blink schedule at 230 MHz starts off slightly faster than the blink schedule at 220 MHz, but ends up being slower overall because the slight increase in clock speed does not make up for the fact that longer maximum blink times are possible at 220 MHz. However, the maximum blink times are long enough to make it better than the blink schedule at 270 MHz. The blink schedule at 240 MHz has a similar clock speed and maximum blink time, but the shorter recharge time and slightly faster clock speed make its execution time much shorter. The blink schedule at 250 MHz is similar to the blink schedule at 230 MHz. It has a similar clock speed and maximum blink time to the blink schedule at 260 MHz, but its longer recharge time and slightly slower clock speed make its execution time much longer. The blink schedule at 260 MHz defines our performance lower bound with the lowest execution time, .933 seconds.

Table 4.2. We show the performance and security for individual fault mitigation techniques. We measure performance by a technique's performance overhead and security by a technique's representative mitigation percentage (RMP) to demonstrate how many attacks a technique can mitigate from a representative sample of fault attacks.

Technique	Performance Overhead	RMP
Masking	2.8% [99]	33.3%
Time Redundancy	29.7% [88]	23.8%
CAMFAS	120% [27]	52.4%
Blinking (Isolation)	97%	100%

4.3.2 Mitigation Technique Comparison

We compare the different fault attack mitigation techniques (masking, time redundancy, CAMFAS and isolation) in terms of performance and demonstrate how a designer can make tradeoffs by selectively applying them to protect against fault attacks. We measure performance with performance overhead and security in terms of representative mitigation percentage (RMP). We calculate RMP by determining how many of the represented attacks are mitigated as shown in equation 4.1. We use RMP as a measure of security because it demonstrates how secure an implementation is after applying a fault mitigation technique.

$$RMP = \frac{attacks\,mitigated}{representative\,sample} \tag{4.1}$$

Individual Techniques

Table 4.2 shows the performance overhead for using each technique to mitigate as many fault attacks as possible.

We see that masking has very low performance overhead because it only needs to provide mitigation during the SB steps. However, only seven out of the 21 attacks in our representative list are FSA attacks so RMP is only 33.3%. Time redundancy has the next lowest performance overhead, but only five attacks from our representative list are DFA attacks so RMP is only 23.8%. CAMFAS has the highest performance overhead because it cannot be applied selectively;

Table 4.3. The attacks, its type, injection location, and the number of faults to perform the attack. Attacks that require more faults are considered more difficult than those requiring fewer faults. We mitigate attacks from least difficult to most difficult

Attack	Type	Range	Faults
[146]	DFA	MC7-MC8	1
[67]	BFA	AK7-SB8	2
[40]	DFA	MC6-MC7	5
[107]	DFA	MC8-MC9	6
[57]	BFA	AK9-SB10	7
[49]	BFA	AK8-AK9	14
[18]	DFA	AK9-SB10	16
[13]	FSA	SB1	32
[85]	FSA	SB10	50
[60]	BFA	MC9-SB10	50
[14]	DFA	AK0	128
[130]	CFA	SB1-SB2	256
[130]	CFA	AK9-SB10	256
[13]	FSA	AK0	285
[32]	CFA	AK0	1568
[56]	FSA	SB9	1×10^4
[56]	FSA	SB8	2×10^4
[56]	BFA	AK7-AK8	3×10^4
[56]	FSA	SB7	3.5×10^4
[56]	BFA	AK6-AK7	4×10^4
[56]	FSA	SB6	5×10^4

it is a compiler option that can only be turned on or off. CAMFAS can mitigate six more attacks than time redundancy because it protects against DFA and BFA attacks, but it cannot mitigate FSA or CFA attacks. Consequently, its RMP is only 52.4%. Finally, isolation has a performance overhead of 97% and an RMP of 100% because it involves introducing thousands of noops to provide power and clock isolation, which allows it to mitigate all four categories of fault attacks.

Combined Techniques

Since only isolation is able to mitigate all of the attacks, we proposed combined mitigation strategies: Time redundancy, Blinking, and Masking(TBM), Time redundancy and Blinking (TB), Blinking and Masking (BM), Blinking (B), CAMFAS, Masking and Blinking (CMB) and

CAMFAS and Blinking (CB). Figure 4.3 shows our results after using combined mitigation strategies to selectively protect against fault attacks according to their attack difficulty as shown in Table 4.3.

The CB and CMB strategies have the highest overall performance overhead with 150% and 145% performance overhead respectively. This is because using just CAMFAS incurs 120% performance overhead. B and BM have high performance overhead initially, but have lowest overall performance overhead with 97% and 95% respectively. Conversely, TB and TBM have the lowest initial performance overhead and the third and fourth highest overall performance overhead with 127% and 122% performance overhead respectively. These trends occur because of the overlap between injection locations for different types of fault attacks. TB and TBM use time redundancy to mitigate DFA attacks in the beginning and then overlap these regions later using blinking to mitigate BFA and CFA attacks. On the other hand, B and BM use blinking to mitigate all three types of attacks simultaneously. CMB, BM and TBM have slightly lower overhead than CB, B and TB because they are able to use masking rather than blinking to mitigate FSA attacks.

4.4 Conclusions

Adversaries can use clock and voltage glitching to easily inject faults for fault attacks, but isolation implementing blinking has the potential to isolate a chip's clock signal and power supply so adversaries are not able to manipulate them. Additionally, metal shielding can be added to prevent fault injection through electromagnetic interference. While blinking could incur high performance overhead, it could be performed selectively by taking advantage of the fact that faults must be injected during specific steps of the algorithm and attacks could be mitigated individually according to their attack difficulty. Furthermore, we show that isolation which implements blinking mitigates biased fault attacks for the lowest performance overhead and could be the only known strategy to mitigate combined fault attacks.

4.5 Acknowledgements

Chapter 4 is coauthored with Alric Althoff, Scott Davidson, Dustin Richmond, Michael Taylor and Ryan Kastner and is currently being prepared for submission for publication of the material. The dissertation author was the primary author of this chapter.

Chapter 5

EM Analysis Mitigation

The development of devices containing greater amounts of sensitive information has sparked the creation of many strong, mathematically secure cryptographic algorithms. Unfortunately, side channel analysis (SCA) attacks bypass these algorithms by monitoring the effects of the algorithm on a physical platform through power consumption, electromagnetic emanations (EM), timing of operations, or acoustic vibrations. By measuring these aspects of the physical computations, an attacker is able to discover sensitive information, e.g., extracting the secret key from a cryptographic algorithm.

Signature aTtenuation Embedded CRYPTO with Low-Level metAl Routing (STELLAR) mitigates power and EM SCA by routing the cryptographic core through the lower metal layers, and then embedding the crypto core locally within a signature attenuation hardware, which suppresses the critical correlated crypto signature significantly before passing it through to the higher metal layers to connect to the external pins [38]. STELLAR, proposed in 2019 [38], was shown to be secure against EM and power SCA even after 1M encryptions for an AES-128. However, STELLAR incurs a 50% power overhead to actively suppress the crypto power signature (when compared to the unprotected crypto core). To address this issue, we propose `iSTELLAR` – a programmable technique to implement STELLAR intermittently based on the side-channel leakage, thereby providing security while reducing the power overhead.

The goal of `iSTELLAR` is to minimize the power consumption while maximizing the

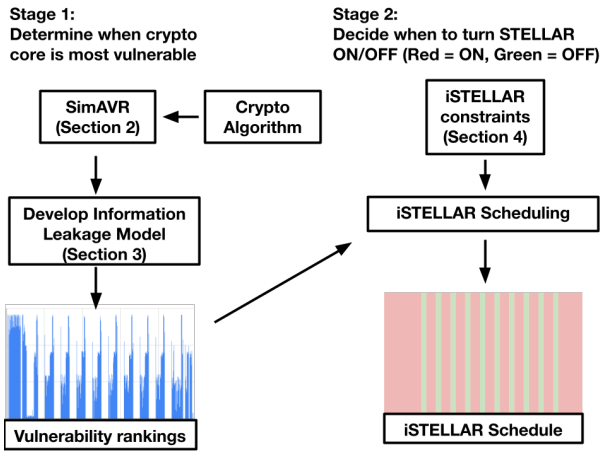


Figure 5.1. STELLAR provides high resilience to side channel attacks but requires 50% power overhead. `iSTELLAR` performs STELLAR intermittently to reduce overhead in two stages. In the first stage we generate joint mutual information (JMI) rankings to determine which parts of the algorithm's execution reveal information about the key. In the second stage, we generate an iSTELLAR schedule to determine when STELLAR should be turned ON (shown in red) or OFF (shown in green).

security of the crypto implementations against SCA attacks. Unfortunately, `iSTELLAR` cannot instantly turn ON and OFF, as it requires a turn ON delay of a few cycles. In this paper, we study those start-up constraints and propose scheduling algorithms to only turn ON `iSTELLAR` at critical time points to minimize both power overhead and information leakage (i.e., maximize SCA security).

In order to maximize the SCA security, we determine which periods of time to turn STELLAR ON based on information leakage as shown in figure 5.1. We develop an information leakage model by selecting an algorithm and performing an implementation of the selected algorithm in SimAVR[121]. Using our information leakage model, we rank the information leakage of the implementation cycle by cycle based on its usefulness to an attacker in a side channel attack. Utilizing these information leakage rankings and `iSTELLAR`'s constraints, the `iSTELLAR` scheduling algorithm decides the cycles in which to turn STELLAR ON or turn it OFF.

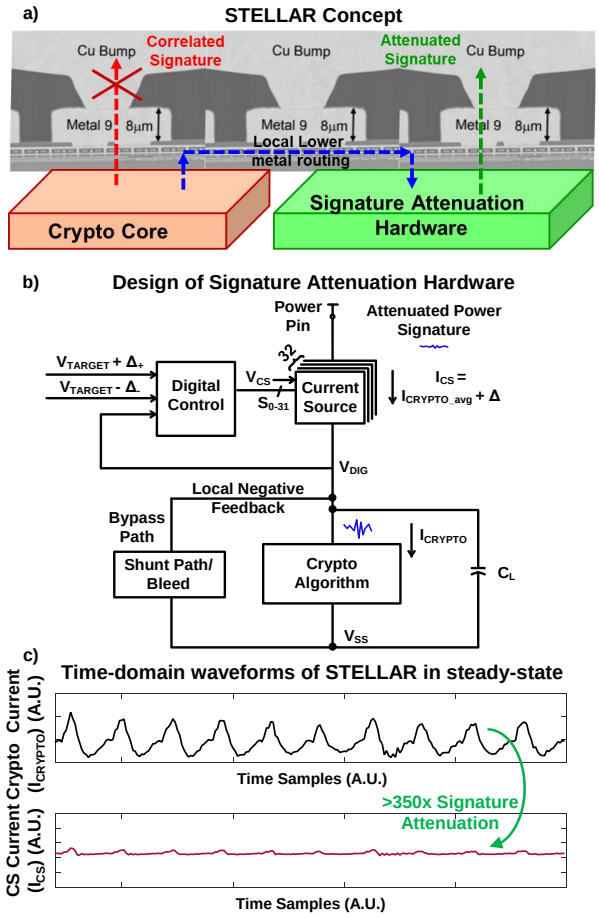


Figure 5.2. (a) STELLAR technique; and (b) design of the Signature Attenuation Hardware (SAH). (c) Time-domain measurements of the unprotected and protected AES with the SAH (STELLAR) shows $> 350\times$ current-domain signature attenuation.

5.1 STELLAR

The concept of STELLAR is shown in figure. 5.2(a). The STELLAR technique proposes routing the crypto core within the lower metal layers and then embedding it within a *signature attenuation hardware (SAH)* which suppresses the correlated critical crypto signature significantly before it reaches to the higher metal layers. figure 5.2(b) shows the design of the SAH, which ensures that the current from the top *current source (CS)* remains almost equal to the average crypto current independent of the crypto core switching. STELLAR prevents against both EM as well as power SCA attacks. For EM SCA protection, STELLAR is placed locally within the

lower-level metal layers embedding the crypto core, which is also routed in the lower metals. It has been shown that the higher metal layers leak more EM radiation compared to the lower metals due to its larger dimensions. Hence, STELLAR uses the SAH circuit to attenuate the critical signature within the lower metal layers before it goes through the higher metals to prevent against EM SCA. The SAH reduces the correlated signal to noise ratio (SNR) significantly and hence provides high power SCA immunity as well [38, 35].

As shown in figure 5.2(b), the SAH is designed with a CS on top which provides a high output impedance ensuring that the voltage fluctuations across the entire execution of AES are significantly suppressed before the critical information reaches the supply pin. Now, as the top CS current cannot be exactly equal to the average crypto current, it is set to the closest quantization level, and the quantization error (Δ) is bypassed through a shunt bleed path. Also, to compensate for process, voltage and temperature (PVT), a switched mode digital control loop with a guard band is used which turns ON or OFF the required number of CS slices to maintain an average crypto current from the top. Recently, in 2021 [59], a fully-synthesizable implementation of the STELLAR has also been proposed. `iSTELLAR` aims to intermittently turn ON the STELLAR countermeasure – only protecting the most sensitive computations. Hence, it is important to analyze the start-up constraints involved with the SAH design.

5.2 Motivation

Figure 5.3 provides an example of why STELLAR incurs larger than necessary overhead by protecting all portions of the computation. It shows information leakage of a software implementation of AES-128 (obtained from the DPA Contest v4.2 [30]) running on an AVR microcontroller. The leakage is computed using joint mutual information (JMI)(JMI explained in detail in section 5.3). Samples of the power trace that have larger JMI values reveal more information about the secret key. While some cycles have very high information leakage (noted by the high peaks in JMI), other cycles have low leakage or zero leakage. Given this, we can

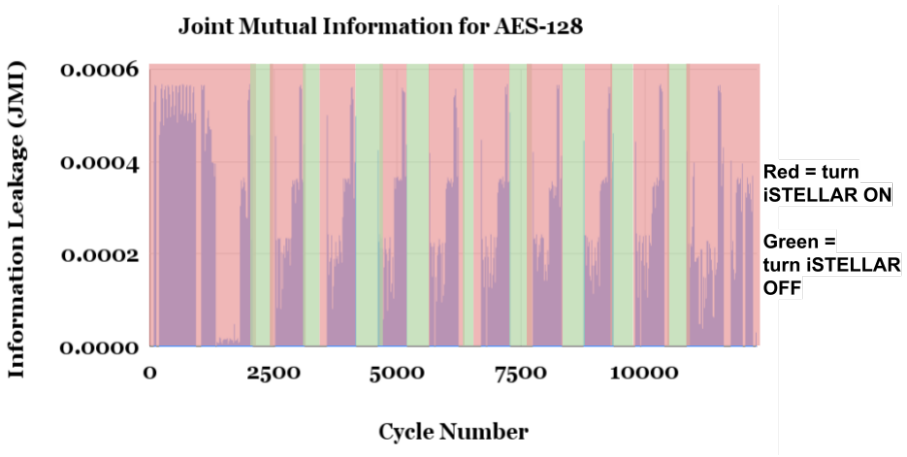


Figure 5.3. The joint mutual information (JMI) of a subset of a power trace collected from an AES-128 implementation from DPA Contest v4.2 [30]. The x-axis shows the time (cycle number) and the y-axis the JMI, which corresponds to the amount of information leakage. Cycles that have larger JMI reveal more information about the key.

turn STELLAR ON for the cycles with high leakage (shown in red), and turn STELLAR OFF for the cycles with low or no leakage to minimize the power consumption (shown in green).

5.3 Security Evaluation

`iSTELLAR` requires security evaluation in order to determine when it is most beneficial to turn STELLAR ON and how effective STELLAR will be when turned ON. We choose to use joint mutual information(JMI) as a quantitative metric to determine the most vulnerable points in the algorithm's execution and use minimum traces to disclosure (MTD) as a qualitative metric to determine how effective STELLAR will in mitigating SCA.

5.3.1 Joint Mutual Information (JMI)

Usage

We use JMI to rank time periods within the traces and these rankings are used to determine when we turn STELLAR ON/OFF [104, 150]. We calculate JMI using equation 5.1.

$$JMI_i = \sum_{j \in B} I(f(t_i, \hat{m}, \hat{s}) \frown f(t_j, \hat{m}, \hat{s}); \hat{s}) \tag{5.1}$$

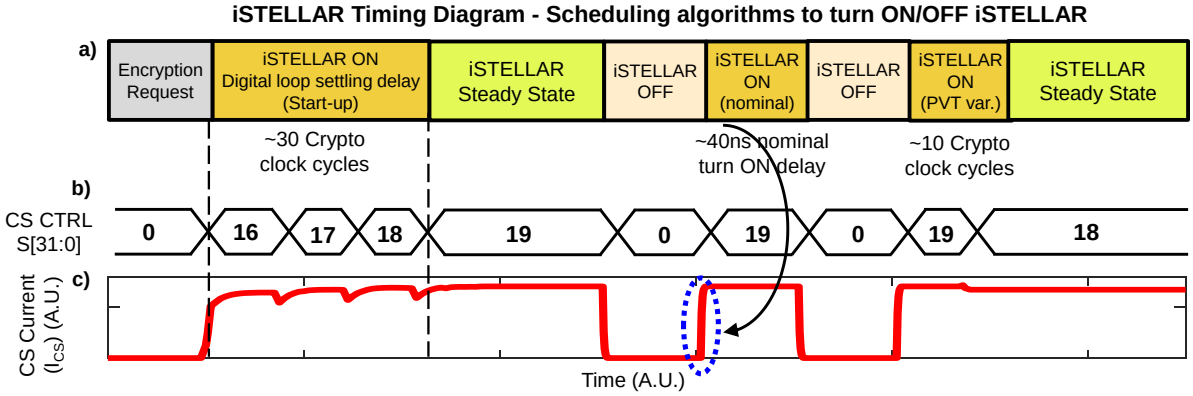


Figure 5.4. (a) Timing diagram of the iSTELLAR operation: At start-up, for the first time (only for one-time calibration), the delay is contributed by the digital control loop settling delay, which takes ~ 3 cycles to settle to the number of CS slices to ensure average crypto current from the top CS and the digital loop runs at a $10\times$ lower frequency than the crypto core. In steady state, the main delay is caused due to the settling time of the analog biases to set the top CS stage in saturation to ensure a high output impedance and maximize the signature attenuation of the current source. (b) Output of the digital control loop determines the number of CS slices to be turned ON. (c) CS current with the iSTELLAR operation, showing $\sim 40ns$ turn ON delays for all different conditions. The steady state delay constraint (circled in blue) is also illustrated in figure 5.5 which needs to be accounted for in the iSTELLAR scheduling algorithm.

$a \frown b$ calculates the concatenation between a and b, and f is a function used to represent a trace. The trace takes an independently and uniformly random message and the secret key from vectors $\hat{m}$ and $\hat{s}$. I(C; D) determines the mutual information for a and b which determines how much we can learn about C based on how it is related to D. It is calculated using equation 5.2 where H(C) is the entropy of C and $H(C|D)$ is the conditional entropy of C given D. B is the set of indices (i) that we have chosen to turn STELLAR ON.

$$I(C;D) = H(C) - H(C|D) \tag{5.2}$$

As indices are added to B, they are given a ranking based on their JMI. The index with the greatest mutual information with the key will be ranked highest and would be selected first. The algorithm repeats this process with successive time indices ranked according to how much easier they would make it for an attacker to recover the key. Once the time indices have been

ranked, the JMI scores are normalized so that the sum of all the JMI rankings is 1.

We determine JMI during design time to ensure we perform iSTELLAR in a manner that is consistent across any and all executions of the cryptographic algorithm. The iSTELLAR schedule is determined before the execution starts, and is the same regardless of the data that is being computed. Thus, this does not introduce a timing channel.

Reasoning

One reason we choose joint mutual information (JMI) as a metric is because it considers how each point in an algorithm's execution is related to all the other points[2]. This is important because it addresses the issue of variable complementarity[68]. As an example, consider a scenario where $x1 \oplus x2 = y$ under the assumption that x1 and x2 are statistically independent Boolean variables[3]. In this scenario, the mutual information between x1 and y =0 and the mutual information between x2 and y is 0. However, if we concatenate x1 and x2 ,the mutual information between the concatenation of x1, x2 and y is greater than 0 because the information from these Boolean variables completely determines y. Similarly, a SCA security evaluation metric such as the Test Vector Leakage Assessment (TVLA) may determine the time index of a security-sensitive algorithm has low vulnerability by considering just that index [65]. However, combining functions or multivariate histograms with a few time indices, it may still be attractive to an adversary if considered alongside additional time indices. In comparison, other metrics such as the Test Vector Leakage Assessment (TVLA) only consider one point in an algorithm's execution at a time. While metrics that only consider one point in an algorithm's execution at a time are very effective for determining vulnerability to a number of powerful attacks[19, 77, 43, 41], an implementation may still be vulnerable to attacks which consider multiple points in an algorithm's execution at a time[26, 61, 97].

Another reason we choose joint mutual information as a metric is because it provides a numeric score[2]. Having a numeric score allows us to precisely compare each cycle in an algorithm's execution to other cycles and precisely compare one potential iSTELLAR schedule to

any other potential iSTELLAR schedules. We assume that each cycle in an algorithm's execution that handles sensitive information has the potential to reveal it if left unprotected and this is indicated by a non-zero JMI value. However, different cycles may reveal differing amounts of information to an adversary. Time periods with high JMI values indicate high vulnerability to SCA attacks and similarly time periods with low JMI values indicate low vulnerability to SCA attacks.

It is not possible to launch an attack if JMI = 0 because if this is true, the measurements with differing secret key hypotheses are always equal[3]. Therefore, it is impossible for an attacker to differentiate between different secret key hypotheses given sets of measurements.

While we performed our analysis using JMI, our methodology can easily use other leakage assessment metrics. Any metric capable of providing a numeric score to compare one cycle in an algorithm's execution to another could be used in place of JMI to rank and determine the time points that are most vulnerable to SCA.

5.3.2 Minimum Traces to Disclosure (MTD)

Usage

Figure 5.2(c) shows that the signature in case of the protected implementation is significantly suppressed by $> 350\times$ compared to the unprotected implementation, thereby promising a $> 350^2\times$ improvement in the MTD. Recently, STELLAR has been fabricated in TSMC 65nm process using AES256 as the crypto core [36]. Embracing signature attenuation and local lower metal routing, the countermeasure achieved a MTD of $> 1B$, which to the best of our knowledge is the best reported result to date. Due to the efficiency of STELLAR, we assume that if STELLAR is turned ON, then the cycles it protects will not leak sufficient information for an adversary to launch a SCA attack and these cycles will assume a JMI value of 0.

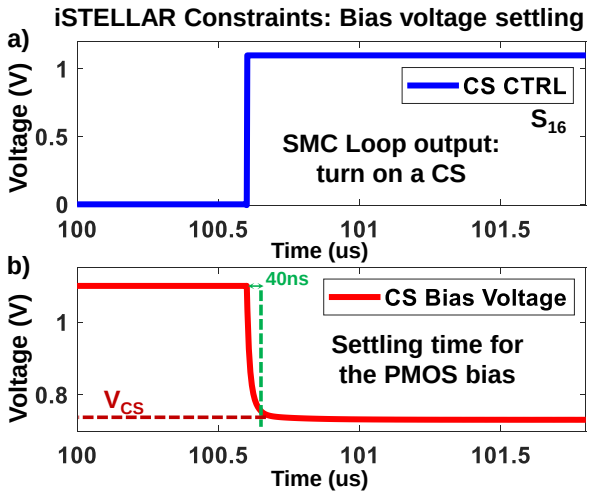


Figure 5.5. Transient analysis of the CS bias voltage settling time. (a) As the digital controller turns on a CS stage, (b) the PMOS voltage requires a settling time of $\sim 40ns$ to bias the CS in to saturation to provide a high output impedance for STELLAR.

Reasoning

We chose minimum traces to disclosure (MTD) as a metric because it models the time and effort it would take for an adversary to successfully launch an SCA attack in a practical scenario. We did not use MTD to compare each cycle in algorithm's execution to the others because it is computationally expensive when considering how each point in an algorithm's execution is related to all other points. As an example, let us first consider the scenario where we only consider one point in an algorithm's execution at a time. In this scenario we only need to calculate the MTD for (n:1) combinations where n is the number of cycles in the crypto algorithm. This has a time complexity of O(n). However, if we need to consider how each point is related to all other points we must calculate the MTD for (n:1), (n:2), (n:3)...(n:m) where m is the number of time points an adversary uses for their attack and $m < n$. This increases the time complexity to O(n:m).

5.4 iSTELLAR

STELLAR can remain ON for any amount of time. However, if we turn STELLAR OFF to conserve power during cycles with low or no leakage, we must wait a certain amount of time for STELLAR to complete its start-up process. As a result, we must determine when to turn STELLAR ON/OFF while allowing it the necessary time to start back up after being turned OFF.

5.4.1 Constraints

The timing diagram for the iSTELLAR operation is shown in figure 5.4. At start-up, the signature attenuation hardware (SAH) requires a few cycles for the digital switched mode control (SMC) loop to set the CS current at the average crypto current. The SMC, which runs at $10\times$ lower frequency compared to the crypto core, requires $\sim 2-3$ cycles to reach steady state, which is $\sim 20-30$ crypto clock cycles. In steady state, the scheduling algorithms for iSTELLAR intermittently turn the SAH ON and OFF depending on the information leakage content in the traces. Once the digital loop is stable and the STELLAR operates in steady-state, the iSTELLAR algorithm may want to turn STELLAR OFF after a certain time point. With STELLAR turned OFF, once iSTELLAR turns it ON again, the CS biases need to settle to the correct bias voltages to ensure the transistors remain in saturation for a high output impedance thereby leading to a high signature attenuation.

Figure 5.5(a) shows that one of the 32 bits of the SMC output (S_{16}) switches from zero to one, which means that the corresponding CS slice should be turned ON. Now, to turn ON a CS slice, the bias voltage at the gate of the PMOS needs to transition from the supply voltage (1.1V) to the CS bias voltage (V_{CS}). With the AES operating at 50MHz, the settling time for the PMOS bias is $\sim 40ns$. This implies that, we need to wait for $\sim 40ns$ once we turn OFF STELLAR to turn it back ON. For the purpose of our calculations in sections 5.4.3 and 5.5.2 we conservatively estimate this analog settling time to have a turn ON delay of 3 cycles in our digital loop to account for a broad range of clock speeds. This analog settling time as shown in

figure 5.5 is $\sim 40ns$, and this is the typical steady state delay the iSTELLAR needs to handle each time the STELLAR SAH circuit is turned ON and OFF. Note that this delay is due to the settling of the analog bias voltages to turn ON the PMOS CS slice, and is equal to the RC time constant for the node, where R is the on resistance and C is the gate capacitance of the PMOS. Now, if the frequency of the crypto core is increased, the size of the PMOS CS slice can be increased accordingly so that the RC time constant remains the same (R reduces, C increases due to larger size). Hence, iSTELLAR only needs to deal with the analog bias settling delay of 40ns as it turns OFF and ON the PMOS CS (figure 5.5). While this is typically the only delay, iSTELLAR may require up to 10 crypto clock cycles (equivalent to 1 cycle of the SMC loop) to account for the PVT(process/voltage/temperature) variations and settle to the optimal number of CS slices, as illustrated in figure 5.4.

Overheads & Performance of iSTELLAR

As discussed, in steady state operation of the STELLAR, the only constraint for turning it ON and OFF is the turn ON delay for the analog bias to bring the top current source stage into saturation region. Our scheduling algorithm for iSTELLAR ensures that at the end of its OFF time, it turns ON the STELLAR circuit for the bias settling. Hence, there are no performance overheads associated with the turn ON or OFF of the STELLAR hardware.

Overall, the area overhead for STELLAR is $\sim 40\%$ and the power overhead is $\sim 50\%$ [38], which is drastically reduced using the proposed iSTELLAR technique without incurring any performance penalty.

5.4.2 iSTELLAR Lower Bound

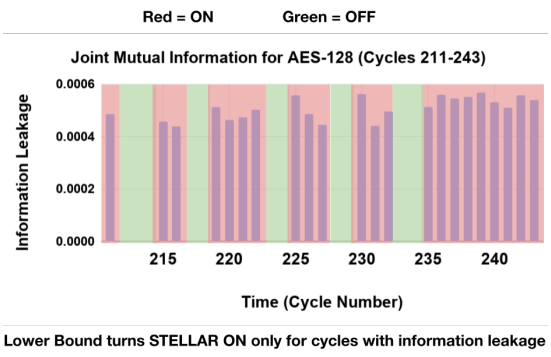
To determine the best times to turn STELLAR ON, we began by developing a best-case scenario algorithm to minimize power consumption and maximize security to establish iSTELLAR's lower bound for power consumption. In this algorithm, we assume we can turn STELLAR back ON immediately after turning it OFF.

First, we determine the total information leakage for a trace according to each cycle's JMI. Next, we find the highest leakage cycle and mark it as a cycle where we would like to turn STELLAR ON. After this, we add it to a list of cycles we plan to turn STELLAR ON for and determine the new total information leakage and power overhead for turning STELLAR ON. We continue this process for every cycle that has a JMI greater than zero or until we reach a power overhead threshold selected by the designer. For example, in figure 5.6a, we would mark cycles in the following order: 239, 230, 225, 242, 238, 237, 243, 240, 235, 219, 241,222, 232, 226, 211, 221, 220, 215, 227, 231, 216. After marking these cycles we would choose to turn STELLAR ON for the cycles we selected (shown in red) and OFF for the cycles we did not select (shown in green).

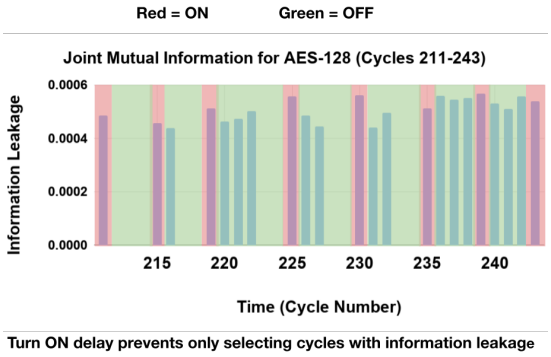
5.4.3 iSTELLAR Scheduling

Using the lower bound algorithm under realistic constraints presents an interesting challenge due to the turn ON delay required after turning STELLAR OFF. In most circumstances, this will be the 40 ns turn ON delay which we conservatively estimate to be 3 crypto clock cycles to account for a broad range of clock speeds. This means that after we turn STELLAR OFF, we must wait 3 clock cycles before turning STELLAR back ON. As a result, if any cycle we plan to select is within 3 cycles of a cycle we have already selected, we will have to ignore it because it would violate the 3 cycle turn ON delay. Similarly, if we want to account for the 10 cycle turn ON delay due to PVT variations, if any cycle we plan to select is within 10 cycles of a cycle we have already selected, we will have to ignore it because it would violate the 10 cycle turn ON delay.

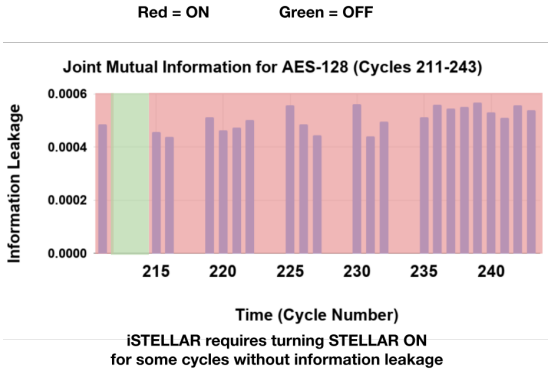
In figure 5.6b, the lower bound algorithm would mark cycles 230 and 239 because they have the highest JMI. Next, it would mark cycle 225 rather than cycle 236 because it is within 3 cycles of another cycle we have already marked. For the same reason, cycles 216-218, 220-224, 226-229, 231-234,236-238, and 240-242 would be ignored and cycles 215, 219, 235 and 243 would be marked.



(a) iSTELLAR Lower Bound



(b) Challenge



(c) iSTELLAR

Figure 5.6. (a) By turning ON STELLAR (shown in red), we can hide parts of the trace, which effectively eliminates the information leakage, that is, JMI = 0. (b) However, if we turn it OFF for cycles where there is no information leakage, we must wait for a set amount of cycles (shown in green) for it to start back up . (c) If we keep STELLAR ON for some cycles that have no information leakage, then we are able to hide nearby cycles with high information leakage.

To further decrease information leakage, we chose to check for and address cycles that occur during the turn ON delay between STELLAR being turned OFF and being turned back ON. If a new cycle we plan to mark would violate the turn ON delay of a cycle we have already marked, we will keep STELLAR ON between the cycle we plan to mark and the cycle we have already marked.

Figure 5.6c shows how using this approach addresses the challenge from our first approach. Using this approach, the lower bound algorithm would still mark cycles 230 and 239 because they have the highest JMI. Next, it would mark cycle 236 and realize that it would violate the turn ON delay if we turned STELLAR OFF then tried to turn it ON again for cycle

239. As a result, we would also mark cycles 237 and 238 even though they do not have the next highest JMI. This process continues until we have marked all of the cycles that have a JMI value greater than 0. It will also mark cycles 216-217, 223-224, 228-229 and 233-234 even though they have a JMI value of 0 to ensure STELLAR stays ON between cycles that would violate each others' turn ON delay. However, it will not mark cycles 212-214 because no surrounding cycles violate each others' turn ON delay time. Even though we use power to turn STELLAR ON for cycles where STELLAR is unnecessary, we are able to achieve higher security at a lower power overhead.

5.5 Results

Using `iSTELLAR`'s constraints, the JMI rankings for each cycle, `iSTELLAR`'s power overhead lower bound and an `iSTELLAR` scheduling algorithm, we are able to develop an iSTELLAR schedule to minimize information leakage and power overhead. We can choose to keep STELLAR OFF for as many cycles as we choose, to make tradeoffs between power and security. This section evaluates these tradeoffs.

5.5.1 Experimental Setup

We develop power traces using SimAVR to simulate an Atmel ATmega328 chip[121, 74]. SimAVR is capable of executing binaries compiled by the avr-gcc toolchain as they would be run on an AVR microcontroller and we use it to collect power traces using a Hamming distance leakage model [19]. Although we evaluate power traces, a similar experimental setup could be performed to use EM traces.

To perform our evaluation, we collect power traces for 2^{14} experimental plaintext and secret key vectors on an implementation of AES-128 from DPA Contest v4.2 [30], an implementation of PRESENT from the avr library and an implementation of AES from the avr library. Under this model, each time point in a trace consists of the difference in Hamming distance between an opcode and its predecessor for different experimental plaintext and secret key vectors

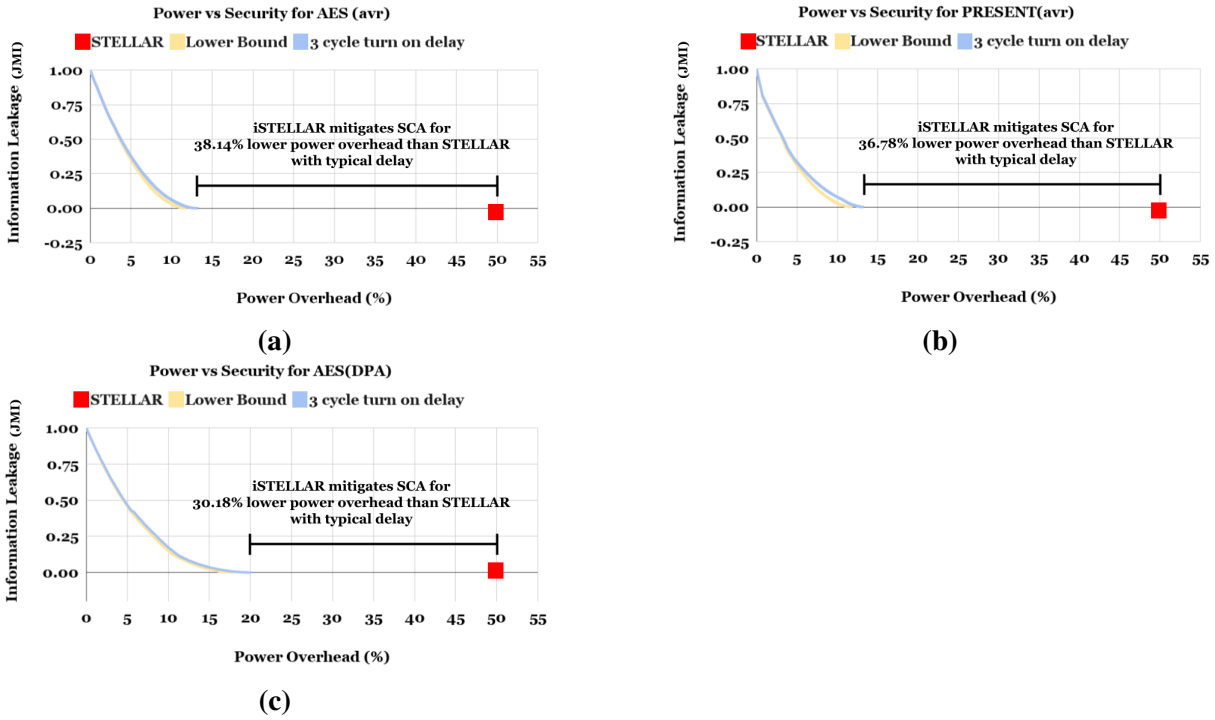


Figure 5.7. (a) Considering the 3 cycle turn ON delay, iSTELLAR achieves 38.14% less power overhead on AES(avr), (b) 36.78% less power overhead on PRESENT(avr), and (c) 30.18% less power overhead than STELLAR on AES(DPA)

[3]. For our information leakage model we assume that toggling a bit consumes one bit of power and leaving a bit unchanged consumes no power. Our information leakage evaluation is independent of the instruction type or the type of the data.

5.5.2 Power vs Security

Since STELLAR is responsible for eliminating information leakage, we consider any decision to keep STELLAR ON/OFF as a tradeoff between power and security. We can choose to maximize security by ensuring STELLAR is ON for every cycle that leaks information or we can minimize power overhead by having the designer select a specific power threshold and only turning STELLAR ON for cycles the iSTELLAR scheduling algorithm has marked up to that point. Once we establish an iSTELLAR schedule, the schedule will be constant for all executions of the algorithm regardless of the input data to avoid introducing a timing channel.

Figure 5.7a shows the power and security for using iSTELLAR on an implementation

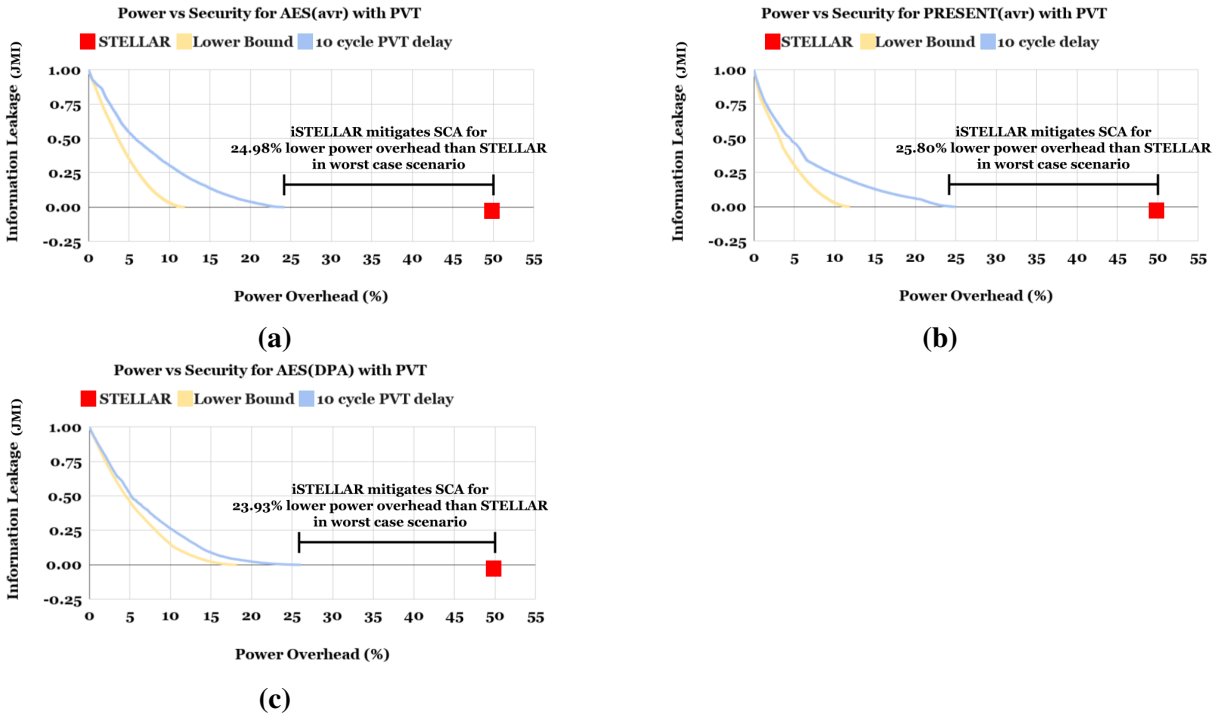


Figure 5.8. (a) Considering the process/voltage/temperature (PVT) variations, iSTELLAR achieves 24.98% less power overhead on AES (avr), (b) 25.80% less power overhead on PRESENT (avr), (c) and 23.93% less power overhead than STELLAR on AES (DPA).

of AES-128 from the avr library. The red square labeled STELLAR is representative of the baseline STELLAR technique. This assumes that no iSTELLAR scheduling algorithm is used and STELLAR remains ON for the entire algorithm's execution. Under these conditions, STELLAR incurs a 50% power overhead, but is able to eliminate power and EM leakage to reduce the sum of JMI rankings to 0. This means it is not possible to launch an SCA attack because an adversary will not be able to differentiate between different secret key hypotheses by measuring the power consumption or electromagnetic emanations.

The yellow line in figure 5.7a labeled Lower Bound assumes the scenario outlined in section 5.4.2 which ignores iSTELLAR's turn ON delay constraint to establish a best-case scenario for minimizing power consumption and maximizing security. Under these conditions, the sum of JMI rankings starts at 1 which means power and EM leakage is completely unmitigated and vulnerable to attack. If we choose to turn STELLAR on, only for cycles that have a JMI

Table 5.1. Power Overhead for iSTELLAR

Algorithm	AES(avr)	PRESENT(avr)	AES(DPA)
STELLAR	50%	50%	50%
Lower Bound	11.87%	12.67%	19.82%
3 cycle turn ON delay	13.14%	13.22%	20.05%
10 cycle PVT delay	24.20%	25.02%	26.07%

value great than 0, we are able to reduce the sum of JMI rankings to 0 for 11.87% power overhead. Additionally, if 11.87% power overhead is still too high, the designer can establish a power threshold, and the iSTELLAR scheduling algorithm is able to maximize security for that threshold.

The blue line in figure 5.7a labeled 3 cycle turn on delay assumes the scenario outlined in section 5.4.3 which requires a 3 cycle turn ON delay to turn STELLAR back ON after it has been turned OFF. Under these conditions, we are able to reduce the sum of JMI rankings to 0 for 13.14% power overhead, only 1.27% higher than the Lower Bound scenario. Furthermore, we are also able to maximize security within a power threshold for around the same power overhead as the Lower Bound scenario.

Figure 5.7b shows the power and security for using `iSTELLAR` on an implementation of PRESENT from the avr library and 5.7c shows the power and security for using an implementation of AES-128 from DPA Contest v4.2. Figure 5.8 repeats the scenarios for 5.7 with the a 10 cycle PVT turn ON delay rather than the typical 3 cycle turn ON delay to account for worst case conditions. These results are summarized in table 5.1.

5.5.3 Discussion

We believe that the efficiency of iSTELLAR may be dependent on the leakage distribution in the implementations of the different algorithms. The implementation of AES-128 from DPA Contest v4.2 may have the power overhead closest to the Lower Bound scenario because its leakage is distributed over fewer cycles. While 28.71% of the cycles in this implementation of

AES-128 have a JMI value of 0, only 14.06% of the cycles in the AES-128 implementation from the avr library have a JMI value of 0, and only 13.57% of the cycles in the PRESENT from the avr library have a JMI value of 0. We further note that the implementations of PRESENT and AES-128 from the avr library have similar leakage distributions and similar power overheads necessary to turn STELLAR ON for.

5.6 Related Work

Power side-channel countermeasures attempt to modify the power trace signal to hide any information related to the key. Some techniques add active equalization circuitry to diminish power variations during execution and keep the power supply constant [109, 110]. Other techniques use signal attenuation hardware to reduce the power cost of noise injection [37] or use a suppression circuit to reduce low frequency power variations and a low-pass filter to reduce high frequency power variations [128]. There are some ideas to use internal power sources which an adversary cannot modify e.g., a charge-pump circuit using on-chip capacitors [119] and a switched capacitor circuit to isolate an AES core from the power supply line [145]. The disadvantage to these works is that they were not applied selectively in order to allow designers to make trade-offs between performance, area, and security. While computational blinking [3] also identifies non-uniformity in information leakage, it implements a switched capacitor circuit rather than using signal attenuation hardware. As a result, applying the different techniques intermittently has different accommodations and requires different constraints. STELLAR is one of the most recent proposals as a signature attenuation-based countermeasure and has been demonstrated to achieve the highest security (MTD of 1B traces) with only $\sim$ 50% power and $\sim$ 40% area overheads [35, 59]. Moreover, it is a generic countermeasure (agnostic to any crypto algorithm) without any performance degradation.

5.7 Conclusion

Although STELLAR provides protection from power and EM SCA, it incurs larger than necessary power overhead because many of the cycles it protects do not have any information leakage. By turning STELLAR ON and OFF, we are able to to eliminate all information leakage with minimal power overhead. However, turning STELLAR OFF, we must give it a set amount of cycles to turn back ON. Utilizing our proposed scheduling algorithm for `iSTELLAR`, we address this issue by turning STELLAR ON for all cycles with high information leakage as well as some of prior cycles with low information leakage to avoid violating startup constraints. We were able to eliminate information leakage with 38.14% lower power overhead.

5.8 Acknowledgements

Chapter 5 is coauthored with Debayan Das, Shreyas Sen and Ryan Kastner and is currently being prepared for submission for publication of the material. The dissertation author was the primary author of this chapter.

Chapter 6

Conclusion

In this dissertation we have demonstrated how the blinking methodology can be used to mitigate power analysis attacks, fault analysis attacks and EM analysis attacks. A switched capacitor can be used to mitigate power analysis and fault attacks and STELLAR can be used to mitigate power analysis attacks and EM analysis attacks. Power and EM analysis attacks use JMI as a vulnerability metric and the sum of remaining JMI rankings as an evaluation metric. Fault attacks use fault attack difficulty as an evaluation metric and RMP as an evaluation metric. In all of these cases, we are able to determine the best times to use the mitigation strategy by determining when it is most beneficial to utilize the mitigation technique, how much overhead a hardware designer is willing to sacrifice for security and the constraints for when it is possible to turn a mitigation strategy on or off.

Bibliography

[1] Manfred Aigner and Elisabeth Oswald. Power analysis tutorial, 2000.

[2] Alric Althoff, Jeremy Blackstone, and Ryan Kastner. Holistic power side-channel leakage assessment: Towards a robust multidimensional metric. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019.

[3] Alric Althoff, Joseph McMahan, Luis Vega, Scott Davidson, Timothy Sherwood, Michael B. Taylor, and Ryan Kastner. Hiding intermittent information leakage with architectural support for blinking. *International Symposium on Computer Architecture*, pages 638–649, 2018.

[4] Frederic Amiel, Christophe Clavier, and Michael Tunstall. Fault analysis of dpa-resistant algorithms. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 223–236. Springer, 2006.

[5] Frederic Amiel, Karine Villegas, Benoit Feix, and Louis Marcel. Passive and active combined attacks: Combining fault attacks and side channel analysis. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pages 92–102. IEEE, 2007.

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

[7] Alessandro Barenghi, Guido Bertoni, Emanuele Parrinello, and Gerardo Pelosi. Low voltage fault attacks on the rsa cryptosystem. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 23–31. IEEE, 2009.

[8] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.

[9] Mark R Barron. Creating consumer confidence or confusion-the role of product certification marks in the market today. *Marq. Intell. Prop. L. Rev.*, 11:413, 2007.

[10] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Annual International Cryptology Conference*, pages 131–146. Springer, 2000.

[11] Alfonso Blanco Blanco, Jose María de Fuentes, Lorena González-Manzano, Luis Hernández Encinas, Agustín Martín Muñoz, José Luis Rodrigo Oliva, and J Ignacio Sánchez García. A framework for acquiring and analyzing traces from cryptographic devices. In *International Conference on Security and Privacy in Communication Systems*, pages 283–300. Springer, 2017.

[12] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably secure masking of aes. In *International workshop on selected areas in cryptography*, pages 69–83. Springer, 2004.

[13] Johannes Blömer and Volker Krummel. Fault based collision attacks on aes. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 106–120. Springer, 2006.

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

[15] Andrey Bogdanov, Ilya Kizhvatov, Kamran Manzoor, Elmar Tischhauser, and Marc Witteman. Fast and memory-efficient key recovery in side-channel attacks. In *International Conference on Selected Areas in Cryptography*, pages 310–327. Springer, 2015.

[16] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *International workshop on cryptographic hardware and embedded systems*, pages 450–466. Springer, 2007.

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

[18] Arnaud Boscher and Helena Handschuh. Masking does not protect against differential fault attacks. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 35–40. IEEE, 2008.

[19] Eric Brier, Christophe Clavier, , and Francis Olivier. Correlation power analysis with a leakage model. *International Workshop on Crypto- graphic Hardware and Embedded Systems*, pages 16–29, 2004.

[20] Robert Callan, Farnaz Behrang, Alenka Zajic, Milos Prvulovic, and Alessandro Orso. Zero-overhead profiling via em emanations. In *Proceedings of the 25th international symposium on software testing and analysis*, pages 401–412, 2016.

[21] Robert Callan, Nina Popovic, Angel Daruna, Eric Pollmann, Alenka Zajic, and Milos Prvulovic. Comparison of electromagnetic side-channel energy available to the attacker from different computer systems. In *2015 IEEE International Symposium on Electromagnetic Compatibility (EMC)*, pages 219–223. IEEE, 2015.

[22] Robert Callan, Alenka Zajic, and Milos Prvulovic. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 242–254. IEEE, 2014.

[23] Robert Callan, Alenka Zajić, and Milos Prvulovic. Fase: Finding amplitude-modulated side-channel emanations. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 592–603. IEEE, 2015.

[24] Robert Locke Callan. *Analyzing software using unintentional electromagnetic emanations from computing devices*. PhD thesis, Georgia Institute of Technology, 2016.

[25] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.

[26] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[27] Zhi Chen, Junjie Shen, Alex Nicolau, Alex Veidenbaum, Nahid Farhady Ghalaty, and Rosario Cammarota. Camfas: A compiler approach to mitigate fault attacks via enhanced simdization. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 57–64. IEEE, 2017.

[28] Mathieu Ciet and Marc Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, codes and cryptography*, 36(1):33–43, 2005.

[29] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 252–263. Springer, 2000.

[30] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M Abdelaziz Elaabid, Benoit Gerard Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomne, Daisuke Nakatsu, Kazuo Ohta, Kazuo Sakiyama, Laurent Sauvage, Werner Schindler, Marc Stottinger, Nicolas Veyrat-Charvillon, Matthieu Walle, and Antoine Wurker. Practical improvements of side- channel attacks on aes: feedback from the 2nd dpa contest. *Journal of Cryptographic Engineering*, pages 259–274, 2014.

[31] Christophe Clavier, Benoit Feix, Georges Gagnerot, and Mylene Roussellet. Passive and active combined attacks on aes combining fault attacks and side channel analysis. *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 10–19, 2010.

[32] Christophe Clavier, Benoit Feix, Georges Gagnerot, and Mylène Roussellet. Passive and active combined attacks on aes combining fault attacks and side channel analysis. In *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 10–19. IEEE, 2010.

[33] Jeremy Cooper, Elke Demulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. Test vector leakage assessment (tvla) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.

[34] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 231–237. Springer, 2000.

[35] Debayan Das, Josef Danial, Anupam Golder, Nirmoy Modak, Shovan Maity, Baibhab Chatterjee, Dong-Hyun Seo, Muya Chang, Avinash L Varna, Harish K Krishnamurthy, et al. Em and power sca-resilient aes-256 through¿ 350× current-domain signature attenuation and local lower metal routing. *IEEE Journal of Solid-State Circuits*, 56(1):136–150, 2020.

[36] Debayan Das, Josef Danial, Anupam Golder, Nirmoy Modak, Shovan Maity, Baibhab Chatterjee, Donghyun Seo, Muya Chang, Avinash Varna, Harish Krishnamurthy, et al. 27.3 em and power sca-resilient aes-256 in 65nm cmos through¿ 350× current-domain signature attenuation. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 424–426. IEEE, 2020.

[37] Debayan Das, Shovan Maity, Saad Bin Nasir, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. Asni: Attenuated signature noise injection for low-overhead power side-channel attack immunity. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(10):3300–3311, 2018.

[38] Debayan Das, Mayukh Nath, Baibhab Chatterjee, Santosh Ghosh, and Shreyas Sen. Stellar: A generic em side-channel attack protection through ground-up root-cause analysis. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 11–20. IEEE, 2019.

[39] Nicolas Debande, Youssef Souissi, Maxime Nassar, Sylvain Guilley, Thanh-Ha Le, and Jean-Luc Danger. "re-synchronization by moments": An efficient solution to align side-channel traces. In *2011 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2011.

[40] Patrick Derbez, Pierre-Alain Fouque, and Delphine Leresteux. Meet-in-the-middle and impossible differential fault analysis on aes. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 274–291. Springer, 2011.

[41] Guo-liang Ding, Zhi-xiang Li, Xiao-long Chang, and Qiang Zhao. Differential electromagnetic analysis on aes cryptographic system. In *2009 Second Pacific-Asia Conference on Web Mining and Web-based Application*, pages 120–123. IEEE, 2009.

[42] Daniel Dinu and Ilya Kizhvatov. Em analysis in the iot context: lessons learned from an attack on thread. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 73–97, 2018.

[43] Er-Peng Duan, Ying-Jian Yan, and Pei-Zhi Li. Correlation electromagnetic analysis against aes cryptographic on implementations of fpga. *Computer Engineering and Design*, 33(8):2926–2930, 2012.

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

[45] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, 24(6):522–533, 2007.

[46] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on bliss lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1857–1874, 2017.

[47] Hai-Feng Fan, Ying-Jian Yan, Jin-Fu Xu, and Fang Ren. Simulation of correlation power analysis against aes cryptographic chip [j]. *Computer Engineering and Design*, 2:260–262, 2010.

[48] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. Fault attack on elliptic curve montgomery ladder implementation. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 92–98. IEEE, 2008.

[49] Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on aes with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–118. IEEE, 2013.

[50] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *International workshop on cryptographic hardware and embedded systems*, pages 251–261. Springer, 2001.

[51] Mingze Gao, Khai Lai, and Gang Qu. A highly flexible ring oscillator puf. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[52] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *International workshop on cryptographic hardware and embedded systems*, pages 207–228. Springer, 2015.

[53] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. Ecdsa key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1626–1638, 2016.

[54] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs. *Journal of Cryptographic Engineering*, 5(2):95–112, 2015.

[55] Robin Getz and Bob Moeckel. Understanding and eliminating emi in microcontroller applications. *National Semiconductor*, 1996.

[56] Nahid Farhady Ghalaty, Bilgiday Yuce, and Patrick Schaumont. Analyzing the efficiency of biased-fault based attacks. *IEEE Embedded Systems Letters*, 8(2):33–36, 2016.

[57] Nahid Farhady Ghalaty, Bilgiday Yuce, Mostafa Taha, and Patrick Schaumont. Differential fault intensity analysis. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 49–58. IEEE, 2014.

[58] Marjan Ghodrati. *Thwarting electromagnetic fault injection attack utilizing timing attack countermeasure*. PhD thesis, Virginia Tech, 2018.

[59] Archisman Ghosh, Debayan Das, Josef Danial, Vivek De, Santosh Ghosh, and Shreyas Sen. 36.2 an em/power sca-resilient aes-256 with synthesizable signature attenuation using digital-friendly current source and ro-bleed-based integrated local feedback and global switched-mode control. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 499–501. IEEE, 2021.

[60] Ashrujit Ghoshal, Sikhar Patranabis, and Debdeep Mukhopadhyay. Template-based fault injection analysis of block ciphers. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 21–36. Springer, 2018.

[61] Benedikt Gierlichs, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Revisiting higher-order dpa attacks. In *Cryptographers' Track at the RSA Conference*, pages 221–234. Springer, 2010.

[62] Christophe Giraud. Dfa on aes. In *International Conference on Advanced Encryption Standard*, pages 27–41. Springer, 2004.

[63] Anupam Golder, Debayan Das, Josef Danial, Santosh Ghosh, Shreyas Sen, and Arijit Raychowdhury. Practical approaches toward deep-learning-based cross-device power side-channel attack. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2720–2733, 2019.

[64] Jovan D. Golić and Christophe Tymen. Multiplicative masking and power analysis of aes. *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 198–212, 2002.

[65] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgí. A testing methodology for side channel resistance validation. *NIST noninvasive attack testing workshop*, pages 115–136, 2011.

[66] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In *International Workshop on Public Key Cryptography*, pages 199–211. Springer, 2003.

[67] Xiaofei Guo, Debdeep Mukhopadhyay, Chenglu Jin, and Ramesh Karri. Security analysis of concurrent error detection against differential fault analysis. *Journal of Cryptographic Engineering*, 5(3):153–169, 2015.

[68] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[69] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1095–1108, 2017.

[70] Yu-ichi Hayashi. State-of-the-art research on electromagnetic information security. *Radio Science*, 51(7):1213–1219, 2016.

[71] Yu-ichi Hayashi, Naofumi Homma, Takafumi Aoki, Yuichiro Okugawa, and Yoshiharu Akiyama. Transient analysis of em radiation associated with information leakage from cryptographic ics. In *2013 9th International Workshop on Electromagnetic Compatibility of Integrated Circuits (EMC Compo)*, pages 78–82. IEEE, 2013.

[72] Ludger Hemme. A differential fault attack against early rounds of (triple-) des. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 254–267. Springer, 2004.

[73] Naofumi Homma, Yu-ichi Hayashi, and Takafumi Aoki. Electromagnetic information leakage from cryptographic devices. In *2013 International Symposium on Electromagnetic Compatibility*, pages 401–404. IEEE, 2013.

[74] Microchip Technology Inc. Atmega328, 2019.

[75] Michael Keating, David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[76] Chong Hee Kim and Jean-Jacques Quisquater. New differential fault analysis on aes key schedule: Two faults are enough. In *International Conference on Smart Card Research and Advanced Applications*, pages 48–60. Springer, 2008.

[77] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[78] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[79] Gierad Laput, Chouchang Yang, Robert Xiao, Alanson Sample, and Chris Harrison. Em-sense: Touch recognition of uninstrumented, electrical and electromechanical objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 157–166, 2015.

[80] Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques Fournier, Bruno Robisson, and Assia Tria. A dfa on aes based on the entropy of error distributions. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 34–43. IEEE, 2012.

[81] Sae Lee. *High Efficiency Power Delivery for Chip Multiprocessors Using Voltage Stacking*. PhD thesis, 2016.

[82] Sae Kyu Lee, Tao Tong, Xuan Zhang, David Brooks, and Gu-Yeon Wei. A 16-core voltage-stacked system with adaptive clocking and an integrated switched-capacitor dc–dc converter. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(4):1271–1284, 2016.

[83] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: an approach based on machine learning. *International Journal of Applied Cryptography*, 3(2):97–115, 2014.

[84] Liran Lerman, Zdenek Martinasek, and Olivier Markowitch. Robust profiled attacks: should the adversary trust the dataset? *IET Information Security*, 11(4):188–194, 2017.

[85] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 320–334. Springer, 2010.

[86] Yannan Liu, Jie Zhang, Lingxiao Wei, Feng Yuan, and Qiang Xu. Dera: Yet another differential fault attack on cryptographic devices based on error rate analysis. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[87] Owen Lo, William J Buchanan, and Douglas Carson. Power analysis attacks on the aes-128 s-box using differential power analysis (dpa) and correlation power analysis (cpa). *Journal of Cyber Security Technology*, 1(2):88–107, 2017.

[88] Tal G Malkin, François-Xavier Standaert, and Moti Yung. A comparative cost/security analysis of fault attack countermeasures. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 159–172. Springer, 2006.

[89] John G Maneatis and Mark A Horowitz. Precise delay generation using coupled oscillators. *IEEE Journal of Solid-State Circuits*, 28(12):1273–1282, 1993.

[90] Stefan Mangard. A simple power-analysis (spa) attack on implementations of the aes key expansion. In *International Conference on Information Security and Cryptology*, pages 343–358. Springer, 2002.

[91] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[92] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked aes hardware implementations. *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 157–171, 2005.

[93] Martin Marinov. Remote video eavesdropping using a software-defined radio platform. *MS thesis, University of Cambridge*, 2014.

[94] Zdenek Martinasek, Felix Iglesias, Lukas Malina, and Josef Martinasek. Crucial pitfall of dpa contest v4. 2 implementation. *Security and Communication Networks*, 9(18):6094–6110, 2016.

[95] Zdenek Martinasek, Vaclav Zeman, Lukas Malina, and Josef Martinasek. K-nearest neighbors algorithm in profiling power analysis attacks. *Radioengineering*, 25(2):365–382, 2016.

[96] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? an a priori statistical power analysis of leakage detection tests. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 486–505. Springer, 2013.

[97] Luke Mather, Elisabeth Oswald, and Carolyn Whitnall. Multi-target dpa attacks: Pushing dpa beyond the limits of a desktop computer. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 243–261. Springer, 2014.

[98] James Clerk Maxwell. Viii. a dynamical theory of the electromagnetic field. *Philosophical transactions of the Royal Society of London*, (155):459–512, 1865.

[99] Thomas S Messerges. Securing the aes finalists against power analysis attacks. In *International Workshop on Fast Software Encryption*, pages 150–164. Springer, 2000.

[100] Thomas S Messerges. Using second-order power analysis to attack dpa resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 238–251. Springer, 2000.

[101] Thomas S Messerges, Ezzat A Dabbish, and Robert H Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE transactions on computers*, 51(5):541–552, 2002.

[102] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Investigations of power analysis attacks on smartcards. *Smartcard*, 99:151–161, 1999.

[103] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Power analysis attacks of modular exponentiation in smartcards. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 144–157. Springer, 1999.

[104] Patrick Emmanuel Meyer, Colas Schretter, and Gianluca Bontempi. Information-theoretic feature selection in microarray data using variable complementarity. *IEEE Journal of Selected Topics in Signal Processing*, pages 261–274, 2008.

[105] Oliver Mischke, Amir Moradi, and Tim Güneysu. Fault sensitivity analysis meets zero-value attack. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 59–67. IEEE, 2014.

[106] Yannick Monnet, Marc Renaudin, Régis Leveugle, Christophe Clavier, and Pascal Moitrel. Case study of a fault attack on asynchronous des crypto-processors. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 88–97. Springer, 2006.

[107] Amir Moradi, Mohammad T Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against aes cryptosystem. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 91–100. Springer, 2006.

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

[109] Radu Muresan and Stefano Gregori. Protection circuit against differ- ential power analysis attacks for smart cards. *IEEE Transactions on Computers*, pages 1540–1549, 2008.

[110] Radu Muresan, Vahedi Haleh, and Stefano Gregori. On-chip current flattening circuit with dynamic voltage scaling. *IEEE International Symposium on Circuits and Systems*, pages 4–pp, 2006.

[111] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. Eddie: Em-based detection of deviations in program execution. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 333–346, 2017.

[112] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order dpa attacks for masked smart card implementations of block ciphers. In *Cryptographers' Track at the RSA Conference*, pages 192–207. Springer, 2006.

[113] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In *International workshop on fast software encryption*, pages 413–423. Springer, 2005.

[114] Henry W Ott. *Electromagnetic compatibility engineering*. John Wiley & Sons, 2011.

[115] Colin O'Flynn and Zhizhang David Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 243–260. Springer, 2014.

[116] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[117] Sikhar Patranabis, Abhishek Chakraborty, Debdeep Mukhopadhyay, and PP Chakrabarti. Using state space encoding to counter biased fault attacks on aes countermeasures. *IACR Cryptol. ePrint Arch.*, 2015:806, 2015.

[118] Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration*, 40(1):52–60, 2007.

[119] Stefania Perri, Corsonello Pasquale, and Martin Margala. An integrated countermeasure against differential power analysis for secure smart- cards. *IEEE International Symposium on Circuits and Systems*, pages 4–pp, 2006.

[120] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *International workshop on cryptographic hardware and embedded systems*, pages 77–88. Springer, 2003.

[121] Michel Pollet. Simavr, 2017.

[122] Mikhail Popovich, Andrey Mezhiba, and Eby G Friedman. *Power distribution networks with on-chip decoupling capacitors*. Springer Science & Business Media, 2007.

[123] Thomas Popp. An introduction to implementation attacks and countermeasures. *IEEE/ACM international conference on Formal Methods and Models for Codesign*, pages 108–115, 2009.

[124] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *International Conference on Research in Smart Cards*, pages 200–210. Springer, 2001.

[125] Tauhidur Rahman, Domenic Forte, Jim Fahrny, and Mohammad Tehranipoor. Aro-puf: An aging-resistant ring oscillator puf design. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 69. European Design and Automation Association, 2014.

[126] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. Scaul: Power side-channel analysis with unsupervised learning. *IEEE Transactions on Computers*, 69(11):1626–1638, 2020.

[127] Mark Randolph and William Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptography*, 4(2):15, 2020.

[128] Ronald D. Williams Ratanpal, Girish B., and Travis N. Blalock. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, pages 179–189, 2004.

[129] Matthieu Rivain. Differential fault analysis on des middle rounds. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 457–469. Springer, 2009.

[130] Bruno Robisson and Pascal Manet. Differential behavioral analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 413–426. Springer, 2007.

[131] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A diagonal fault attack on the advanced encryption standard. *IACR Cryptol. ePrint Arch.*, 2009(581), 2009.

[132] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. Accuracy enhancement of electromagnetic side-channel attacks on computer monitors. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–9, 2018.

[133] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. Electromagnetic side-channel attacks: potential for progressing hindered digital forensic analysis. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, pages 138–143, 2018.

[134] Jörn-Marc Schmidt and Christoph Herbst. A practical fault attack on square and multiply. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 53–58. IEEE, 2008.

[135] Gustavus J Simmons. Symmetric and asymmetric encryption. *ACM Computing Surveys (CSUR)*, 11(4):305–330, 1979.

[136] Steven W Smith et al. *The scientist and engineer's guide to digital signal processing*, volume 14. California Technical Pub. San Diego, 1997.

[137] Petr Socha, Vojtěch Miškovskỳ, Hana Kubátová, and Martin Novotnỳ. Optimization of pearson correlation coefficient calculation for dpa and comparison of different approaches. In *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 184–189. IEEE, 2017.

[138] Petr Socha, Vojtech Miškovskỳ, Hana Kubátová, and Martin Novotnỳ. Correlation power analysis distinguisher based on the correlation trace derivative. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 565–568. IEEE, 2018.

[139] NA Sohaib ul Hassan. Em side channel analysis on complex soc architectures. Master's thesis, 2016.

[140] Francois-Xavier Standaert, Loic van Oldeneel tot Oldenzeel, David Samyde, and Jean-Jacques Quisquater. Power analysis of fpgas: How practical is the attack? *International Conference on Field Programmable Logic and Applications*, pages 701–710, 2003.

[141] Barron Stone and Samuel Stone. Comparison of radio frequency based techniques for device discrimination and operation identification. In *11th International Conference on Cyber Warfare and Security: ICCWS2016, Academic Conferences and Publishing Limited*, page 475, 2016.

[142] Barron D Stone and Samuel J Stone. Radio frequency based reverse engineering of microcontroller program execution. In *2015 National Aerospace and Electronics Conference (NAECON)*, pages 159–164. IEEE, 2015.

[143] Samuel J Stone, Michael A Temple, and Rusty O Baldwin. Detecting anomalous programmable logic controller behavior using rf-based hilbert transform features and a correlation-based verification process. *International Journal of Critical Infrastructure Protection*, 9:41–51, 2015.

[144] Qizhi Tian and Sorin A Huss. A general approach to power trace alignment for the assessment of side-channel resistance of hardened cryptosystems. In *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 465–470. IEEE, 2012.

[145] Carlos Tokunaga and David Blaauw. Secure aes engine with a local switched-capacitor current equalizer. In *2009 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, pages 64–65. IEEE, 2009.

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

[147] Jason Waddle and David Wagner. Towards efficient second-order power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–15. Springer, 2004.

[148] Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The myth of generic dpa... and the magic of learning. In *Cryptographers' Track at the RSA Conference*, pages 183–205. Springer, 2014.

[149] Marc Witteman and Martijn Oostdijk. Secure application programming in the presence of side channel attacks. In *RSA conference*, volume 2008, 2008.

[150] Howard Hua Yang and John Moody. Data visualization and feature selection: New algorithms for nongaussian data. *Advances in Neural Information Processing Systems*, pages 687–693, 2000.