EXTENDING DIFFERENTIAL FAULT ANALYSIS TO DYNAMIC S-BOX ADVANCED ENCRYPTION STANDARD IMPLEMENTATIONS

THESIS

Bradley M. Flamm, Civilian

AFIT-ENG-T-14-S-08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-T-14-S-08

EXTENDING DIFFERENTIAL FAULT ANALYSIS TO DYNAMIC S-BOX ADVANCED ENCRYPTION STANDARD IMPLEMENTATIONS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Bradley M. Flamm, B.S.M.

Civilian

September 2014

AFIT-ENG-T-14-S-08

EXTENDING DIFFERENTIAL FAULT ANALYSIS TO DYNAMIC S-BOX ADVANCED ENCRYPTION STANDARD IMPLEMENTATIONS

Bradley M. Flamm, B.S.M.
Civilian

Approved:

//signed//	2 Sep 2014
Maj Thomas E. Dube, PhD (Chairman)	Date
//signed//	2 Sep 2014
Dr. Brett J. Borghetti (Member)	Date
//signed//	2 Sep 2014
Dr. Mark E. Oxley (Member)	Date

AFIT-ENG-T-14-S-08

Abstract

Advanced Encryption Standard (AES) is a worldwide cryptographic standard for symmetric key cryptography. Many attacks try to exploit inherent weaknesses in the algorithm or use side channels to reduce entropy. At the same time, researchers strive to enhance AES and mitigate these growing threats. This paper researches the extension of existing Differential Fault Analysis (DFA) attacks, a family of side channel attacks, on standard AES to Dynamic S-box AES research implementations. Theoretical analysis reveals an expected average keyspace reduction of $2^{-88.9323}$ after one faulty ciphertext using DFA on the State of Rotational S-box AES-128 implementations. Experimental results revealed an average $2^{-88.8307}$ keyspace reduction and confirmed full key recovery is possible.

Table of Contents

Page

Page

List of Figures

List of Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
AK	*AddRoundKey*
CUT	Component Under Test
DFA	Differential Fault Analysis
DFE	Differential Fault Equations
MC	*MixColumns*
NIST	National Institute of Standards and Technology
RAES	Rotational S-box AES
rcon	*RoundConstant*
RMA	Round Modification Analysis
RRA	Round Reduction Analysis
RW	*RotateWord*
S-box	Substitution Box
SB	*SubBytes*
SBR	*S-boxRotation*
SR	*ShiftRows*
SUT	System Under Test
SW	*SubWord*

EXTENDING DIFFERENTIAL FAULT ANALYSIS TO DYNAMIC S-BOX ADVANCED ENCRYPTION STANDARD IMPLEMENTATIONS

I. Introduction

1.1 Motivation

Data security is a growing concern as more information transitions into digital formats. Toward this end, the National Institute of Standards and Technology (NIST) establishes the encryption algorithm standards and best practices within the United States. The current standard for general purpose data encryption, established in 2001, is the Advanced Encryption Standard (AES) [1]. As the quantity and sensitivity of data entrusted to AES grows, so does the incentive to compromise and reveal these secrets, thus many attacks try to exploit inherent weaknesses in the algorithm or use side channels to reduce entropy, such as Differential Fault Analysis (DFA). At the same time, continuing research strives to bolster the security of AES and mitigate these growing threats. One such area of research replaces a static component of the AES algorithm, the Substitution Box (S-box), with a dynamic version. This research extends an existing DFA attack to several research based Dynamic S-box AES implementations.

1.2 Research Objectives

The following itemizes the objectives of this research.

- **Determine if current DFA attacks extend to Dynamic S-box AES variants.** Both cryptanalysis and cryptography are complex and dependent on the smallest details. The consequences of changing any part of the target algorithm are not obvious, and refitting an existing attack to a similar but new algorithm is non-trivial.

- **Reveal expected keyspace reduction power of DFA extensions.** Existing attacks use probabilistic theoretical analysis for computing expected keyspace reduction power.

- **Build functional attacks which demonstrate full key and plaintext recovery.** Working examples of encryption and attack variants enable verification of theoretical results while providing tools for future use and analysis.

- **Provide an easy to follow and self-contained resource which walks through the mechanics and analysis of DFA attacks.** Current research provides pointed discussions of advanced methods [7, 9, 16–18, 24, 26], however basic understanding requires less powerful methods [4, 13, 28]. Although fragmenting analysis makes new research lightweight, it burdens nonexperts.

- **Improve the overall security analysis of Dynamic S-box AES variants.** Often, research does not thoroughly test new encryption proposals. Certain test suites and standards exist which ensure a few properties hold which are necessary, but not sufficient for a secure cryptographic system [3]. Rigorous analysis and testing of algorithms requires significant time, expertise and incentive. Thus, both white and black hats often focus on widespread standards over young and unadopted alternatives.

- **Contribute to the literature of theoretical analysis.** Existing work provides high-level analysis, but often omit lower level details and actual data. This research aims to address all levels of analysis, and building functional attacks creates actual data to verify existing and new theoretical claims.

- **Help inform and shape future discussions of cryptographic standards and algorithmic design decisions.**

1.3 Scope and Limitations

This research considers all DFA attacks as possible sources of extension, and considers all Dynamic S-box AES implementations as possible targets over which to extend. All analysis performed only applies to specific source-target implementation pairs chosen, but the leveraged concepts may yield results on other sources and targets. Due to the high level of complexity and resources required for actual realization of DFA attacks, this research instead relies on software simulated implementations.

1.4 Approach

Extending DFA attacks to AES variants is an untouched area of research. As the founding work, this research focuses on the simplest, non-trivial and interesting target-source combination. Background on each target and source enables a brief analysis for choosing this target-source combination. This research then extends the existing source DFA analysis to the chosen target AES variant. Implementing this new extended attack in software validates the new theoretical analysis and demonstrates actual realization.

1.5 Thesis Organization

The remainder of this document is as follows: Chapter 2 walks through the existing research including some basics of cryptography and field theory, current Dynamic S-box AES designs, and an overview of the existing DFA attacks on AES. Chapter 3 is a practice in theory, explicitly defining AES variants and performing the theoretical analysis of DFA extensions. Chapter 4 describes the methodology used to test and validate these attack extensions. Chapter 5 discusses the experimentation results, specifically their significance and how well they align with the theoretical analysis of Chapter 3. Lastly, Chapter 6 summarizes this work and discusses future areas of related research.

II. Background

This chapter covers a few basics of cryptology in Section 2.1, then walks through AES-128 in Section 2.2 and a few basics of field mathematics in Section 2.2.1. A discussion of Dynamic S-box schemes follows in Section 2.3. Section 2.4 introduces brute force attacks and their constraints. Finally, Section 2.5 introduces DFA attacks providing a comparison of attack power and constraints.

2.1 Cryptology

Cryptology encompasses both the study of keeping secrets, *cryptography*, and breaking them, *cryptanalysis* [27]. To secure information, an encryption algorithm transforms the clear *plaintext* message into a *ciphertext* using a secret encryption *key*. To reveal the secret message, a decryption algorithm transforms a ciphertext back into the plaintext using a secret decryption key. Encrypting with different keys results in different ciphertexts, and only the correct decryption key reveals the original plaintext. Figure 2.1 illustrates this black box view. By convention, the actors involved are Alice, who encrypts plaintexts and sends ciphertexts, and Bob, who receives the ciphertext and decrypts back to plaintexts. The attacker is Eve who has access to ciphertexts through various methods such as listening to network traffic.

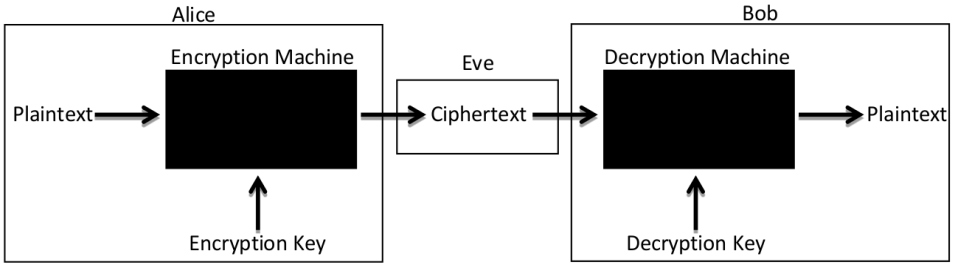

Figure 2.1: High-Level Encryption and Decryption.

2.1.1 Properties.

A 'good' cryptographic algorithm is one which is theoretically secure. That is, the algorithm leaks no information. Given arbitrary ciphertexts, Eve knows nothing about the associated plaintexts or keys. A few underlying principles and properties which are necessary, but not sufficient for a secure cryptographic algorithm follow.

- **Confusion.** The ciphertext does not relate in a simple way to the key [27].
- **Diffusion.** Each bit of the plaintext affects many bits of the ciphertext. Similarly, each bit of the ciphertext relies on many bits of the plaintext [27].
- **Avalanche Criterion.** Changing one bit of the plaintext or key should flip about half of the ciphertext bits [12].
- **Non-linearity.** A simple linear function (addition and multiplication) on the input cannot closely approximate the ciphertext.
- **Apparent Complete Randomness.** Produced ciphertexts statistically appear to be completely random.
- **Large Keyspace.** The encryption key size is sufficiently large enough to make a brute force attack infeasible (see Section 2.4).
- **Kerchkoff's Principle.** Algorithms should not rely on security through obscurity. Instead, Alice and Bob should always assume Eve knows the algorithm [27].

2.1.2 Attacks.

Cryptanalysis attacks conventionally divide into four categories based on the information available. This section includes a fifth category, side channel, which acts as an additional optional descriptor to supplement the first four. For the following explanatory situations the encryption and decryption machines use secret keys which the operator cannot access.

- **Ciphertext Only.** Eve only has access to ciphertexts, but no access to the encryption or decryption machines. Access to ciphertexts is always assumed, otherwise there would be no need for Alice to encrypt her messages to Bob.

- **Known Plaintext.** Eve has no access to the encryption or decryption machines, but has knowledge of what certain plaintext(s) encrypt to. This encompasses ciphertext only.

- **Chosen Plaintext.** Eve has access to the encryption machine. She can encrypt a number of plaintexts to manufacture associated (plaintext, ciphertext) pairs. This encompasses known plaintext and ciphertext only.

- **Chosen Ciphertext.** Eve has access to the decryption machine. She can decrypt a number of ciphertexts to manufacture associated (ciphertext, plaintext) pairs. This encompasses ciphertext only.

- **Side Channel.** Eve has access to information not directly tied to the algorithm, such as timing, processor sounds, power usage or outside information.

2.1.3 Algorithms.

Two main encryption schemes exist: symmetric and asymmetric (also known as private and public key). In an asymmetric (public key) algorithm, decryption is a function which acts upon the ciphertext to restore it to the plaintext, but decryption is not the inverse of encryption. Encryption is a computationally efficient function, but the inverse is computationally inefficient, such as factoring a large number. As a result decryption is a different function which relies on a different key to efficiently undo the work of encryption. RSA is the most recent standard public key algorithm [2]. In a symmetric (private key) algorithm, decryption is the inverse of encryption. That is, encryption is an easily invertible function reliant on a key. The same key enables both decryption and encryption. Often users and developers choose symmetric encryption schemes, rather than asymmetric, to

encrypt large volumes of data because of their increased speed. AES is the most recent standard symmetric key algorithm [2], and is what this paper examines.

2.2 AES

AES is a worldwide standard symmetric key encryption algorithm defined in [1]. Being symmetric, the same secret key enables both encryption and decryption of a particular message, and decryption is the inverse of encryption. Unprotected input messages are plaintexts, P, while the secure outputs are ciphertexts, C, both of length 128 bits. Figure 2.2 displays pseudocode for AES, and Section 2.2.3 further details the process.

```
def AES_Encryption(plaintext, key, size):
    # set the number of rounds to 10, 12 or 14
    nbrRounds = getNbrRounds(size)
    # the expanded keySize
    expandedKeySize = 16*(nbrRounds+1)
    # expand the key into 176, 208 or 240 byte key
    expandedKey = expandKey(key, size, expandedKeySize)
    # the input is stored by column
    state = transpose(plaintext)
    # round 0
    roundKey = createRoundKey(expandedKey, 0)
    state = addRoundKey(state, roundKey)
    # rounds 1-9,11 or 13
    for i in range(1, nbrRounds):
        roundKey = createRoundKey(expandedKey, 16*i)
        state = subBytes(state)
        state = shiftRows(state)
        state = mixColumns(state)
        state = addRoundKey(state, roundKey)
    # round 10, 12 or 14
    roundKey = createRoundKey(expandedKey, 16*nbrRounds)
    state = subBytes(state, False)
    state = shiftRows(state, False)
    state = addRoundKey(state, roundKey)
    # unmap the block again into the output
    ciphertext = transpose(state)
    return ciphertext
```

Figure 2.2: AES Pseudocode Based on [8].

The data of the algorithm at intermediate stages of encryption and decryption is the state, S. To establish a standard throughout this paper, referencing of the state uses up to three indexes: ${}^{i}\mathrm{S}_{k}^{j}$. The round index is i, j is the operation index and k is the byte index: ${}^{round}\mathrm{S}_{byte}^{operation}$. A 4x4 matrix of bytes represents each particular state ${}^{i}\mathrm{S}^{j}$ with $k \in \{0, 1, ..., 15\}$ indexed as shown below in Figure 2.3. Alternatively, $k \in$ {R0, R1, R2, R3} or $k \in$ {C0, C1, C2, C3} references a row or column of the state respectively, top to bottom and left to right, rather than a particular byte. Key lengths are 128, 192, or 256 bits with increased length corresponding to stronger theoretical cryptographic properties. These key lengths identify implementations of AES: AES-128, AES-192, and AES-256. Reference to the key is equivalently the secret key and the encryption key. The algorithm consists of four repeated steps performed on the state: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. Section 2.2.2 discusses each in detail.

S_0	S_1	S_2	S_3
S_4	S_5	S_6	S_7
S_8	S_9	S_{10}	S_{11}
S_{12}	S_{13}	S_{14}	S_{15}

Figure 2.3: Generic AES State Representation with Byte Indexing.

2.2.1 Galois Field 2^8.

AES uses the Galois Field GF(2^8), a number system, for mathematical manipulations of bytes, treating them as polynomials. GF(2^8) provides unique properties for calculation of all bit manipulations, hexadecimal notation simply improves portability and ease of storage. Each bit in the byte $b_7b_6b_5b_4b_3b_2b_1b_0$, where b_i is the i^{th} bit, represents the coefficient of

the x^i term of the polynomial

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0.$$

For example,

$$\text{0xA4} = 1010\ 0100 = 1x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 0x^1 + 0x^0 = x^7 + x^5 + x^2.$$

The base 2 in GF(2^8) represents that coefficients are in modulus two. The following example highlights addition.

$$\text{0xA4} + \text{0x86} = 1010\ 0100 + 1000\ 0110$$

$$= (1x^7+0x^6+1x^5+0x^4+0x^3+1x^2+0x^1+0x^0)+(1x^7+0x^6+0x^5+0x^4+0x^3+1x^2+1x^1+0x^0)$$

$$= (x^7 + x^5 + x^2) + (x^7 + x^2 + x)$$

$$= 2x^7 + x^5 + 2x^2 + x$$

$$= x^5 + x = 0010\ 0010 = \text{0x22}$$

Thus, addition is simply the bitwise XOR operation, that is,

$$\begin{array}{r} 1010\ 0100 \\ \oplus\ \underline{1000\ 0110} \\ 0010\ 0010 = \text{0x22}. \end{array}$$

This observation has several implications. It affirms the intuition that 0 is the additive identity `I`, since for any byte β, $\beta + 0 = \beta = \beta + \texttt{I}$. Also, the XOR of any number with itself is 0, so for any byte β, $\beta + \beta = 0$. From this property an inverse of addition exists and is, in fact, itself. For any byte α:

$$(\beta + \alpha) + \alpha = \beta + (\alpha + \alpha) = \beta + 0 = \beta$$

This fact enables further manipulation of equations. In the real numbers, performing the inverse of '+5' to both sides of the equation, $x + 5 = 9$, solves for x, resulting in $x + 5 - 5 = 9 - 5 \Rightarrow x = 4$. Similar manipulations are possible in GF(2^8). Supposing β is some unknown byte with the relation, $\beta + 0x14 = 0x96$, it is now possible to solve for β,

$$\beta + 0x14 + 0x14 = 0x96 + 0x14 \Rightarrow \beta = 0x82.$$

Additionally, it is impossible to add together any two numbers within GF(2^8) and end up with a number outside of GF(2^8). Because of this, GF(2^8) is said to be closed under addition. Further, addition in GF(2^8) is commutative, $\alpha + \beta = \beta + \alpha$, and associative, $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$. These are important and non-trivial properties. For perspective, subtraction is not commutative, $5 - 4 = 1 \neq -1 = 4 - 5$, or associative, $5 - (4 - 1) = 5 - (3) = 2 \neq 0 = (1) - 1 = (5 - 4) - 1$, over the Integers, and division is not closed over the Integers, $2 \div 3 = 2/3 \notin \mathbb{Z}$.

The exponent of 8 represents that eight powers of x, zero through seven, make up elements of GF(2^8). If multiplication achieves a power of x greater than or equal to eight, a reduction occurs using the irreducible polynomial for GF(2^8), $x^8 + x^4 + x^3 + x + 1$. This polynomial enables construction of GF(2^8), specifically the relation $x^8 + x^4 + x^3 + x + 1 = 0$. So, the equivalence relation for reducing polynomials to powers less than 8 is $x^8 = x^4 + x^3 + x + 1$. Using this relation the following example illustrates multiplication.

$$0xA4 \cdot 0x02 = 1010\ 0100 \cdot 0000\ 0010$$

$$= (1x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 0x^1 + 0x^0) \cdot (0x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 0x^0)$$

$$= (x^7 + x^5 + x^2) \cdot x$$

$$= x^8 + x^6 + x^3$$

$$= (x^4 + x^3 + x + 1) + x^6 + x^3$$

$$= x^6 + x^4 + 2x^3 + x + 1$$

$$= x^6 + x^4 + x + 1 = 0101\ 0011 = 0x53.$$

Because of the relation $x^8 = x^4 + x^3 + x + 1$, GF(2^8) is also closed under multiplication. Multiplying the reduction by an appropriate power of x reduces powers greater than 8. For example, $x^{11} = x^3 \cdot x^8 = x^3(x^4 + x^3 + x + 1) = x^7 + x^6 + x^4 + x^3$. This also illustrates that multiplication distributes over addition. The multiplicative identity, like in the Real Numbers, is 1. Lastly, every element aside from 0 has a multiplicative inverse (i.e., for every $\beta \neq 0$ there exists an α such that $\beta \cdot \alpha = 1$).

2.2.2 State Operations.

- ***SubBytes* (SB).** [Substitution] The S-box performs bytes substitutions. This transforms one byte at a time, altering every byte in the state matrix. The S-box is an 8-bit 16x16 table built from an affine transformation on multiplicative inverses which guarantees full permutation (S-box(a) ≠ a) and provides non-linearity [1, 25]. A table logically represents this substitution function such that the incoming higher order nibble identifies the row, while the lower nibble identifies the column. The corresponding table entry then replaces the incoming byte. This substitution function is fixed and well known. Figure 2.4 is a representation of the S-box. An example lookup is S-box(0x12) = 0xC9.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.4: AES S-box.

- ***ShiftRows* (SR).** [Rotation] This step cyclically shifts the bytes in each row providing inter-column diffusion. Iterating over every row, the i^{th} row rotates i bytes to the left, visually diagonalizing the columns for $i \in \{0, 1, 2, 3\}$. Figure 2.5 below illustrates the generic application of SR to the state.

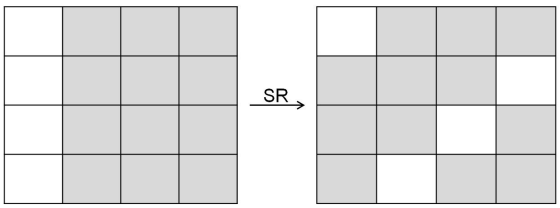

Figure 2.5: AES Shift Row operation.

- ***MixColumns* (MC).** [Linear Combination] An invertible linear transformation provides intra-column diffusion. A fixed and well-known matrix M multiplies with each column of the state, S_{Ci} for $i \in \{0, 1, 2, 3\}$. Figure 2.6 shows the multiplication of this fixed matrix with the first column of the state, $M \times S_{C0}$. Multiplication and addition are as defined in GF(2^8).

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} S_0 \\ S_4 \\ S_8 \\ S_{12} \end{bmatrix} = \begin{bmatrix} 2S_0 + 3S_4 + S_8 + S_{12} \\ S_0 + 2S_4 + 3S_8 + S_{12} \\ S_0 + S_4 + 2S_8 + 3S_{12} \\ 3S_0 + S_4 + S_8 + 2S_{12} \end{bmatrix}$$

Figure 2.6: Mix Column Example.

- ***AddRoundKey* (AK).** [Addition / Exclusive Or] This step integrates the round key with each state byte adding them together in the field (i.e., using the XOR function). Figure 2.7 illustrates the AK operation.

S_0	S_1	S_2	S_3		K_0	K_1	K_2	K_3		$S_0 \oplus K_0$	$S_1 \oplus K_1$	$S_2 \oplus K_2$	$S_3 \oplus K_3$
S_4	S_5	S_6	S_7	+	K_4	K_5	K_6	K_7	=	$S_4 \oplus K_4$	$S_5 \oplus K_5$	$S_6 \oplus K_6$	$S_7 \oplus K_7$
S_8	S_9	S_{10}	S_{11}		K_8	K_9	K_{10}	K_{11}		$S_8 \oplus K_8$	$S_9 \oplus K_9$	$S_{10} \oplus K_{10}$	$S_{11} \oplus K_{11}$
S_{12}	S_{13}	S_{14}	S_{15}		K_{12}	K_{13}	K_{14}	K_{15}		$S_{12} \oplus K_{12}$	$S_{13} \oplus K_{13}$	$S_{14} \oplus K_{14}$	$S_{15} \oplus K_{15}$

Figure 2.7: Generic Add Key Step.

2.2.3 Encryption.

Depending on the AES implementation (128, 192 or 256 bit key), the algorithm iterates over the state operations 10, 12 or 14 times creating rounds with an additional round zero application of AK, and the last round (10, 12 or 14) omitting MC. Figure 2.8 shows AES-128 encryption as a logical application of operations that form the rounds. Figure 2.9 depicts the AES-128 encryption algorithm left to right, top to bottom, and shows proper round and operation state indexing. The algorithm stores the plaintext into the state by column rather than by row, and similarly outputs ciphertext by column rather than by row.

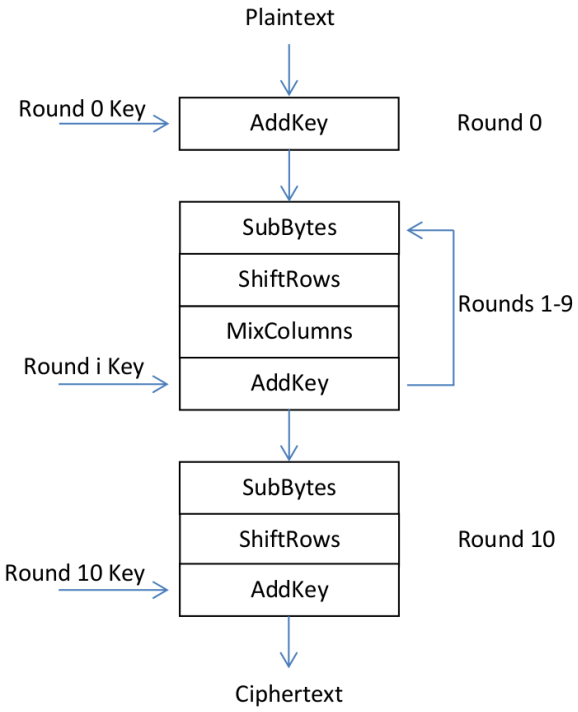

Figure 2.8: Logical AES-128 Encryption.

2.2.4 Decryption.

Decryption is simply the inverse of each step performed in the opposite order, using the round keys in reverse order. The inverse algorithm steps are *InverseSubBytes* (SB^{-1}), *InverseShiftRows* (SR^{-1}), *InverseMixColumns* (MC^{-1}) and *AddRoundKey* (AK). Figure 2.10 shows the logical flow of decryption, bottom to top.

- **SB^{-1}.** The inverse S-box reverses the lookup process.
- **SR^{-1}.** The i^{th} row rotates i bytes to the right, $i \in [0, 3]$.
- **MC^{-1}.** Matrix multiplication with the inverse of the constant matrix used in MC.
- **AK.** XOR is its own inverse, thus $AK^{-1} = AK$, and AK is sufficient.

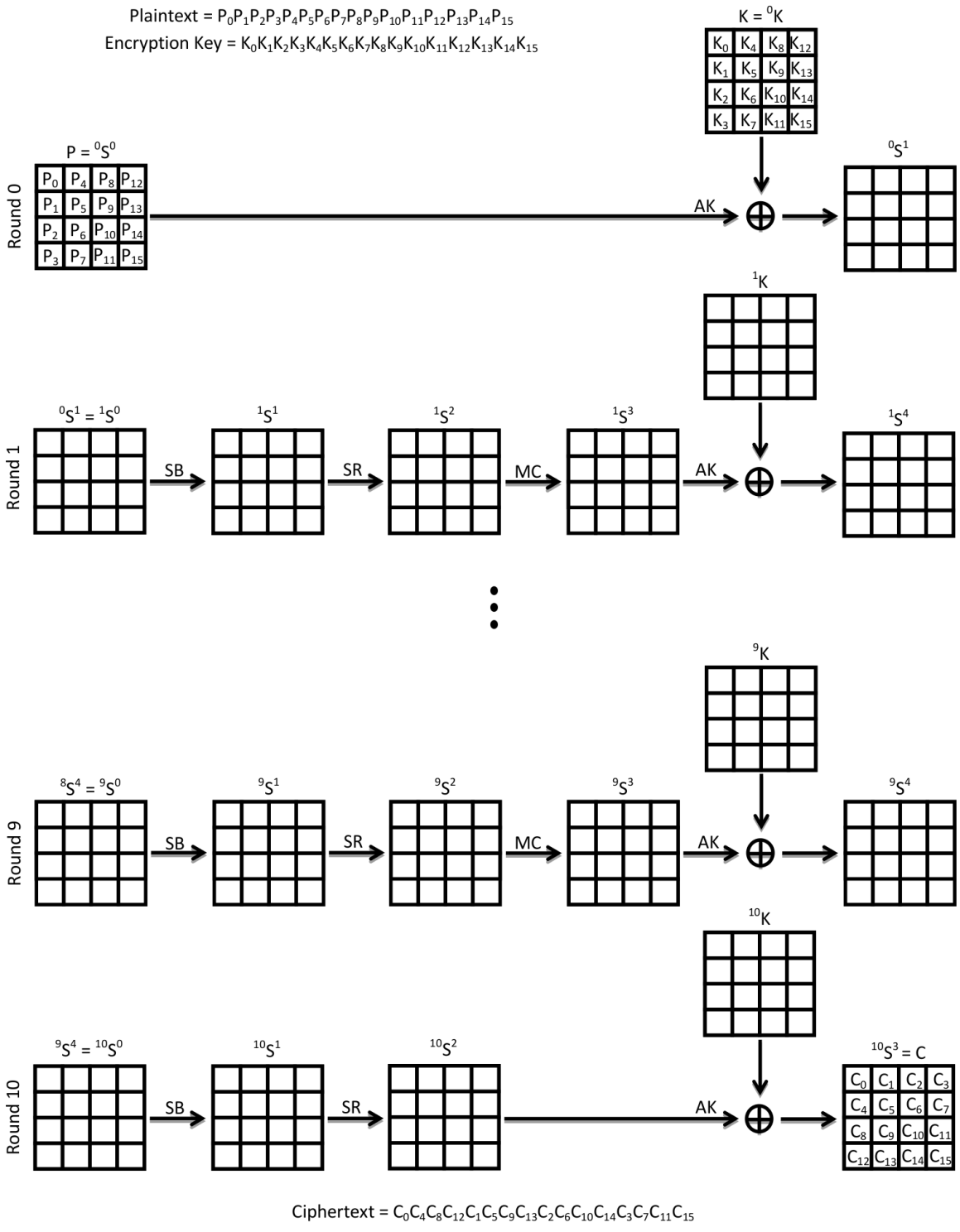

Figure 2.9: AES-128 Encryption.

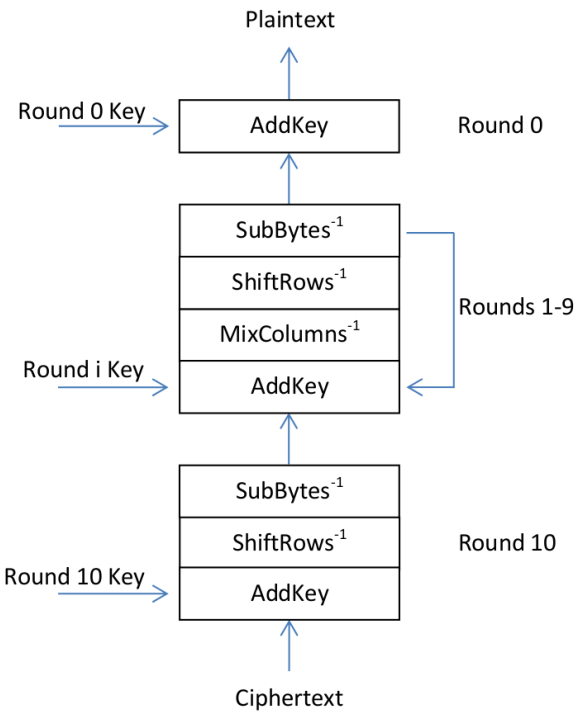

Figure 2.10: Logical AES-128 Decryption.

2.2.5 Key Expansion Algorithm.

Another important aspect of AES is the generation of round keys through expansion of the encryption key. AES-128 expands the 4x4 representation into a 4x44, or 11 4x4 round keys. Similarly, AES-192 expands 4x6 to 4x52, and AES-256 from 4x8 to 4x60. This expansion algorithm is the key schedule. Below are the relevant aspects of the process for AES-128; AES-192 and AES-256 are logically similar. The following list defines necessary operations and terminology.

- ***RotateWord* (RW).** Similar to the *ShiftRow* operation, a four byte word rotates one byte to the left, such that the first byte becomes the last byte (e.g., RW(01 AB DC EF) = AB DC EF 01).

- ***SubWord* (SW).** Similar to the *SubByte* operation, the S-box substitutes each byte of a four byte word.

- ***RoundConstant* (rcon).** Represents the exponentiation of 2 within GF(2^8), rcon(i) = x^{i-1}. Rounds 1-10 use an rcon value. Table 2.1 shows the computation of each of these values.

rcon(1)	=	x^0	=	1	=	0000 0001
rcon(2)	=	x^1	=	x	=	0000 0010
rcon(3)	=	x^2	=	x^2	=	0000 0100
rcon(4)	=	x^3	=	x^3	=	0000 1000
rcon(5)	=	x^4	=	x^4	=	0001 0000
rcon(6)	=	x^5	=	x^5	=	0010 0000
rcon(7)	=	x^6	=	x^6	=	0100 0000
rcon(8)	=	x^7	=	x^7	=	1000 0000
rcon(9)	=	x^8	=	$x^4 + x^3 + x + 1$	=	0001 1011
rcon(10)	=	x^9	=	$x^5 + x^4 + x^2 + x$	=	0011 0110

Table 2.1: Calculation of Rcon Values 1-10.

By convention, W is the expanded round key matrix with $W[i][j]$ denoting the i^{th} column [0-43], j^{th} byte [0-3]. As with storing the plaintext in ${}^0S^0$, the first four columns of W are the encryption key, filled by column. These first four columns are the round 0 key, with each subsequent set of four columns being the next round key. For columns 4-43: $W[i] = W[i-4] \oplus \beta$. If i is not divisible by 4 (i.e., if the current column is not the first column of a round key), then beta equals $W[i-1]$. However, if $i \bmod 4 = 0$ (i.e., the current column is the first column of round r's key), then beta equals SW(RW($W[i-1]$)) $\oplus$ [rcon(r), 0, 0, 0]. From this, round i key ${}^iK = [W(4i) - W(4i+3)]$. Figure 2.11 depicts the AES-128 key schedule.

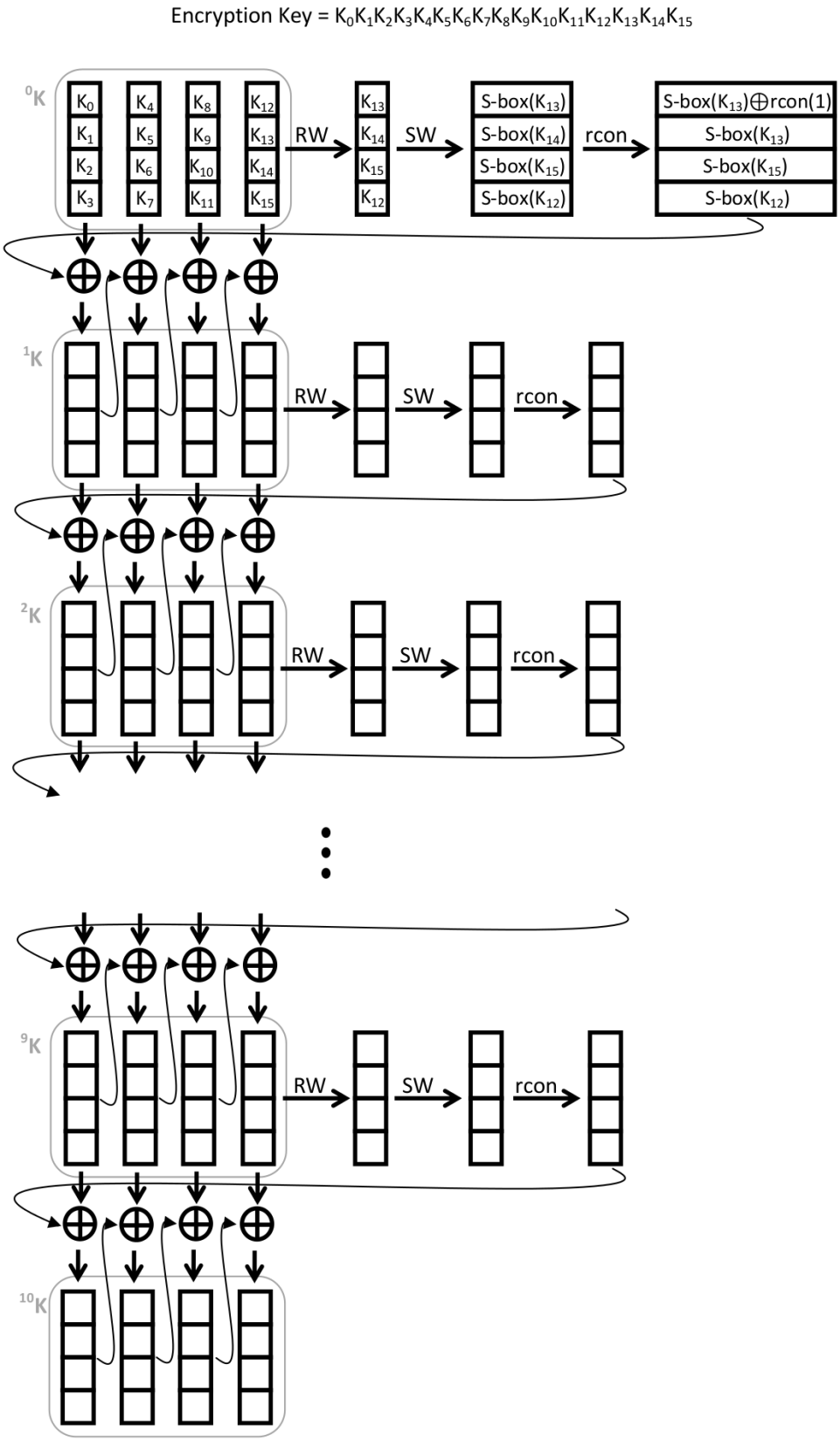

Figure 2.11: AES-128 Key Schedule.

Knowing the last round key, ^{10}K, enables reversal of the 128 bit key schedule since RW, SW, and rcon are all fixed and well known. Figure 2.12 illustrates this process. Normal use encryption and decryption never reverse the key schedule, instead always building up the expanded key from the encryption key. However, many attacks leverage this property; recovery of K_{10} reveals the encryption key.

2.3 Dynamic S-box

The S-box specifically is a common focus of research because it is the only operation adding non-linearity. Several dynamic S-box AES approaches exist including: S-box rotation [15, 22], chaotic S-box generation [10, 21, 30–32], switch S-boxes [5] and using different irreducible polynomials in GF(2^8) for S-box construction [6]. A brief explanation of each follows.

2.3.1 Rotational S-box.

This research variant of AES uses an altered key schedule to create two expanded keys: one for encryption, and one for rotation. The Rotational S-box variant also introduces a new algorithmic step, *S-boxRotation*, performed at the start of each round except round zero. Each round uses one of these 256 S-boxes. This new algorithm reportedly matches or slightly exceeds the performance of standard AES for diffusion through avalanche effect measures and Strict Avalanche Criterion [15, 22].

- ***S-boxRotation* (SBR).** Based on a manipulation of the round rotation key, the S-box rotates a specified amount (round rotation value) to the left. This rotation is a cyclic byte rotation, wrapping around from the top left to the bottom right of the S-box.

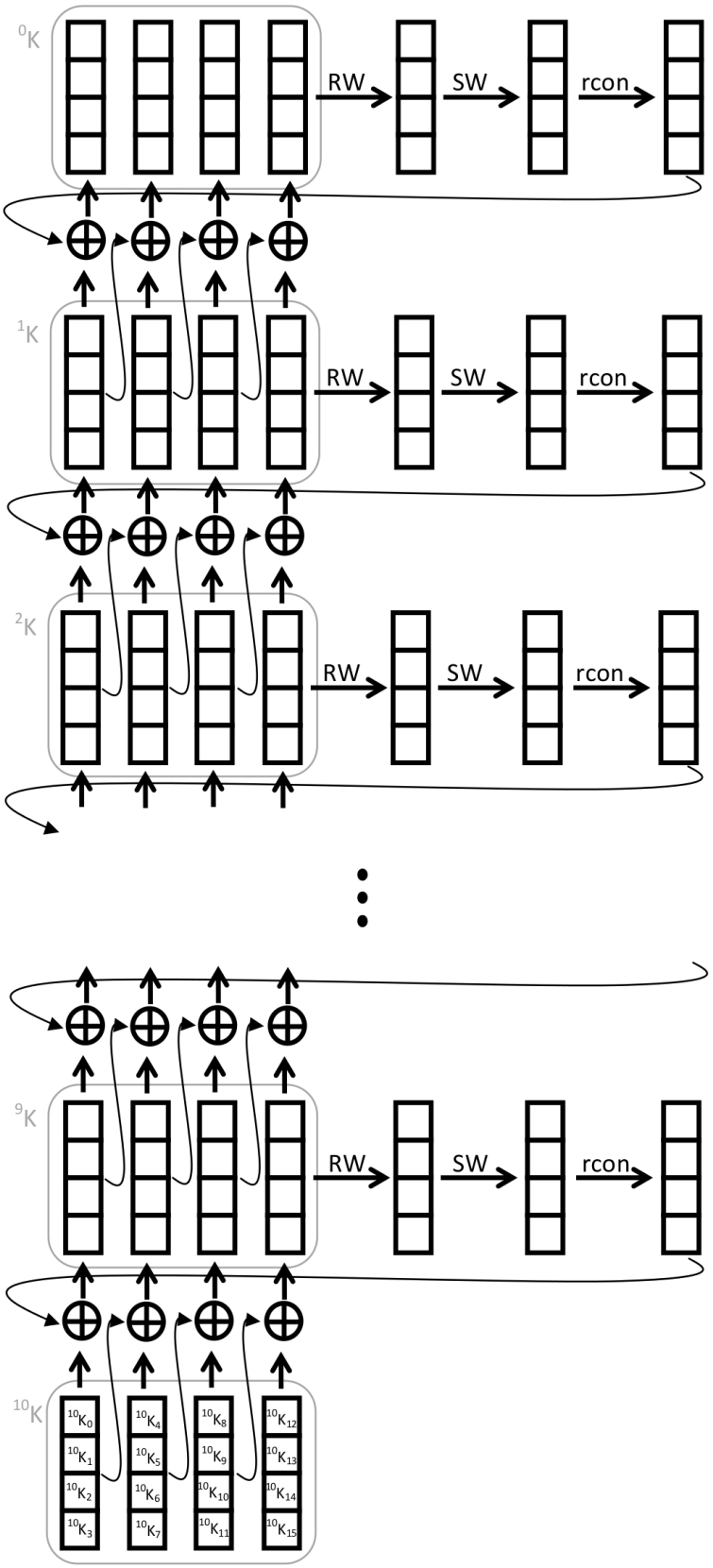

Figure 2.12: Reversal of the AES-128 Key Schedule.

2.3.2 Chaotic S-box.

This research variant of AES computes an S-box for each encryption key by applying a chaotic function on the encryption key. So, in AES-128, up to 2^{128} unique possible S-boxes exist, but each encryption only relies on one. Chaotic schemes are a popular choice for cryptographic applications due to their sensitivity to initial conditions, non-linearity, appearance of randomness, and determinism [10, 21, 30–32]. Existing methods include use of logistic map [10, 32], coupled map lattice of spatiotemporal chaos [21] and a piecewise linear chaotic map [30, 31].

2.3.3 Reduction S-box.

This research variant of AES allows choice of the S-box used by making the irreducible polynomial over GF(2^8), conventionally $x^8 = x^4 + x^3 + x + 1$, which the S-box construction uses by way of multiplicative inverses, an encryption parameter. Sending this polynomial with the key enables decryption [6]. Since 30 irreducible polynomials exist in GF(2^8), 30 possible S-boxes exist. No other logical changes apply to the algorithm.

2.3.4 Switch S-box.

This research variant of AES uses a pseudo-random number generator to determine if encryption uses the S-box or inverse S-box. Decryption then uses the other. Alice appends 0 or 1 to the ciphertext to signify which S-box decryption requires. So, each encryption uses one of two S-boxes [5].

2.4 Brute Force Attacks

Two types of brute force attacks exist, online and offline. In both types, Eve throws resources at the problem to test all possibilities until revealing the secret. Online brute force attacks do not require Eve to have any knowledge of a system. Instead, she attempts to use a password protected service, such as online banking or hard drive decryption, with every possible key (and potentially username). Because online brute force attacks rely on authenticating with a service, these attacks cannot leaverage precomputation and instead

occur in real time. Offline brute force attacks are a realization of a known plaintext, chosen plaintext or chosen ciphertext attacks. For example, if Eve knows E(P) = C, she can encrypt P with every possible K until a match of C is found. Offline brute force attacks can leverage precomputation and vary in complexity and approach ranging from exhaustive search and table lookup, to combinatory Time/Memory Tradeoff attacks such as Rainbow Tables. Brute force attacks succeed when computation time, block size, or key size are sufficiently small. Attacks on AES, and all cryptographically secure solutions, are infeasible by definition due to computation time and storage costs [27]. AES allows for an increase in both with its AES-192 and AES-256 implementations.

2.4.1 Time/Memory Trade-off.

If Eve has a known plaintext, one which remains constant and often used by Alice, such as a header "Dear Bob,", the time intensive extreme of this spectrum dictates Eve encrypting the plaintext with every possible key, and each time checking for a match against the ciphertext. This method is good for singular attacks, but quickly repeats a great deal of work if Alice changes the key. The memory intensive extreme of this spectrum has Eve encrypting this known plaintext with every possible key, and creating a dictionary of (ciphertext, key) entries. Now each time the key changes, Eve only needs to perform a lookup to obtain the new key. Compromises between these two extremes are often the best option, so as to reduce repeated work, while maintaining a reasonable storage burden. Similar trade-offs are commonplace within cryptanalysis, with the best option dictated by the attacker's available resources and goals.

2.4.2 Brute Force Mitigation Techniques.

As previously stated, the three algorithmic factors which affect the feasibility of a brute force attack are computation time, block size, and key size. The following list explores each in more detail.

- **Computation Time.** Often simply encryption time, this is the time required to try one possible key. Longer and more complex algorithms and artificial delays increase this burden. Artificial delays are practical against online brute force attacks where Eve must interface with a front end authentication rather than the encryption algorithm directly. Small increases significantly burden the attacker while remaining unnoticeable to users. Considering a hypothetical encryption system which encrypts in one nanosecond and a target algorithm that has 2^{50} keys, key recovery requires a maximum of:

$$2^{50} \text{ keys} \times \frac{1 \text{ sec}}{10^9 \text{ keys}} \times \frac{1 \text{ day}}{86400 \text{ sec}} \approx 13 \text{ days}.$$

  However, artificially suppressing the encryption time to 0.001 seconds, still apparently instantaneous to an end user, jumps this to:

$$2^{50} \text{ keys} \times \frac{1 \text{ sec}}{10^3 \text{ keys}} \times \frac{1 \text{ day}}{86400 \text{ sec}} \times \frac{1 \text{ year}}{365.25 \text{ days}} \approx 35,678 \text{ years}.$$

- **Block Size.** This is the amount of data encrypted at once. If Eve wants to store all the encryptions of a particular byte plaintext for an algorithm with 2^{40} keys (6 byte key + 1 byte ciphertext = 7 bytes per iteration), it requires:

$$\frac{7 \text{ bytes}}{1 \text{ iteration}} \times \frac{1 \text{ terabyte}}{1000^4 \text{ bytes}} \times 2^{40} \text{ iterations} \approx 7.7 \text{ terabytes}.$$

  This although quite large is not wholly unreasonable. If the block size increases from 1 byte to 16 bytes, this changes to (6 byte key + 16 byte ciphertext = 22 bytes per iteration) requiring:

$$\frac{22 \text{ bytes}}{1 \text{ iteration}} \times \frac{1 \text{ terabyte}}{1000^4 \text{ bytes}} \times 2^{40} \text{ iterations} \approx 24.2 \text{ terabytes}.$$

  Again actual storage space is feasible provided a specialized computing environment or a specific investment in storage, however efficiently managing and accessing this data becomes increasingly complex, especially if Eve targets several plaintexts.

- **Key Size.** This affects both storage and time constraints directly, however most restrictive to time. Assuming a hypothetical encryption system which encrypts in 1×10^{-15} seconds and a target algorithm that has 2^{80} keys, recovery requires a maximum of:

$$2^{80} \text{ keys} \times \frac{1 \text{ sec}}{10^{15} \text{ keys}} \times \frac{1 \text{ day}}{86400 \text{ sec}} \times \frac{1 \text{ year}}{365.25 \text{ days}} \approx 38 \text{ years.}$$

Although a substantial amount of time and computing power, conceivably secrets exist worth the investment. Increasing the keyspace to the equivalent of AES, this becomes impossible:

$$2^{128} \text{ keys} \times \frac{1 \text{ sec}}{10^{15} \text{ keys}} \times \frac{1 \text{ day}}{86400 \text{ sec}} \times \frac{1 \text{ year}}{365.25 \text{ days}} \approx 1.08 \times 10^{16} \text{ years.}$$

2.5 Differential Fault Analysis

DFA is a category of side channel attacks, which leverage physical implementations rather than theoretical weaknesses in the cryptographic algorithm. DFA relies on inducing faults through controllable external factors such as voltage fluctuations, clock cycle speed or a laser. These physical effects cause the current operation to resolve incorrectly, and just inject one or more random byte faults. The algorithm continues execution to completion, propagating the fault and creating a faulty ciphertext. This faulty ciphertext and its corresponding correct ciphertext, in conjunction with knowledge of timing and placement of the original fault allow for the construction of Differential Fault Equations (DFE). Solving these equations reduces the possible encryption key space, fully revealing the key or making brute force attacks feasible. Central to the success of these equations is the SubBytes operation.

DFA divide into four main categories: *DFA on the State* [4, 13, 16, 19, 26], *DFA on the Key Schedule* [7, 17], *Round Modification Analysis* [4, 9], and *DFA on the Algorithm* [4, 7, 24]. Location of fault induction and the assumptions made about the faults differentiate these attacks. The actual implementation and realization of these faults is

another area of research entirely outside the scope of this paper. However, [9, 14, 18, 24] demonstrate arbitrary assumptions about fault placement and timing are reasonable.

2.5.1 DFA on the State.

Fault injection logically produces a fault in the state just before the MixColumns operation of round r, a near terminal round. Actual injection of the fault can occur during any operation between MixColumns of rounds $r-1$ and r. Without loss of generality the fault is random and corrupts byte 0, S_0. MixColumns and ShiftRows propagate this fault, building up relations within the columns of the XOR of the correct and faulty ciphertext. Figure 2.13 shows fault propagation in AES-128 with the fault injected at ${}^{7}S_0^2$. Current versions of this attack fully recover the key with 2 faulty ciphertexts for AES-128, 2 for AES-192, and 3 for AES-256, and allow fault injection up to the third to last round while maintaining a reasonable level of complexity [16].

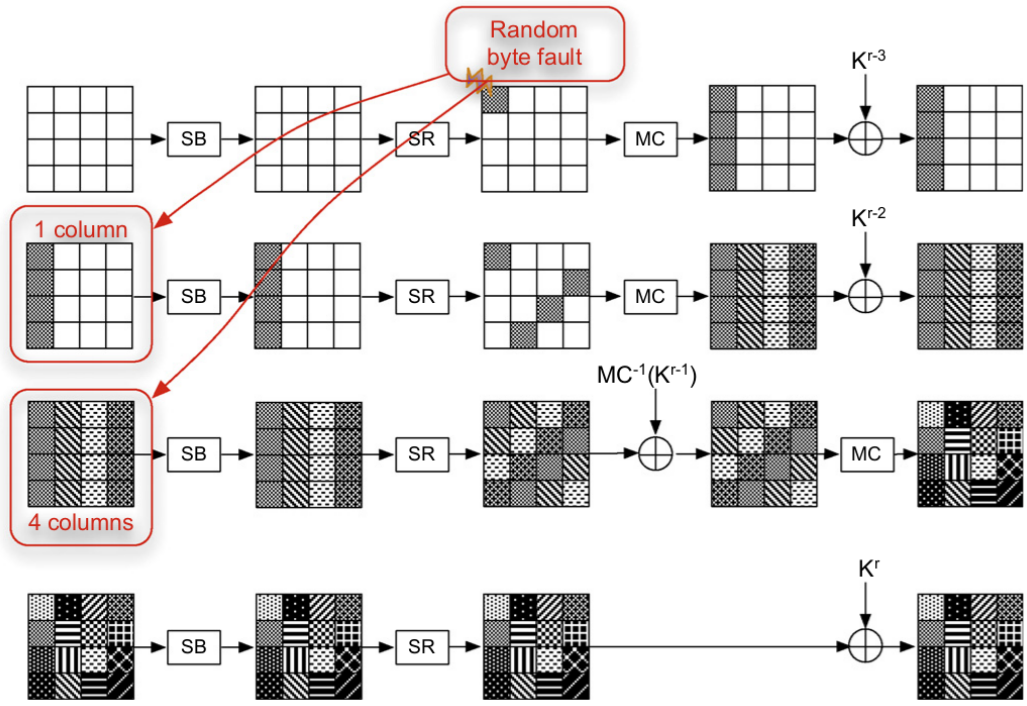

Figure 2.13: DFA on the State Fault Propagation in AES-128 [16].

2.5.2 *DFA on the Key Schedule.*

Fault injection occurs on the r^{th} round key. The fault then propagates throughout the state and the following round keys. Without loss of generality the fault is random and corrupts the first byte of the r^{th} key. More complex relations than those from DFA on the State build up from the XOR of the correct and faulty ciphertext. Figure 2.14 shows specifically how the fault spreads through the key schedule while Figure 2.15 shows fault propagation through the state and key. Current versions of this attack fully recover the key with 2 faulty ciphertexts for AES-128, 4 or 6 for AES-192, and 4 for AES-256, and allow fault injection up to the third to last round [17].

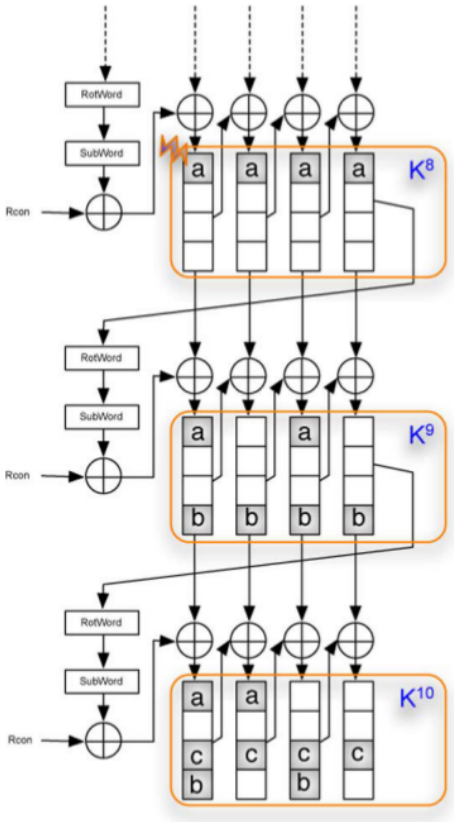

Figure 2.14: DFA on the Key Schedule Fault Propagation in AES-128 [17].

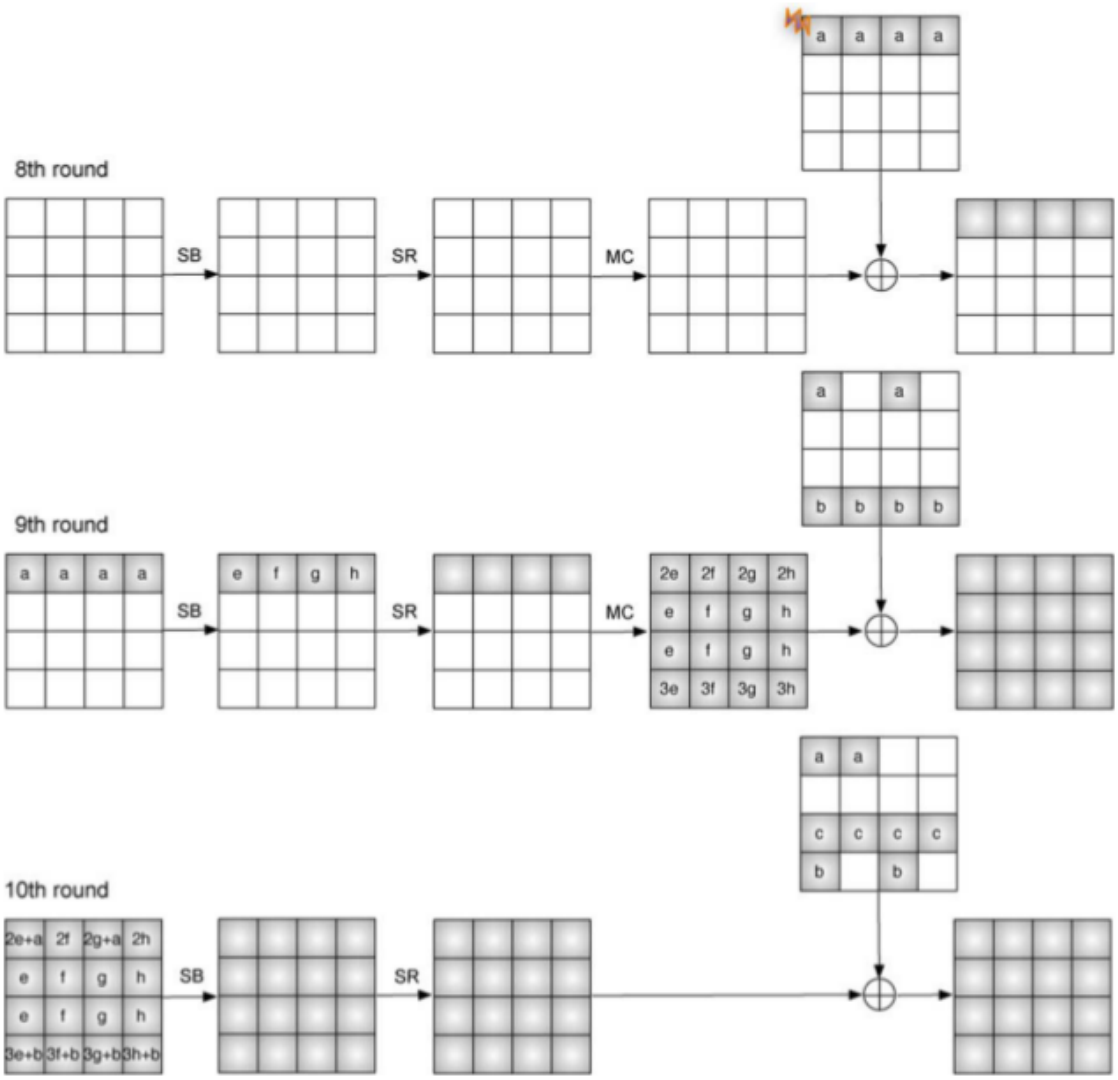

Figure 2.15: DFA on the Key Schedule Fault Propagation in AES-128 [17].

2.5.3 Round Modification Analysis.

Round Modification Analysis (RMA) is a generalization of Round Reduction Analysis (RRA), which induces a fault, changing the number of AES rounds executed. RRA reduces the number of rounds, typically to one or two, weakening the encryption significantly. RMA however allows for the possibility of increasing the rounds of AES, resulting in

faulty ciphertexts. Like other forms of fault analysis, these ciphertexts reduce the key range possibilities, making brute force attacks feasible [4, 9].

2.5.4 DFA on the Algorithm.

Although not explicitly defined in prior work, [4, 7, 24] exploit a fault induced into an algorithmic component such as the S-box or rcon. These attacks allow unique control and in some cases even enable known plaintext attacks. These often require explicit control over fault values.

2.5.5 DFA Mitigation Techniques.

Because DFA relies on inducing faults, error checking schemes mitigate this threat. Examples include recalculating the last several rounds of an encryption checking for a match, and timing analysis checking operations run their expected time [11, 16, 20, 23, 29]. Because these are an extra burden, minimum safeguards protect the most easily exploitable last rounds. Thus, research pushes successful DFA towards more complex and computationally expensive fault injections in earlier rounds, and more control over fault injection location and value.

2.6 Background Summary

AES is the current symmetric key cryptographic standard. As such, improving and attacking AES are continuous areas of research. One potential area of improvement uses a Dynamic S-box rather than the current fixed S-box. This potentially reduces the amount of viable precomputation possible in brute force attacks, adds additional complexity to the algorithm and increases encryption time. One current attack vector on AES is DFA. These attacks use correctly and incorrectly encrypted ciphertexts to build up relations that allow key recovery.

III. Theoretical Attack Analysis

This chapter discusses research design decisions and explains the attack extension. Section 3.1 defines the problem and outlines the approach. Section 3.2.1 discusses trade-offs and complexities of variant AES implementations, then explicitly defines the target variant in Section 3.2.2. A discussion of necessary assumptions and extensibility of DFA attacks follows in Section 3.2.3. Section 3.2.4 explains the existing theoretical analysis of the attack source. Finally, Section 3.3 provides theoretical attack extension analysis.

3.1 Problem Definition

3.1.1 Goals and Hypothesis.

The goal of this research is to determine the complexity of extending DFA to existing dynamic S-box AES designs. This research expects DFA attacks become more complex and difficult on a dynamic S-box AES design based on the additional complexity introduced. This research expands the overall security analysis of a dynamic S-box AES to assess its practicality and usefulness compared to the standard AES.

3.1.2 Approach.

This research employs probabilistic analysis to determine the keyspace reduction power of non-trivial DFA attack extensions to dynamic S-box AES research variants. Specifically choosing attack targets and sources guides analysis towards interesting and non-trivial extensions. These extensions use the concepts of existing work, while providing novel approaches where necessary.

3.2 Attack Targets and Sources

3.2.1 Potential Attack Targets.

As outlined in Section 2.3 the four possible Dynamic S-box designs are Rotational, Chaotic, Reduction, and Switch. 128 bit key length implementations are the base case, and, thus, the first step in extension. A high-level cursory consequences discussion follows.

- **Rotational.** Although this variant adds an additional operation, the AES encryption algorithm retains much of its structure. The S-box, though dynamic, relies on the existing AES S-box, with 256 total permutations each round, for 10 rounds, a total of 2^{80} possibilities from a simple operation on the existing structures. The S-box rotation appears to add a great deal of complexity with little additional effort or change to the algorithm.

- **Chaotic.** No change to the logical flow of the AES algorithm means current systems would only need to update the key schedule and S-box. However, building and storing the S-box for each encryption would likely limit the amount of optimization encryption hardware could perform. The potential increase of complexity is $256! \approx 8.5 \times 10^{506}$ for every possible S-box. Each key creates exactly one S-box, limiting this to 2^{128}. However, construction potentially creates any S-box, including the cryptographically broken. For example, the possibility exists that a key creates the identity S-box.

- **Reduction.** Within finite Galois fields, there exist only a certain number of irreducible polynomials. Only 30 exist for a Galois Field of size 2^8. This only introduces a complexity of about 2^5, which is a trivial work factor.

- **Switch.** Two S-box possibilities, both already employed, make this variation similar to AES when examined by necessary components. Updating to this S-box scheme would require the least work and would allow the most optimization. However,

this variation also provides the least increased complexity of 2. This increased complexity is only for offline attacks though, because by sending 0 or 1 in the clear, Eve knows which S-box to use, so there is no increase in complexity.

From this exploration, extending to the chaotic design would require significant computing power and analysis of chaotic properties, because no changes occur to the logical flow of the algorithm, but a huge pool of 2^{128} possible S-boxes exist. Reduction would be a trivial extension by repeating the attack 30 times or require no extra work if the irreducible polynomial identifier was sent 'in the clear' with the key, and switch would be no more complex, but simply require implementation. The rotational limit of 256 S-box options makes the work factor reasonable while the possibility of any one of these S-boxes used each round makes for interesting complexity. Thus rotational which does not alter the nature of the algorithm and adds complexity, while maintaining a feasible work factor is the most interesting and reasonable option to attack.

3.2.2 Attack Target Implementation.

As discussed in Section 2.3.1, Rotational S-box AES variants add an additional round operation, SboxRotation. Repeatedly applying this operation to the same S-box, rather than to the standard AES S-box each time, makes this iterative. Logically in programming, SBR is a function acting on an S-box passed by reference; rotating the S-box a specified amount. Additionally, this variant creates two expanded keys. AK uses the expanded encryption key, while SBR uses the expanded rotation key. Figure 3.1 illustrates the encryption process as rounds of operations. A slightly different key schedule creates these two expanded keys. Two key schedule schemes exist.

- **Key Schedule 1.** The S-box rotates by the XOR of all the bytes of the encryption key. Performing the key schedule as in normal AES, but using this rotated S-box for SubWords, creates an expanded key which is both the expanded encryption key, K, and the expanded rotation key, RK.

- **Key Schedule 2.** Key Schedule 1 creates an expanded rotation key, RK. The once rotated S-box used in Key Schedule 1 rotates a second time by the XOR of all the bytes of the expanded rotation key. Performing the key schedule as in normal AES, but using this twice rotated S-box for SubWords, creates the expanded encryption key, K.

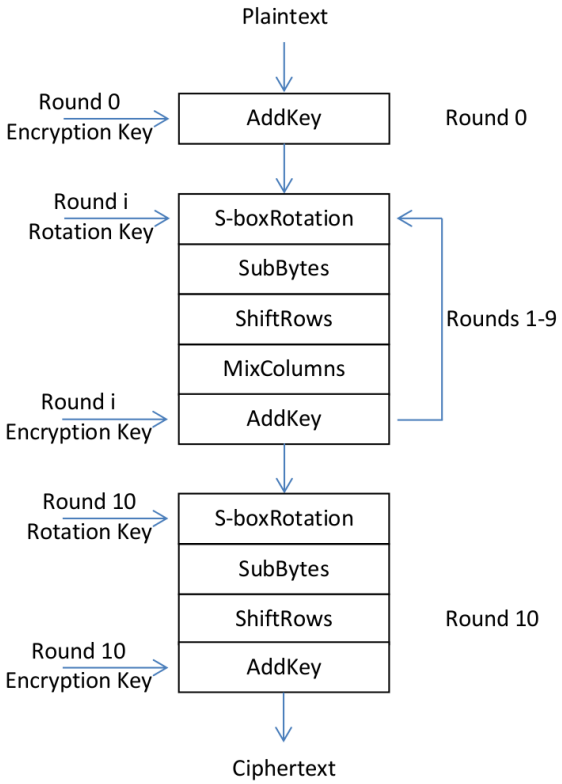

Figure 3.1: Logical RAES-128 Encryption.

Two reduction schemes exist to create rotation values from the round rotation key. This round rotation value designates by how much the S-box rotates in the associated round of encryption. SBR performs this rotation.

- **Rotation Reduction 1.** The round rotation value is the last byte, ${}^{i}\mathrm{RK}_{15}$ in the round rotation key.

- **Rotation Reduction 2.** The round rotation value is the XOR of all the bytes in the round rotation key.

Combinations of these Key Schedules and Rotation Reductions result in four proposed Rotational S-box AES (RAES) implementations labeled Types 1 through 4. Higher numbers relate to increased theoretical security due to complexity, confusion and computation time. Choice of key schedule is the primary security influence.

- **RAES Type 1.** Key Schedule 1 and Rotation Reduction 1
- **RAES Type 2.** Key Schedule 1 and Rotation Reduction 2
- **RAES Type 3.** Key Schedule 2 and Rotation Reduction 1
- **RAES Type 4.** Key Schedule 2 and Rotation Reduction 2

Decryption requires one extra step of priming the inverse S-box. The S-box used in round 10 of encryption is the standard AES S-box rotated 11 or 12 times depending on the Key Schedule used. Rotating the inverse S-box by these same values correctly orients it for decryption. Once correctly initialized, decryption follows as expected with SBR^{-1} a rotation of inverse S-box to the right by the round rotation value. Figure 3.2 illustrates this process.

Explicitly establishing the mechanisms of these implementations required several design decisions beyond the scope of [15, 22]. Appendix A justifies these decisions and discusses the alternatives. Appendix B provides sample encryptions and key schedules of all 4 Types to facilitate validation and future use of this algorithm.

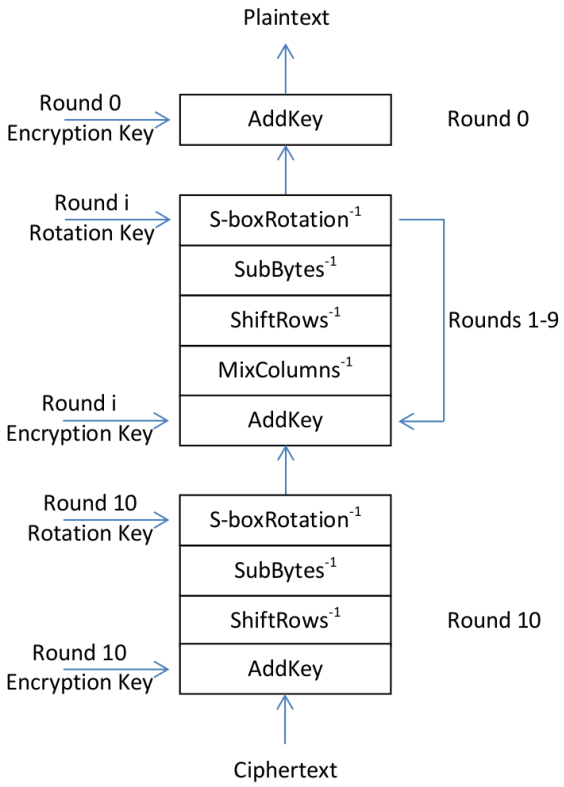

Figure 3.2: Logical RAES-128 Decryption.

3.2.3 Potential Attack Sources.

As outlined in Section 2.5 the four possible DFA attacks are: DFA on the State, DFA on the Key Schedule, Round Modification Analysis, and DFA on the Algorithm. Using a rotational S-box affects each uniquely. A high-level consequences discussion follows.

- **DFA on the State.** This requires no assumptions about the value of the fault injected. Faults only propagate through the state. Rotating the S-box does not affect the location of the propagation, only the values. Most likely, this extension could largely use existing work.

- **DFA on the Key Schedule.** This requires no assumptions about the value of the fault injected. Depending on RAES Type and the key expansion targeted, potentially only the values change, not location of faults. The altered key schedule increases the

complexity and analysis required for this attack, though most likely this extension could largely use existing work.

- **Round Modification Analysis.** This requires a controllable or predictable fault injection value. Leveraging a fault injection is the only DFA element of RMA. Use of a Rotational S-box does not significantly impact methods used in key recovery when altering the number of rounds. These methods are a set of more conventional cryptanalysis approaches.

- **DFA on the Algorithm.** The most abstract DFA category which allows many possibilities. Many creative options allow powerful attacks, but these attacks likely need the most control of values injected. DFA on the Algorithm is an open-ended class with no clear implementations to imitate.

From this analysis, DFA on the State and DFA on the Key Schedule are the most logical and interesting choices in identifying a non-trivial extension of an existing attack. This research pursues DFA on the State for the slightly less expected complexity. Extending to DFA on the State likely allows use of existing attack properties and analysis, while still requiring creative workarounds to the added complexity.

3.2.4 Attack Source Implementation.

Extending DFA on the State to RAES requires understanding of the existing attack. The theoretical analysis detailed in [28] examines probabilities that certian conditions hold to determine the attack's keyspace reduction power. What follows is a synopsis of this analysis.

Eve obtains a correct encryption E of plaintext P, using key, K. She then leverages attack capabilities to inject a fault at ${}^{8}S_{0}^{2}$, obtaining a faulty encryption $\bar{E}$ of plaintext P, using key K. The single byte fault propagates to corrupt the entire ciphertext. $\bar{S}$ and $\bar{C}$ represent the faulty state and ciphertext. Figure 3.3 represents the XOR of the last three

rounds of E and $\bar{E}$. ΔS and ΔC represent the state and ciphertext respectively. Assigning the difference between E and $\bar{E}$ at the fault injection site ${}^{8}\Delta S_0^2$ to the variable 'a', relations build up around this XOR difference. A walkthrough of fault propagation through each operation follows.

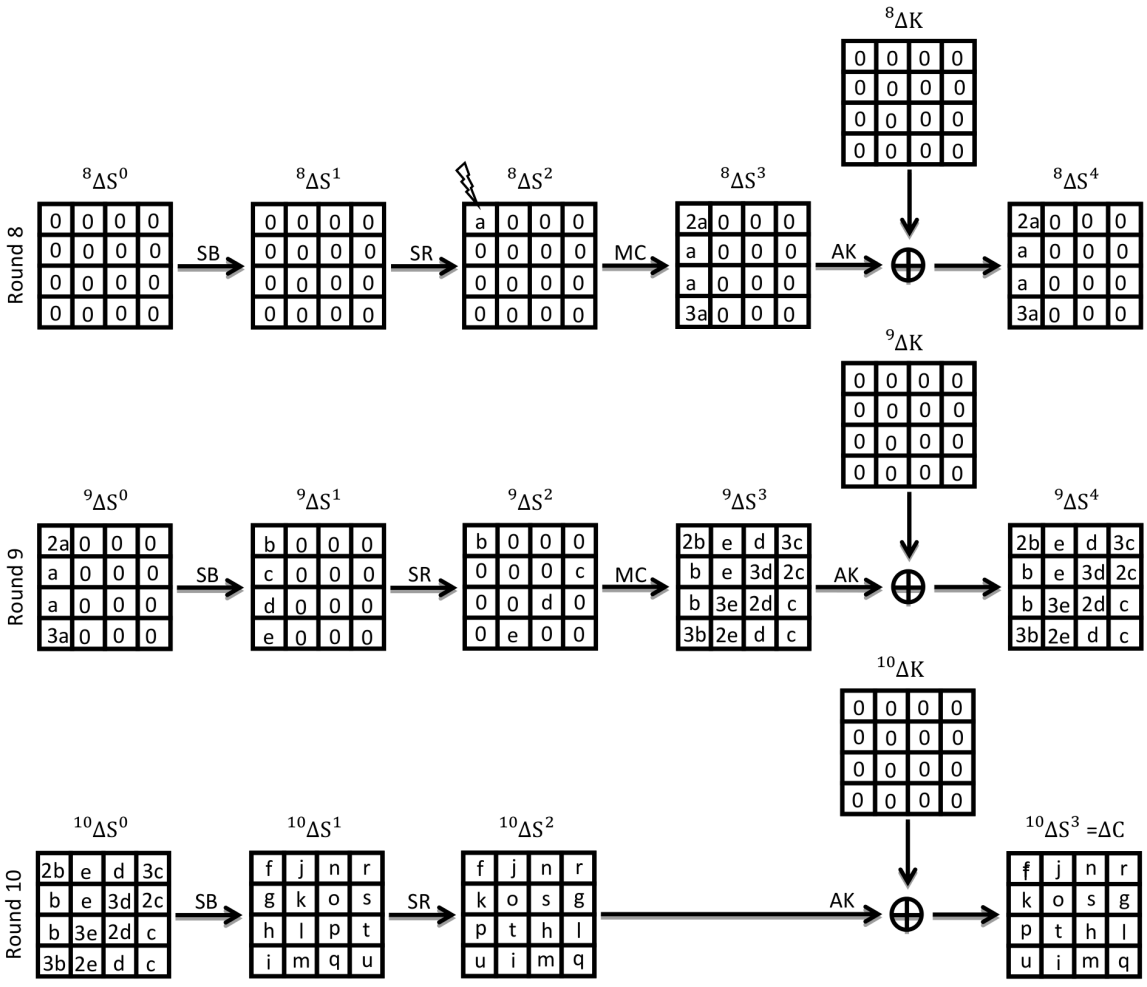

Figure 3.3: XOR of Correct and Faulty AES-128 Encryption, Rounds 8-10.

- **MC.** Figure 3.4 highlights the transition between ${}^{8}\Delta S^2$ and ${}^{8}\Delta S^3$. Figure 3.5 shows the underlying math. Because multiplication distributes over addition in GF(2^8),

$$MC({}^{8}S^2) \oplus MC({}^{8}\bar{S}^2) = MC({}^{8}S^2 \oplus {}^{8}\bar{S}^2) = MC({}^{8}\Delta S^2).$$

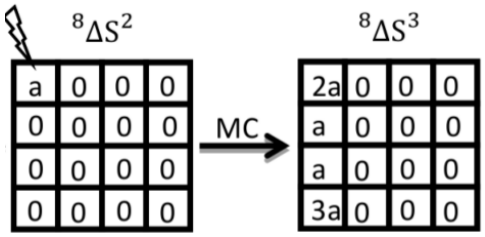

Figure 3.4: XOR of Correct and Faulty AES-128 MC Operation.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} {}^8\Delta S_0^2 \\ {}^8\Delta S_4^2 \\ {}^8\Delta S_8^2 \\ {}^8\Delta S_{12}^2 \end{bmatrix} = \begin{bmatrix} 2({}^8\Delta S_0^2) + 3({}^8\Delta S_4^2) + {}^8\Delta S_8^2 + {}^8\Delta S_{12}^2 \\ {}^8\Delta S_0^2 + 2({}^8\Delta S_4^2) + 3({}^8\Delta S_8^2) + {}^8\Delta S_{12}^2 \\ {}^8\Delta S_0^2 + {}^8\Delta S_4^2 + 2({}^8\Delta S_8^2) + 3({}^8\Delta S_{12}^2) \\ 3({}^8\Delta S_0^2) + {}^8\Delta S_4^2 + {}^8\Delta S_8^2 + 2({}^8\Delta S_{12}^2) \end{bmatrix} = \begin{bmatrix} 2({}^8\Delta S_0^2) + 3(0) + 0 + 0 \\ {}^8\Delta S_0^2 + 2(0) + 3(0) + 0 \\ {}^8\Delta S_0^2 + 0 + 2(0) + 3(0) \\ 3({}^8\Delta S_0^2) + 0 + 0 + 2(0) \end{bmatrix} = \begin{bmatrix} 2({}^8\Delta S_0^2) \\ {}^8\Delta S_0^2 \\ {}^8\Delta S_0^2 \\ 3({}^8\Delta S_0^2) \end{bmatrix}$$

Figure 3.5: XOR MC Walkthrough.

- **AK.** Figure 3.6 highlights the transition between $^8\Delta S^3$ and $^8\Delta S^4$. Because the fault injection does not corrupt the key schedule, the expanded key is the same for both E and $\bar{E}$. Thus, as shown in Figure 3.6, every byte of $^8\Delta K$ is 0. Because XOR, or addition in GF(2^8) is commutative, the order of performing this XOR does not matter, and so $^8\Delta S^4 = AK(^8\Delta S^3) = {}^8\Delta S^3 \oplus {}^8\Delta K = {}^8\Delta S^3$. The equations below demonstrates this relationship.

$$\begin{aligned} {}^8\Delta S_0^4 &= ({}^8S_0^3 \oplus {}^8K_0) \oplus ({}^8\bar{S}_0^3 \oplus {}^8K_0) \\ &= ({}^8S_0^3 \oplus {}^8\bar{S}_0^3) \oplus ({}^8K_0 \oplus {}^8K_0) \\ &= ({}^8S_0^3 \oplus {}^8\bar{S}_0^3) \oplus (0) \\ &= {}^8\Delta S_0^3 \end{aligned}$$

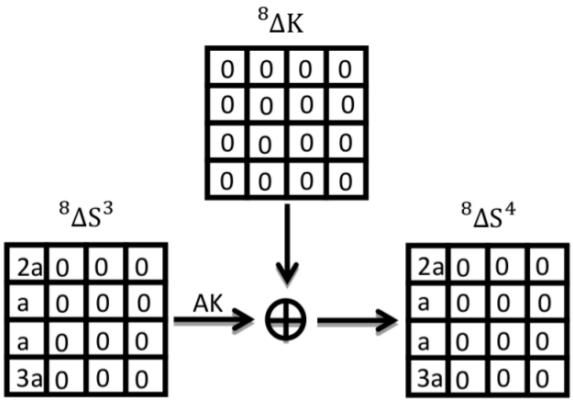

Figure 3.6: XOR of Correct and Faulty AES-128 AK Operation.

- **SB.** $^{9}\Delta S^{1} = SB(^{9}S^{0}) \oplus SB(^{8}\bar{S}^{0})$. Distributing SB over XOR is not possible.

$$SB(00) \oplus SB(01) = 63 \oplus 7C = 1F$$

$$SB(00 \oplus 01) = SB(01) = 7C \neq 1F$$

$$SB(02) \oplus SB(03) = 77 \oplus 7B = 0C \neq 1F$$

This prohibits further reductions, thus relations from $^{9}\Delta S^{0}$ cannot move forward into $^{9}\Delta S^{1}$. Figure 3.7 highlights this.

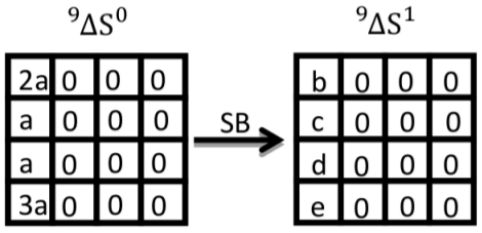

Figure 3.7: XOR of Correct and Faulty AES-128 SB Operation.

- **SR.** No manipulation of byte values occur in this step, thus the XOR values remain unchanged, but move byte position as dictated by the SR operation. Figure 3.8 shows this.

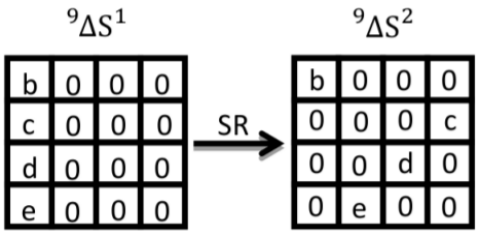

Figure 3.8: XOR of Correct and Faulty AES-128 SR Operation.

Clearly defined fault propagation allows discussion of the analysis to move forward. As the attacker, Eve only has C and $\bar{C}$, and thus ΔC. Knowing AK has no effect on ΔS values, $^{10}\Delta S^{2} = \Delta C$. Again, SR does not affect ΔS values, and so $^{10}\Delta S^{1} = SR^{-1}(\Delta C)$. Thus with C and $\bar{C}$, Eve also knows $^{10}\Delta S^{1}$. This attack exploits the known relations that exist in $^{10}\Delta S^{0}$, and the possible ^{10}K that enable $^{10}\Delta S^{1}$ to step back and satisfy these relations. The set of DFE to represent this for $^{10}\Delta S^{0}_{C0}$ follow.

$$2b = SB^{-1}(C_{0} \oplus {}^{10}K_{0}) \oplus SB^{-1}(\bar{C}_{0} \oplus {}^{10}K_{0}) \tag{3.1}$$

$$b = SB^{-1}(C_{7} \oplus {}^{10}K_{7}) \oplus SB^{-1}(\bar{C}_{7} \oplus {}^{10}K_{7})$$

$$b = SB^{-1}(C_{10} \oplus {}^{10}K_{10}) \oplus SB^{-1}(\bar{C}_{10} \oplus {}^{10}K_{10})$$

$$3b = SB^{-1}(C_{13} \oplus {}^{10}K_{13}) \oplus SB^{-1}(\bar{C}_{13} \oplus {}^{10}K_{13})$$

Since b can be any value a byte can hold, except 0, b is in {1, 2, ..., 255}. Were b zero, fault injection failed, and thus $C = \bar{C}$, so there is nothing to exploit. Examining the first equation of the set, regardless of what values C_0 and $\bar{C}_0$ hold, of the 255 possible values of 2b, 128 yield 0 $^{10}K_0$ key hypothesis which satisfy Equation 3.1, 126 yield 2 and 1 yields 4. Iterating over all possible values of C_0, $\bar{C}_0$ and $^{10}K_0$ reveal this. Thus, on average for any one particular value of 2b, there exists $(2 \times 126 + 4)/255 = 256/255$ just over 1 valid $^{10}K_0$ hypothesis. Considering all 255 possible values of 2b yields an expected $255 \times \frac{256}{255} = 256$ $^{10}K_0$ hypotheses. This result is not a reduction yet since $^{10}K_0$ is one byte which can hold one of $2^8 = 256$ values. The same holds for each of the four equations in the set.

Now considering all four equations at once, for a given value of b, each equation on average should return about one ${}^{10}K_i$ value. These four values form a quartet of key bytes $\{{}^{10}K_0,\ {}^{10}K_7,\ {}^{10}K_{10},\ {}^{10}K_{13}\}$ which is one hypothesis. Considering all 255 b values should create 256 of these quartets. This column analysis reduces the keyspace of these four bytes from $(2^8)^4$ to 2^8.

A set of equations like those seen above exist for each of the four columns of ${}^{10}\Delta S_0$, thus each of these reduce similarly. Each column is independent, making no further relations possible from these relationships in round 10. So, the original keyspace of $2^{128} = ((2^8)^4)^4$ reduces to $(2^8)^4 = 2^{32}$ when considering all combinations of these quartets. Equivalently, these sets of equations have a keyspace reduction power of 2^{-96}.

This analysis and process is the essence of the DFA attack. Reductions based on properties that must hold over the SB operation on the XOR of a correct and faulty ciphertext. Stepping back to round 9 produces a similar reduction, and building relations over ${}^{9}S^{0}_{C0}$ further reduces the keyspace to 2^8. However, leveraging round 9 information requires a much greater amount of work for a much smaller reduction. Fully reducing the ^{10}K keyspace to 1 requires additional C, $\bar{\text{C}}$ pairs. This produces two independently reduced ^{10}K keyspaces of 2^{32}. The intersection of these keyspaces yields one unified reduced keyspace. Keys should randomly appear in both with likelihood $2^{32} \times \frac{2^{32}}{2^{128}} = 2^{-64}$. Thus only the valid ^{10}K should remain. Once recovered, as discussed in Section 2.2.5 reversal of the key schedule reveals the original encryption key. The round 10 reduction has a work factor of $(2^8) \times 16 = 2^{12}$ since stepping each ${}^{10}\text{K}_i$ byte back occurs individually and has a keyspace reduction power of 2^{-96}. However the round 9 reduction has a work factor of 2^{32} since stepping back through to ${}^{9}S^{0}_{C0}$ requires calculating ^{9}K which relies on ^{10}K. Individually checking all 2^{32} possible keys has a reduction power of 2^{-24}. If Eve can only obtain one C, $\bar{\text{C}}$ pair and she knew the format of the unencrypted data, she could reasonably

perform this round 9 reduction and decrypt C with each of the 256 possible keys to see if any of the resulting plaintexts conform to the expected data format.

3.3 Attack Analysis

A rough estimate of the memory necessary to calculate the S-box relations used in the attack described in Section 3.2.2 is (possible C_i)×(possible $\bar{C}_i$)×(possible ${}^{10}K_i$)×(storage cost). Each calculation needs to store 4 bytes, C^i, $\bar{C}^i$, ${}^{10}K_i$, and b. Thus roughly $256 \times 256 \times 256 \times 4 = 2^{26}$ bytes, or roughly 0.067 GB. When pushing this analysis to the Rotational S-box, storage costs roughly become (possible round 10 S-box rotation values)×(possible C_i)×(possible $\bar{C}_i$)×(possible ${}^{10}K_i$)×(storage cost). Each calculation needs to store 5 bytes, r_{10}, C_i, $\bar{C}_i$, ${}^{10}K_i$ and b. Thus roughly $256\times256\times256\times256\times5 \approx 2^{34.32}$ bytes, or approximately 21.5 GB. While this may not be a burden for supercomputers or specialized workstations, it is beyond the processing power of most personal workstations. As such, extension requires a different analysis approach. The existing fault propagation model holds, but reduction requires knowing the S-box used in round 10.

3.3.1 Rotation Step Analysis.

Examining the SBR operation, Figure 3.9 shows the standard, unrotated S-box, S-box_0. Looking up 0x02: S-box_0(0x02) = 0x77. Rotating S-box_0 by one results in S-box_1, SBR(1, S-box_0) = S-box_1, Figure 3.10 displays this new rotated S-box. Again looking up 0x02: S-box_1(0x02) = 0x7b. Looking up 0x03 in S-box_0 achieves this same result. Rotating S-box_1 by one results in S-box_2, SBR(1, S-box_1) = S-box_2. Figure 3.11 visualizes this twice rotated S-box. Looking up 0x02 again: S-box_2(0x02) = 0xf2.

	0x_0	0x_1	0x_2	0x_3	0x_4	0x_5	0x_6	0x_7	0x_8	0x_9	0x_a	0x_b	0x_c	0x_d	0x_e	0x_f
0x0_	0x63	0x7c	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76
0x1_	0xca	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0
0x2_	0xb7	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15
0x3_	0x04	0xc7	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75
0x4_	0x09	0x83	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84
0x5_	0x53	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf
0x6_	0xd0	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8
0x7_	0x51	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2
0x8_	0xcd	0x0c	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73
0x9_	0x60	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb
0xa_	0xe0	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79
0xb_	0xe7	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08
0xc_	0xba	0x78	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a
0xd_	0x70	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e
0xe_	0xe1	0xf8	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf
0xf_	0x8c	0xa1	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16

Figure 3.9: RAES S-box$_0$.

	0x_0	0x_1	0x_2	0x_3	0x_4	0x_5	0x_6	0x_7	0x_8	0x_9	0x_a	0x_b	0x_c	0x_d	0x_e	0x_f
0x0_	0x7c	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76	0xca
0x1_	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0	0xb7
0x2_	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15	0x04
0x3_	0xc7	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75	0x09
0x4_	0x83	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84	0x53
0x5_	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf	0xd0
0x6_	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8	0x51
0x7_	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2	0xcd
0x8_	0x0c	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73	0x60
0x9_	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb	0xe0
0xa_	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79	0xe7
0xb_	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08	0xba
0xc_	0x78	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a	0x70
0xd_	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e	0xe1
0xe_	0xf8	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf	0x8c
0xf_	0xa1	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16	0x63

Figure 3.10: RAES S-box$_1$.

	0x_0	0x_1	0x_2	0x_3	0x_4	0x_5	0x_6	0x_7	0x_8	0x_9	0x_a	0x_b	0x_c	0x_d	0x_e	0x_f
0x0_	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76	0xca	0x82
0x1_	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0	0xb7	0xfd
0x2_	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15	0x04	0xc7
0x3_	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75	0x09	0x83
0x4_	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84	0x53	0xd1
0x5_	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf	0xd0	0xef
0x6_	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8	0x51	0xa3
0x7_	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2	0xcd	0x0c
0x8_	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73	0x60	0x81
0x9_	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb	0xe0	0x32
0xa_	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79	0xe7	0xc8
0xb_	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08	0xba	0x78
0xc_	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a	0x70	0x3e
0xd_	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e	0xe1	0xf8
0xe_	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf	0x8c	0xa1
0xf_	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16	0x63	0x7c

Figure 3.11: RAES S-box$_2$.

Again, looking up 0x04 S-box$_0$ produces the same output. Just one rotation of S-box$_0$ by 2, SBR(2, S-box$_0$) also computes S-box$_2$. In fact, any number of rotations reduce to just one rotation of S-box$_0$:

$$\mathtt{SBR}(r_n,\ \mathtt{SBR}(\cdots,\ \mathtt{SBR}(r_1,\ \mathtt{SBR}(r_0,\ \mathrm{S{-}box}_0))\cdots)) = \mathtt{SBR}((r_0{+}r_1{+}\cdots{+}r_n)\%256,\ \mathrm{S{-}box}_0).$$

Addition here is over the integers, not GF(2^8) and although the mod256 is not necessary, it is still correct. Rotating S-box$_0$ by 256 rotates the S-box all the way around back to its starting position. Considering rotation an adjustment of lookup indicies further simplifies the S-box rotation. This research denotes addition over the integers mod256 with $\boxplus$. As previously noted, looking up 0x02 in S-box$_1$ is also 0x03 in S-box$_0$. Adjusting the lookup index of 0x02 by an increase of 1 has the same effect of rotation. This adjustment is exactly 0x02 $\boxplus$ 0x01 since the lookup indicies are the incoming byte nibbles. Again, looking up 0x02 in S-box$_2$ is also 0x04 in S-box$_0$. Adjusting the lookup index by an increase of 2 produces this same effect. This adjustment is exactly 0x02 $\boxplus$ 0x02. In fact, this property holds for all possible rotation values. Similarly, this manipulation holds over the inverse SBR^{-1} as well.

The Rotational S-box implementations use iterative S-box rotations. ^{i}r represents the rotation value for a particular round as calculated from the expanded rotation key. The key schedule rotates by $^{-1}$r and if necessary (Type 3 and Type 4) $^{-2}$r. Then round 0 of encryption uses S-box$_{[^{-2}\mathrm{r}\boxplus]^{-1}\mathrm{r}}$. Since round 0 does not apply SBR, this value is ^{0}R, the total iterative rotation value of the S-box in round 0. Advancing to round 1, the S-box rotates by ^{1}r, and ^{1}R = ^{0}R $\boxplus$ ^{1}r. Thus S-box lookups in round 1 can follow the form SB(byte $\boxplus$ ^{1}R) using S-box$_0$. Repeating this process out through round 10, ^{10}R = ^{9}R $\boxplus$ ^{10}r = $^{-2}$r $\boxplus$ $^{-1}$r $\boxplus$ ^{1}r $\boxplus \cdots \boxplus$ ^{10}r, and S-box$_0$ lookups follow the form SB(byte $\boxplus$ ^{10}R). Thus, ($\boxplus$ ^{i}R) replacing every instance of SBR creates an equivalent algorithm.

3.3.2 Mapping Rotate to an Operation in GF(2^8).

Figure 3.12 shows an alternative view of the fault propagation model using this additive definition of S-box rotation. With S-box rotation now defined as addition mod 256, leveraging the existing attack relations might now be possible.

$$^{10}\Delta S_0^1 = (^{10}S_0^0 \boxplus {}^{10}R) \oplus (^{10}\bar{S}_0^0 \boxplus {}^{10}R)$$

Assuming $\boxplus$ distributes over addition ($\oplus$) in GF(2^8), then $^{10}\Delta S_0^1 = {}^{10}R \boxplus (^{10}S_0^0 \oplus {}^{10}\bar{S}_0^0) = {}^{10}R \boxplus 2b$. Similarly, $^{10}\Delta S_4^1 = (^{10}S_4^0 \boxplus {}^{10}R) \oplus (^{10}\bar{S}_4^0 \boxplus {}^{10}R) = {}^{10}R \boxplus (^{10}S_4^0 \oplus {}^{10}\bar{S}_4^0) = {}^{10}R \boxplus b$. Now assuming $\boxplus$ distributes over multiplication in GF(2^8), then $^{10}\Delta S_0^1 = 2(^{10}R \boxplus b)$ and $^{10}\Delta S_4^1 = (^{10}R \boxplus b)$. Letting $(^{10}R \boxplus b) = v$, the final result is $^{10}\Delta S_0^1 = 2v$, $^{10}\Delta S_4^1 = v$. This result restores the original $^{10}\Delta S^1$ column relations regardless of ^{10}R, requiring no new attack analysis to match the reductions established in the existing attack by using S-box$_0$. However, this conclusion requires proving the assumptions that $\boxplus$ distributes over both addition and multiplication in GF(2^8). Testing these assumptions with discrete values shows $\boxplus$ does not distribute over either addition or multiplication in GF(2^8), so the initial fault propagation remains unexploitable.

$$1 \boxplus (2 \times 3) = 1 \boxplus (6) = 7 \neq 12 = 3 \times 4 = (1 \boxplus 2) \times (1 \boxplus 3)$$

$$1 \boxplus (2 \oplus 3) = 1 \boxplus (1) = 2 \neq 7 = 3 \oplus 4 = (1 \boxplus 2) \oplus (1 \boxplus 3)$$

3.3.3 Alternate Attack Analysis on Standard AES.

Since analysis of SBR as $\boxplus$ is not sufficient to extend the attack, an analysis of the existing attack described in 3.2.2 with a slightly different way of thinking follows. This analysis removes the need for full inspection of all S-box$_i$ properties. Figure 3.13 shows $^{10}\Delta S$ of the standard AES attack for reference.

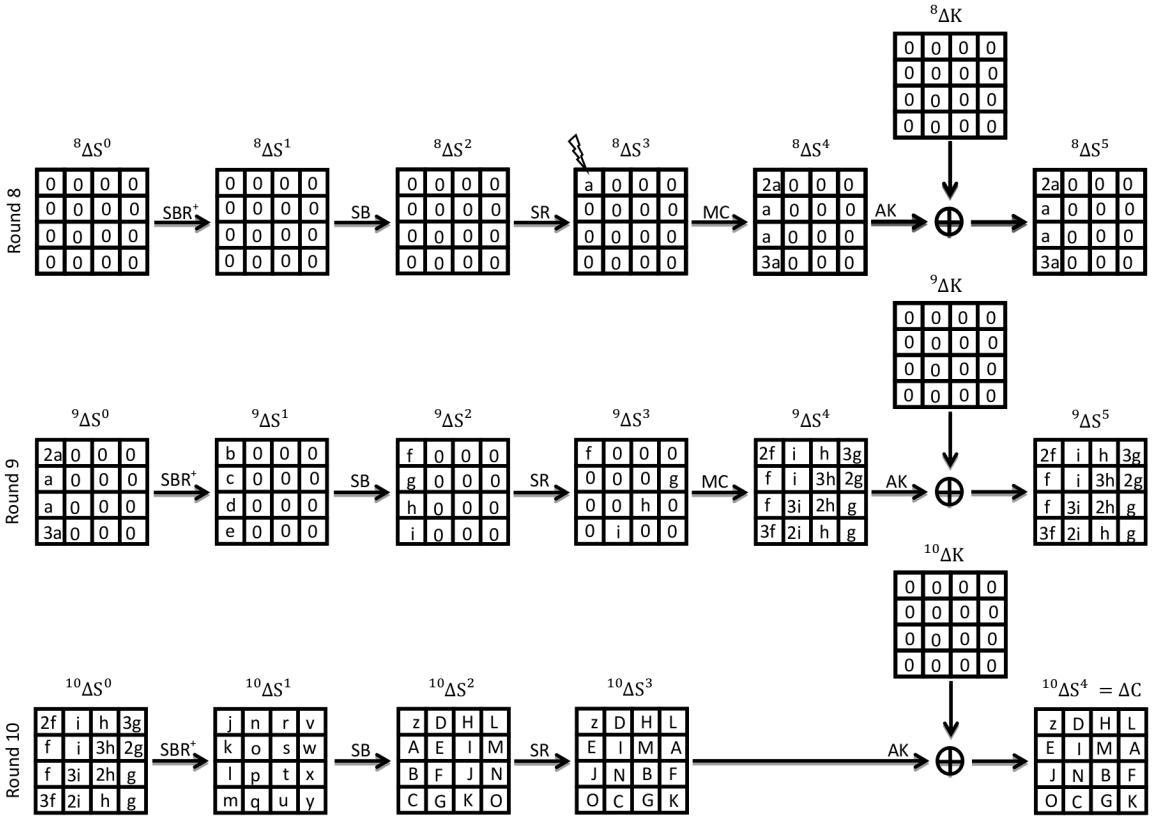

Figure 3.12: XOR of Correct and Faulty RAES-128 Encryption, Rounds 8-10.

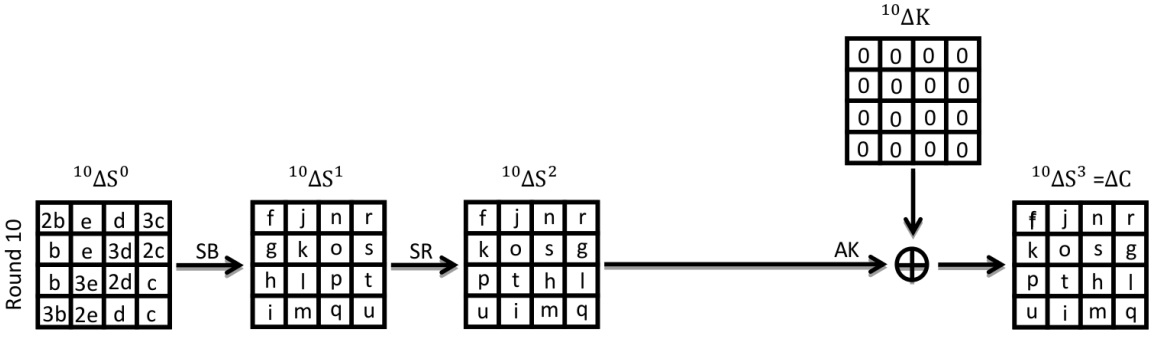

Figure 3.13: XOR of Correct and Faulty AES-128 Encryption, Round 10.

Examining ${}^{10}\Delta S^1$ which is known, 127 ${}^{10}\Delta S^0_0$ values are possible out of 255, this set of values is $\{{}^{10}\Delta S^0_0\}$. Thus, the likelihood of a random value in {1,255} being in $\{{}^{10}\Delta S^0_0\}$ is $\frac{127}{255}$. This likelihood is also true for ${}^{10}\Delta S^0_4$, ${}^{10}\Delta S^0_8$, and ${}^{10}\Delta S^0_{12}$. So, for a given $2b \in \{{}^{10}\Delta S^1_0\}$, the probability of $2b \in \{{}^{10}\Delta S^1_0\}$, $b \in \{{}^{10}\Delta S^1_4\}$, $b \in \{{}^{10}\Delta S^1_8\}$, and $3b \in \{{}^{10}\Delta S^1_{12}\}$ is $1 \times \frac{127}{255} \times \frac{127}{255} \times \frac{127}{255}$. Since there are 127 $2b \in \{{}^{10}\Delta S^0_0\}$, the number of 2b expected to satisfy the above relation and be in each set is $127 \times (\frac{127}{255})^3$.

The 256 possible ${}^{10}K_0$ key byte values step back to 127 2b values. So, each valid 2b value averages to an expected $\frac{256}{127}$ keyspace for that byte. Thus, the average keyspace for a valid 2b, b, b, 3b column is $(\frac{256}{127})^4$.

Combining the number of valid 2b columns with the keyspace for each valid 2b column results in the total average keyspace of a column:

$$(127 \times (\frac{127}{255})^3) \times (\frac{256}{127})^4 = (\frac{127}{127})^4 \times \frac{256^4}{255^3} = \frac{256^4}{255^3}.$$

The expected keyspace per valid 2b and the expected number of valid 2b have no influence on this reduction. Since columns are independent, applying the same relation to each of the four columns creates a total reduction of: $(\frac{256^4}{255^3})^4 \approx 2^{32.0677}$.

3.3.4 General Extension.

The above reworking of the existing attack on standard AES revealed the average reduction across the S-box is independent of the number of resulting valid 2b or the expected keyspace per valid 2b because these values cancel. Thus, no analysis needs to be done around the Rotational S-box. The averages smooth out all inconsistencies and discrete numbers. Therefore, regardless of the S-box used, the average resulting keyspace is about $2^{32.0677}$. Since, in round 10, ${}^{10}R$ can be any value in {0, 1, ..., 255}, 256 of these $2^{32.0677}$ keyspaces exist. The total keyspace remaining after stepping back to ${}^{10}\Delta S^0$ is approximately $2^{32.0677} \times 2^8 = 2^{40.0677}$.

The existing attack reduces in round 9 by stepping back each possible remaining ${}^{10}K$. This extension has an increased work factor based on the larger $2^{40.0677}$ remaining keyspace.

Additionally, stepping back ${}^{10}\mathrm{K}$ to ${}^{9}\mathrm{K}$ requires use of *SubWord*. However, because the expanded encryption key uses a rotated S-box, each $2^{40.0677}$ possible keys steps back 2^8 ways for each possible S-box rotation, further increasing the work factor to $2^{48.0677}$. Since extending an attack is the goal of this research, and the round 10 analysis contains the essence of this DFA on the State attack while maintaining a much higher power to speed ratio, this research only extends the round 10 portion of this attack.

Access to a second cipher pair allows an independent reduction to an alternate reduced keyspace of approximately $2^{40.0677}$. Intersecting these keyspaces creates the remaining valid keyspace. Assuming the incorrect keys in each reduced keyspace are random, $2^{40.0677} \times \frac{2^{40.0677}}{2^{128}} \approx 2^{-47.8646}$ keys remain. Thus, only the valid K_{10} key should remain. Recovery of the encryption key K_0 still requires reversal of the key schedule.

3.3.5 Reversing the Key Schedule.

The previous section provides the theoretical keyspace reduction power of the attack extension regardless of Rotational S-box Type implementation. The analysis shows that full recovery of ${}^{10}\mathrm{K}$ is possible. With standard AES, recovery of ${}^{10}\mathrm{K}$ concludes the attack because the key schedule is fully reversible. Following is an analysis of reversing the key schedule for all RAES Types.

Knowing the S-box in standard AES makes reversal of the key schedule possible. The RAES encryption key schedule uses $\text{S-box}_{{}^{-1}\mathrm{R}}$, which is unknown. However, this S-box is one of only 256 possibilities, meaning there are 256 potential encryption keys. Reversing to each of these is trivial.

- **Key Schedule 1.** For key schedule 1, ${}^{-1}\mathrm{R} = {}^{-1}\mathrm{r}$ is the XOR of all encryption key bytes. Reversal of the expanded encryption key with a particular ${}^{-1}\mathrm{r}$ reveals the first 16 bytes, ${}^{0}\mathrm{K}$, the encryption key. Checking that the XOR of these bytes matches the ${}^{-1}\mathrm{r}$ value used to reverse each particular expanded key reduces the possible ${}^{-1}\mathrm{r}$

values. Since one ${}^{-1}r$ is valid out of 256, and 256 options are checked, ${}^{-1}r$ should reduce to one possibility.

- **Key Schedule 2.** In key schedule 2, ${}^{-2}r$ is the XOR of all encryption key bytes ${}^{0}K$ and ${}^{0}K = {}^{0}RK$. Expanding this out to the expanded rotation key allows computation of ${}^{-1}r$. Checking this ${}^{-1}R$ against the value used to reverse the encryption key, like in the key schedule 1 analysis above, should reduce to one possibility. Thus the same reduction power is possible, requiring an extra step of key expansion.

If more than one ${}^{0}K$ remain after this ${}^{-1}R$ check, two more reduction checks are possible. First, rebuilding the expanded rotation key and calculating is associated ${}^{10}R$ value enables a check that this matches the ${}^{10}R$ used to create ${}^{10}K$. If multiple possibilities still remain, checking ${}^{9}\Delta S^{0}$ relations provide a final reduction. Using round 9 relations is not an unreasonable work factor like a full round 9 reduction because this instance only steps back one ${}^{10}K$ and few possible ${}^{9}R$ should remain after the two prior reductions.

3.4 Theoretical Attack Summary

Overall, this attack extension is slightly less powerful, and less flexible to Eve's resources and constraints. With only one cipher pair, the extended attack is much less powerful, effectively only able to reduce the keyspace to $2^{40.0677}$ with a reasonable work factor, where the existing attack could reduce the keyspace to 2^{8} with reasonable computational effort. However, if Eve has access to, or the capability to create two or more cipher pairs, the attacks effectively have the same expected reduction power of full keyspace recovery.

IV. Methodology

This chapter details the experimental methodology for verifying the theoretical results of Chapter 3. First, Section 4.1 discusses the approach and expected results. Sections 4.2 through 4.7 define the experimental environment including boundaries, workload and metrics. Finally, Sections 4.8 and 4.9 explain the experimental implementation.

4.1 Problem Definition

4.1.1 Goals and Hypothesis.

The goal of this research is to verify the theoretical analysis in Section 3.3.4 and determine the actual attack power of DFA on the State on standard AES and research driven rotational S-box AES designs. This analysis expects DFA on the State of standard AES using one cipher pair reduction to produce an average keyspace of approximately $2^{32.0677}$, while expecting all RAES Types to yield $2^{40.0677}$. This research expands the overall security analysis of a dynamic S-box AES to assess its practicality and usefulness compared to standard AES and validates the existing theoretical DFA work on AES-128 [28].

4.1.2 Approach.

This research attacks both the standard AES-128 implementation with the existing attack as described in Section 3.2.4 as a baseline and the four Rotational S-box AES-128 implementations as defined in Section 3.2.2 with the extended attack as described in Section 3.3.4. Specifically, focusing on ^{10}K keyspace reductions and reductions of valid ^{10}R allows verification of the theoretical analysis performed, and enables the analysis of discrete reductions, not just expected averages. This discrete data sheds further light on the underlying mechanics at work.

4.2 System Boundaries

The System Under Test (SUT) is the Cryptanalysis System. Because the focus is several cryptanalysis techniques on different AES algorithms with the goal of key recovery, the Component Under Test (CUT) is the Solver. The Solver solves the constructed DFE by checking that the $^{10}\Delta S^0$ relations hold. Other components of the system are the cryptanalysis attack, the AES algorithm, the S-box, and the encryption environment. This study limits the scope of these. The encryption algorithm is scoped to only AES-128 variants, specifically AES-128 and RAES-128 Types 1-4. Additionally, DFA attacks on the State are the only cryptanalysis attacks considered. Lastly, the encryption environment is restricted to software, rather than hardware, to more easily facilitate fault injection. Figure 4.1 depicts the system boundaries.

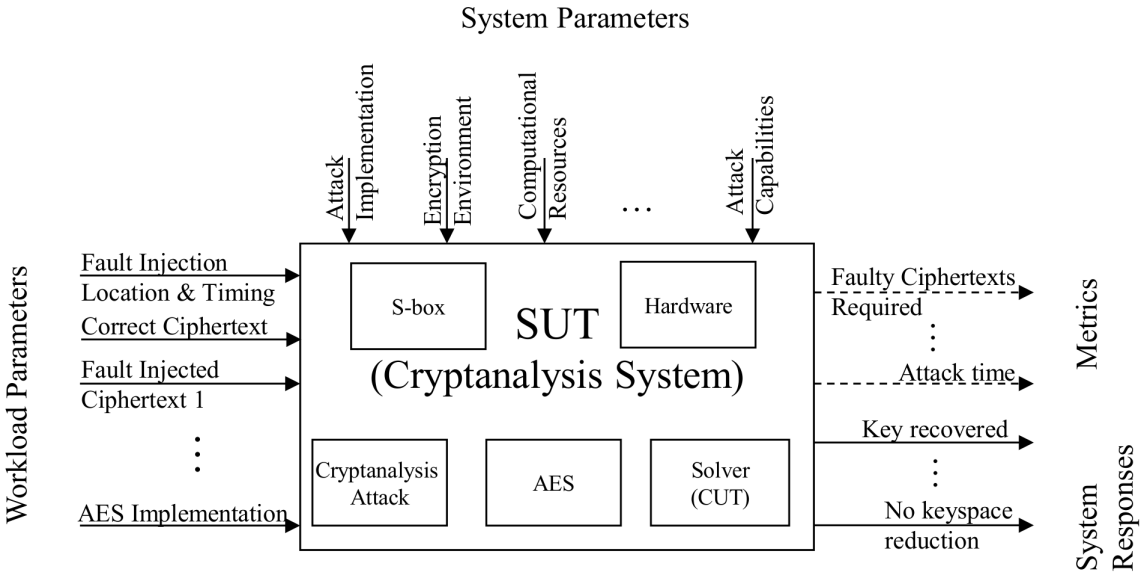

Figure 4.1: System Boundaries.

4.3 System Services

The Cryptanalysis System provides a key recovery service. The possible outcomes are: (1) full recovery of the encryption key, (2) reducing the keyspace to an unsecure size such that an exhaustive search is feasible in one hour on a personal workstation using an Intel i7 CPU with 8 GB of memory, (3) reducing the possible keyspace but exhaustive search remains computationally infeasible in one hour on a personal workstation using an Intel i7 CPU with 8 GB of memory, or (4) discovering no information and the keyspace remains unaffected. This study focuses on outcome (1) as this is the only theoretical result of the attacks considered.

4.4 Workload

The workload submitted to the system is a correct ciphertext and several corresponding fault injected ciphertexts. These pairs are what the DFA specifically exploits. Workload parameters also include the fault injection timing and location data and the AES implementation as a successful DFA on the State attack requires this knowledge. This study limits the fault injection timing and location to ${}^{8}S_{0}^{2}$. The plaintext and key sent to the encryption algorithm should not change attack complexity, thus they are not workload parameters, but instead randomly generated.

4.5 Performance Metrics

Attack efficacy dictates system performance. The number of faulty ciphertexts required for full key recovery most significantly captures efficacy. Reductions at each stage of the solving process more precisely capture this performance and allows comparison to the theoretical power calculated. Lastly, timing metrics roughly gauge work factors. However, since computation time is not the main focus and not of critical importance, minimal measures control the testing environment. Overall, these metrics provide a total

picture of DFA attack power on standard AES-128 and the four Rotational S-box AES-128 implementations.

4.6 System Parameters

The system parameters are the attack implementation, computational resources, access to the encryption machine, encryption environment, and any other available information that might help the attack. Because this study limits its scope to DFA on the State attacks and each depends on fault placement and AES implementation, these workload parameters directly dictate the attack implementation. Computational resources are important because they alter the time required to perform the attacks and put limits on the amount of computation possible through memory limitations. If this system used a fully realized attack, access to the encryption machine would be an important factor in choosing the physical attack vector used to induce the faults. Because DFA attacks rely on inducing faults through physical phenomena, the encryption hardware used affects the practicality of an attack. The examined attacks require introduction of faults to specific positions of the algorithm at specific times, and exploitable hardware is necessary. However, because the encryption environment here is in software which also simulates faults (inducing them intentionally through code as part of the encryption algorithm rather than through actual physical processes on the encryption hardware) an exploitable encryption environment is not a concern for this research. Any prior knowledge of the encryption system beyond the encryption algorithm provides information that may reduce the possible keyspace.

4.7 Factors

The only factor of this study is attack implementation which has five levels relating the standard AES and the four Rotational S-box AES Type implementations. This study fixes all other parameters to one value. To reiterate these values, Figure 4.2 displays each significant parameter.

4.8 Evaluation Technique

Evaluation occurs in multiple reduction steps. Simulations artificially inject faults at arbitrary positions and times of the encryption algorithm without the overhead of specialized hardware that true implementation and measurement would require. Additionally, simulations produce actual data which provides a discrete and quantified set of data to analyze, something missing in a purely analytic technique. This experimentation produced a simulated environment, (R)AES-DFA v1.0, in Python 2.7.3. Appendix A covers the steps taken to validate this software's functionality.

Factors	Levels
Encryption Algorithm	AES, RAES Type 1, RAES Type 2, RAES Type 3, RAES Type 4
Key Length	128 bit
Plaintext Encrypted	Random bits
Encryption Key	Random bits
Attack Type	DFA on the State
Fault Induction Location	Round 8, Byte 0 before MixColumns
Encryption Environment	Software
Computational Resources	2011 2.8 Ghz i7 iMac with 8GB 1333 Mhz DDR3 Ram

Figure 4.2: Factors and Levels.

The simulation runs a given variant of AES for each plaintext and key pair given; first running correctly and then injecting a random fault at the specified position and timing

within the algorithm. The solver reduces the keyspace as much as possible from this cipher pair as described in Section 3.3 using round 10 reductions. Reductions continues with additional faulty ciphertexts until full key recovery. Validation that the attack was successful occurs by checking the recovered key and plaintext values with the actual input data.

The attack implementation treats each potential (^{10}K, ^{10}R) as part of the reduced keyspace. This pairing counts two identical ^{10}K recovered with different ^{10}R as separate valid ^{10}K keys. Pilot studies of the attack implementation revealed that round 10 reductions could only ever reduce the keyspace to 2 regardless of the number of cipher pairs used. Investigation revealed one ^{10}K with two ^{10}R values always made this keyspace of two, not two ^{10}K each with one ^{10}R. Several stages capture all keyspace reductions. First, cipher pairs reduce the keyspace to two. Then key schedule reductions occur to reduce the ^{10}K keyspace to one and correctly reverse to the encryption key ^{0}K. Notable data for analysis captured at each stage includes the size of the remaining keyspace, the valid ^{10}R values, and computation time. Capturing additional verification and replication data make this experimentation fully repeatable. This data includes the plaintext and key used, the resulting ciphertext, and the XOR fault used each time for a new faulty ciphertext. The captured information provides a robust data set from which future work can replicate this work to validate, correct, or improve upon the algorithms, data collected, and following analysis.

4.9 Experimental Design

With only one factor, this experiment is trivially a full-factorial experimental design of the simulated attack described in Section 4.8 requiring the following number of iterations: (# successful attacks developed + # existing comparable attacks)×(#repetitions). Setting the number of repetitions to 2,500 results in $(4 + 1) \times 2,500 = 12,500$ iterations. Although encryption is deterministic, this experiment only uses one class of keys and plaintexts, that

is, completely random. As such, each repetition uses a random key and plaintext, thus changing the resultant ciphertext. The simulation does nothing to standardize the faults injected across attack implementations for each repetition. The large number of repetitions follow from the extreme magnitude of the population and the small time cost of additional operations discovered in pilot studies. A minimum of (2^{128} plaintexts) $\times$ (2^{128} keys) $\times$ (256 fault 1 values) $\times$ (255 fault 2 values) $\approx 2^{272}$ attack vectors exist for any given attack implementation. Thus, even a sample of 2500 is only roughly $2^{-254}\%$ of the possible attack vectors. Analysis uses a 95% confidence level.

4.10 Methodology Summary

The goal of this study is to determine the security of a dynamic S-box AES design by attacking AES-128 and RAES-128 implementations with simulated DFA. The SUT is the Cryptanalysis System, which reduces the entropy of the encryption key. The CUT is the Solver of DFE and recovers the encryption key. The factor tested is attack implementation. Simulated attacks provide more meaningful data for analysis while remaining cheap and easy to implement. This data allows verification of the theoretical analysis in Section 3.3.

V. Analysis of Experimental Attack Results

This chapter analyzes the experimental data captured as described in Section 4.8. The focus of this chapter is validation of the theoretical average keyspace reduction after 1 pair of ciphertexts leveraging round 10 reductions. This chapter also analyzes reductions after multiple pairs along with a few other interesting discussions and observations about the data. The data captured for each attack includes: the AES implementation; total runtime required; number of faulty ciphertexts required; the plaintext encrypted; the encryption key; the recovered encryption key; runtimes required for each faulty ciphertext reduction; keyspace remaining after each faulty ciphertext reduction; the fault value that when XOR'ed with ${}^{8}S_{0}^{2}$ is the resulting faulty value used moving forward; the resulting faulty ciphertext; the average columnspace after each faulty ciphertext reduction; and, if a RAES implementation, the number and values of ${}^{10}R$ still valid after each faulty ciphertext reduction.

5.1 Existing Attack

Prior research establishes a round 10 reduced keyspace of 2^{32}, analysis in Section 3.3.3 establishes a slightly higher $4,501,500,262 \approx 2^{32.0677}$ average. Figure 5.1 displays the frequency of reduced keyspaces after one pair of faulty ciphertexts with the expected and observed averages marked. This data greatly departs from a normal distribution with a very long and non-continuous tail. The observed mean is $5,404,337,163 \approx 2^{32.3314}$, $902,756,005 \approx 2^{29.7497}$ larger than the theoretical average. However, the $\log_2$ of the observed and theoretical means only differ by 0.2637. Figure 5.2 represents this same data in a boxplot. This highlights the skew of the data.

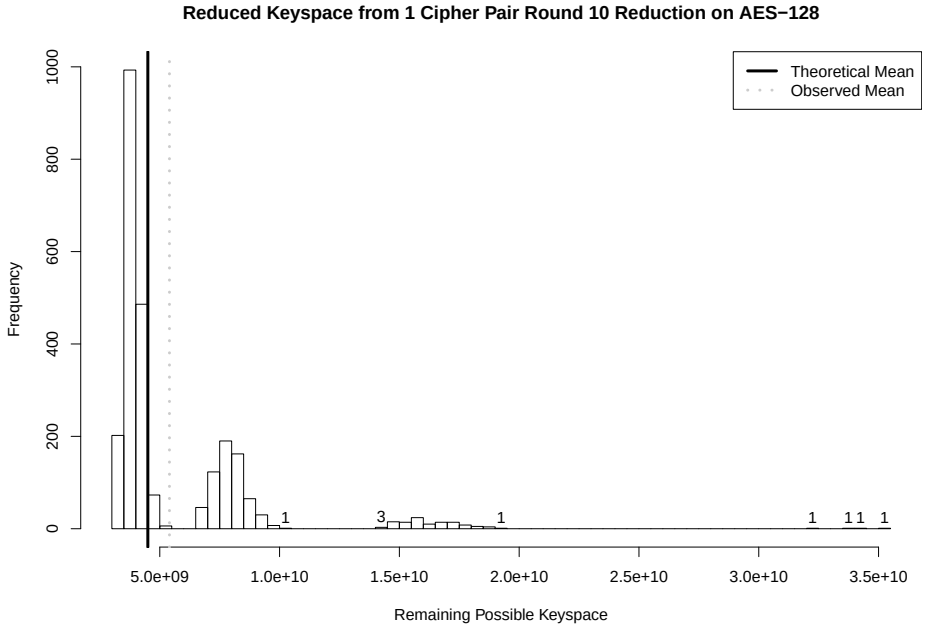

Figure 5.1: Histogram of AES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

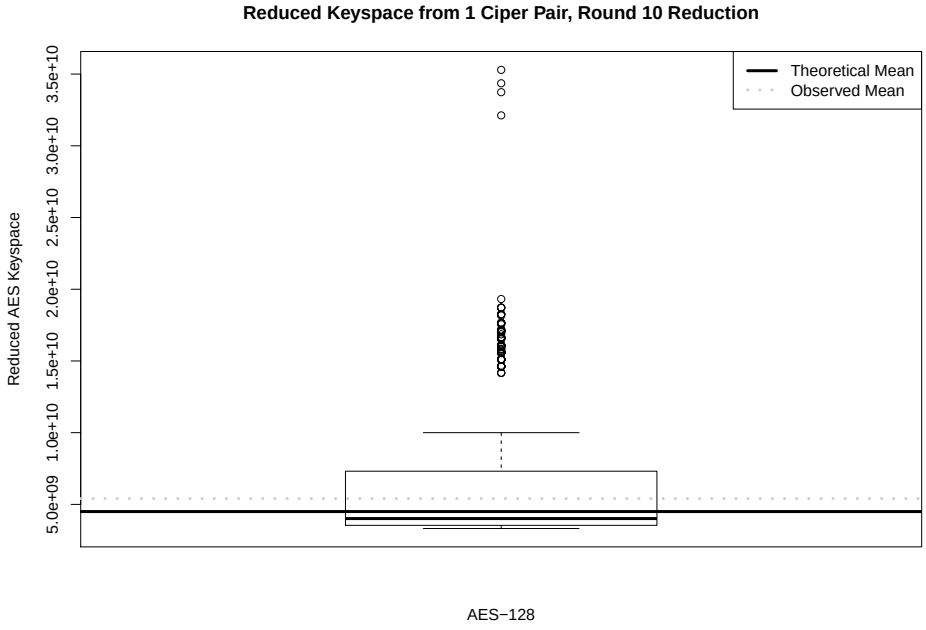

Figure 5.2: Boxplot of AES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

Examining the quartile values in Figure 5.3, the first two quartiles occur in a range of $693,043,200$, while the third quartile spans a range of $3,303,014,399$ and the last $27,981,250,661$. This analysis explores this extreme skew in density later. Figure 5.4 displays the data as $\log_2$ transformed which helps to minimize this skew, although a bottom heavy density remains apparent. Figure 5.5 displays this same $\log_2$ transformation applied to the histogram. An additional line to mark the observed $\log_2$ mean is also added.

0%	25%	50%	75%	100%
$3,317,776,000$	$3,538,944,000$	$4,010,803,200$	$7,313,817,592$	$35,295,068,253$
$2^{31.6275}$	$2^{31.7206}$	$2^{31.9012}$	$2^{32.7679}$	$2^{35.0387}$

Figure 5.3: Quartiles of AES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

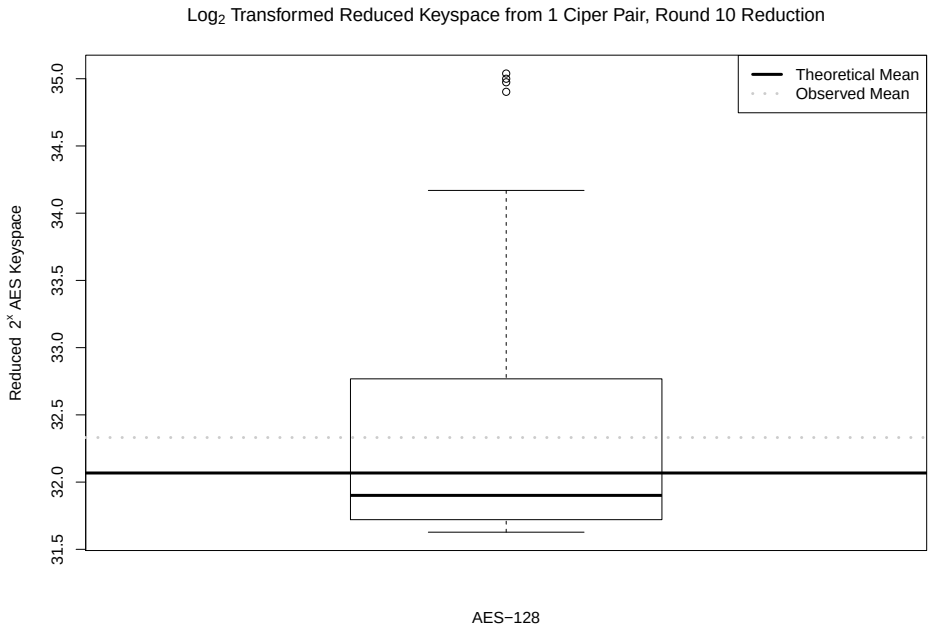

Figure 5.4: Boxplot of Log_2 Transformed AES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

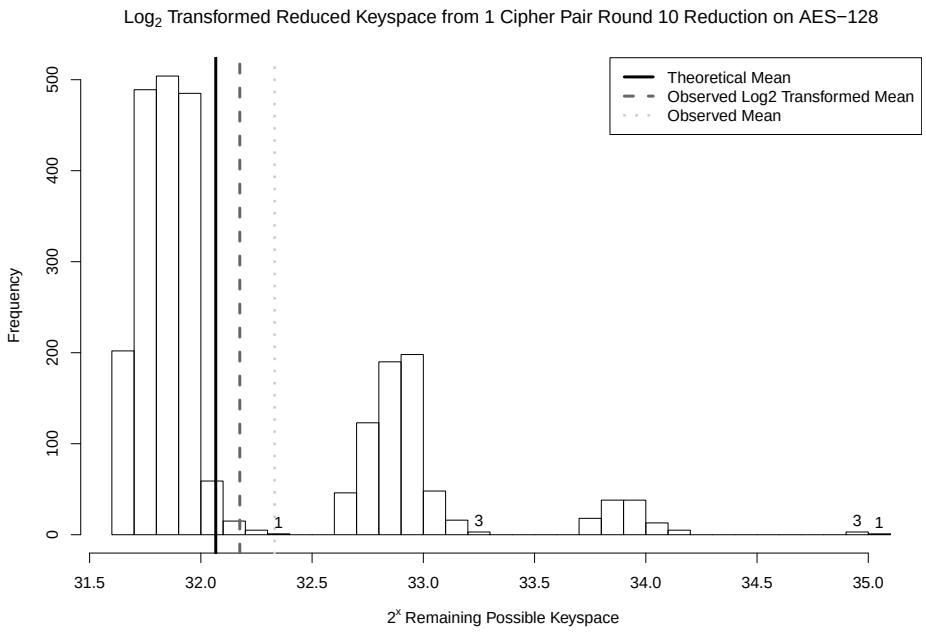

Figure 5.5: Histogram of Log_2 Transformed AES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

This transformation creates a more interesting representation of the data. The data manifests in several high density spikes approximately around 31.8, 32.9, 34.9 and 35.9. Each grouping appears to be close to a normal distribution, and lessening in magnitude with each additional power of 2. This observed data makes sense as the most likely theoretical average is $(127 \times (\frac{127}{255})^3 \times 2^4)^4 = 3,970,610,628 \approx 2^{31.8867}$, and each time another key byte has 4 possibilities rather than 2, an increase by one power of 2 occurs. As these 4 possibility key bytes are unlikely with probability (1/127), the diminishing frequency fits. This analysis explains the drastic skew in density and the inter-group distributions. The close to normal distributions around these groupings also require examination. Part of the reduction power is the number of b relations expected to hold per column. The average is $127 \times (\frac{127}{255})^3 \approx 15.6889$. However, discrete values cannot be decimal creating slightly higher and lower values. If each column has 15 valid relations, $2^{31.8867}$ becomes

$(15 \times 2^4)^4 = 3,317,760,000 \approx 2^{31.6275}$, and at 16 becomes $(16 \times 2^4)^4 = 4,294,967,296 = 2^{32}$. Combinations of 15 and 16 valid relation columns fall between these values and closer to the average (e.g., $(15{\times}2^4)^2{\times}(16{\times}2^4)^2 = 3,774,873,600 \approx 2^{31.8137}$). The number of valid b relations does not have nearly the same magnitude of effect on the expected remaining keyspace as the number of valid byte keys. This smaller effect explains the intra-group distributions.

Figure 5.6 shows the remaining keyspace after two faulty ciphertexts. Two attacks still had 4 possible ^{10}K values and a third had 16 possible ^{10}K values. The attacks with four keys remaining manifest either in one byte with four possible values and the 15 other bytes fixed or two bytes with two possible values and the 14 other bytes fixed. Similarly, the attack with sixteen possible keys remaining manifests in one of several possibilities: two bytes with 4 possible values and the other 14 fixed; one byte with 4 values, two with 2 and the other 13 fixed; or four bytes with 2 values and the other 12 fixed. However, since no attacks yielded 2 possible ^{10}K values after two faulty ciphertexts, only bytes with four possible values likely create the overlap of these three keyspaces. The observed remaining mean keyspace is $\frac{2497+2+2+16}{2500} = 1 + .0084$, significantly larger than the expected $1 + 2^{-63.8646}$. The analysis of multiple faulty ciphertexts in Section 3.3.4 assumed the non-valid keyspace bytes were random because claiming an underlying relationship requires substantial data, analysis and understanding. This attack implementation did not collect the actual potential keyspace values at intermediate reduction stages. Further analysis requires at least this data, so explaining the unexpected non-valid keyspace bytes after two faulty ciphertexts is not possible. However, these three attacks with multiple possible ^{10}K values remaining suggest the non-valid keyspace bytes are not random.

The quartiles show that the median value is well below the theoretical average, however the extreme upper half values skew the mean above the median. Overall, the theoretical averages appear to underestimate the true average reduction by not properly

accounting for either the likelihood of the one 4 key byte associated b value being valid, or the associated drastic increase in remaining keyspace. Although still an underestimation, the theoretical average appears to more closely estimate the average $\log_2$ remaining keyspace. Additionally, the reduction power between two cipher pairs does not provide enough information to adequately predict the number of cipher pairs required on average. Reduced keyspaces associated with faulty ciphertexts appear non-random and to have some increased association.

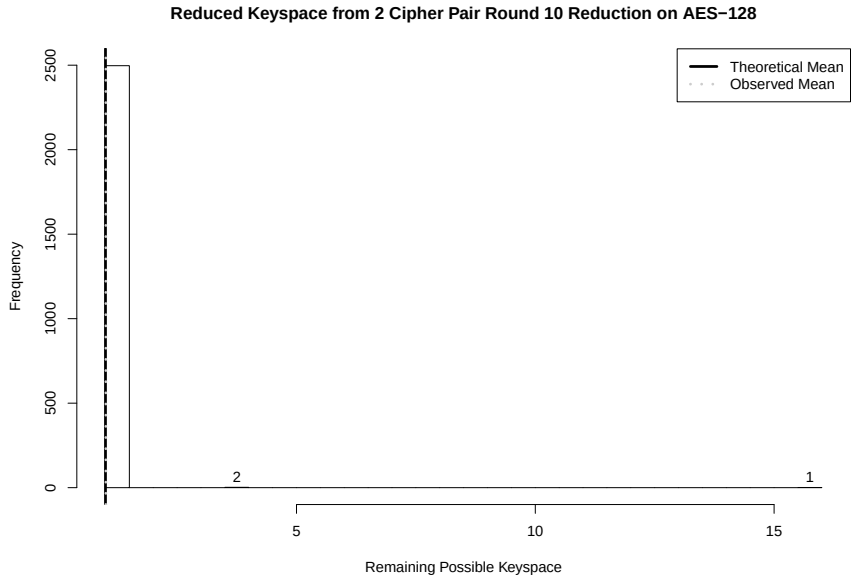

Figure 5.6: Histogram of AES-128 ^{10}K Keyspace, 2 Faulty Ciphertext Round 10 Reduction.

As mentioned in Section 4.5, because this experimentation used no explicitly controlled testing environment, rigorous analysis of timing data is not valid. However, to provide a context to the work factor required for the attack, a histogram of attack times follows in Figure 5.7. The average attack time is 0.1747 seconds.

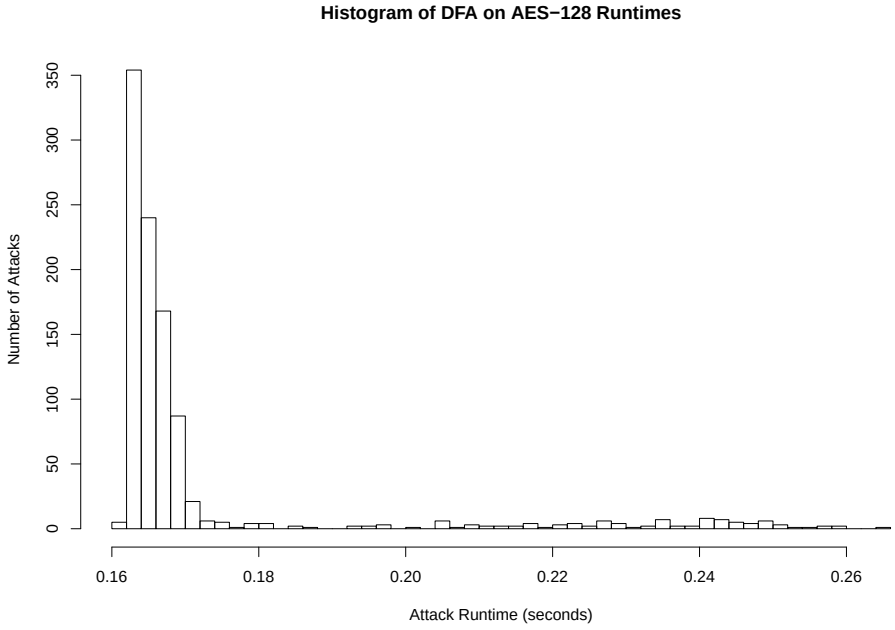

Figure 5.7: Histogram of AES-128 DFA Runtime.

5.2 Attack Extension

Section 3.3.4 established an average theoretical remaining keyspace of $2^{40.0677}$ after 1 cipher pair round 10 reduction on RAES implementations. This section analyzes the observed experimental data in an effort to validate this expected theoretical analysis. First, Figure 5.8 shows the $\log_2$ transformed boxplot of this reduction across each of the four RAES-128 Type implementations. Theoretical analysis resulted in the same reduction regardless of Type, these observed data agree. Formally checking this conclusion with the Tukey multiple comparisons of means test in Figure 5.9 confirms that there is no statistical difference in the average reduction power regardless of RAES implementation. The 0 RAES Implementation Type is the combination of all Types 1-4. Thus, analysis moving forward is performed on all RAES implementations treated as one population.

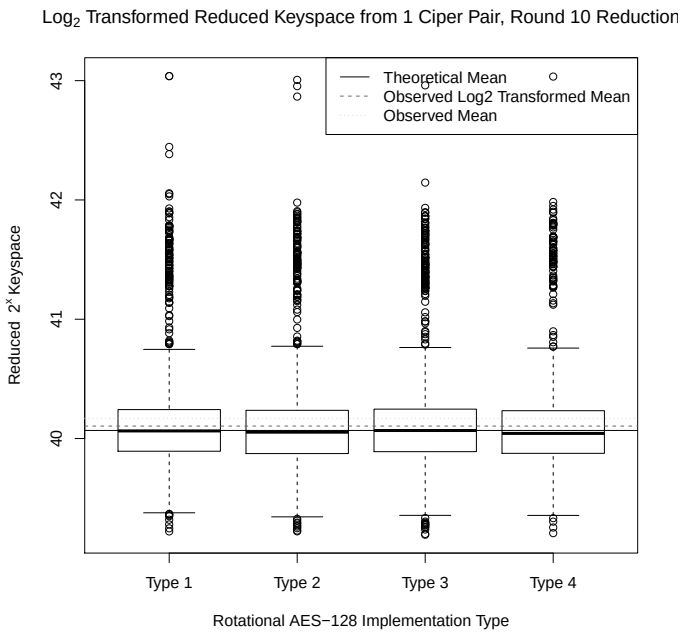

Figure 5.8: Boxplots of Log_2 Transformed RAES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

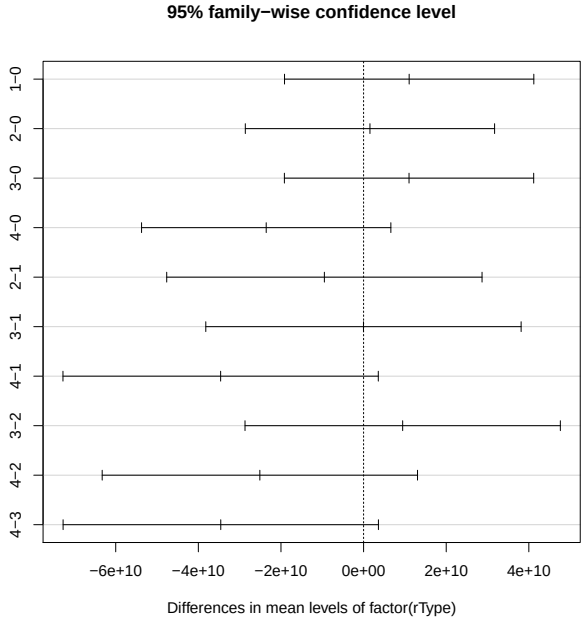

Figure 5.9: Tukey Multiple Comparisons of Means Test of RAES-128 Types, 1 Faulty Ciphertext Round 10 Reduction.

Figure 5.10 is a histogram of the 1 cipher pair, round 10 reduction on all Rotational S-box AES implementations. This reduction, although much closer to normal than the existing attack's data still maintains the extreme skew in density with a long right tail and high outliers. The expected theoretical mean is $1,152,384,067,070 \approx 2^{40.0677}$. The observed mean is $1,236,479,882,848 \approx 2^{40.1693}$, $84,095,815,778$ larger than expected. However, the difference of the $\log_2$ of the means is only 0.1016.

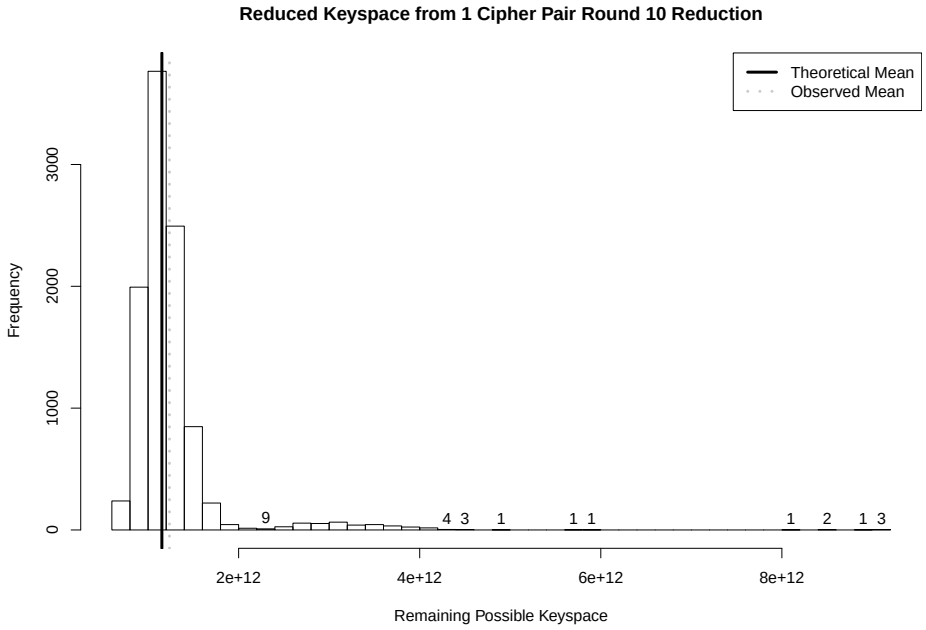

Figure 5.10: Histogram of RAES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

Examining the quartiles in Figure 5.11, like in the existing attack on AES, the median is below the expected average. However, now the third quartile contributes significantly less to the skew. Instead, the immense magnitude of the values in the last quartile produce this skew. Due to the extreme scaling of keyspaces, like before in the existing attack analysis, using a $\log_2$ transformation helps make this more meaningful data. Figure 5.12

displays the transformed histogram. The result is a normal distribution centered near the theoretical average. Unfortunately that does not encompass all the data; another near normal distribution centered near 41.5 and several extreme right outliers also are present.

0%	25%	50%	75%	100%
$628,701,657,333$	$1,014,220,360,081$	$1,142,741,075,045$	$1,298,812,893,235$	$9,033,458,910,972$
$2^{39.1935}$	$2^{39.8835}$	$2^{40.0556}$	$2^{40.2403}$	$2^{43.0384}$

Figure 5.11: Quartiles of RAES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

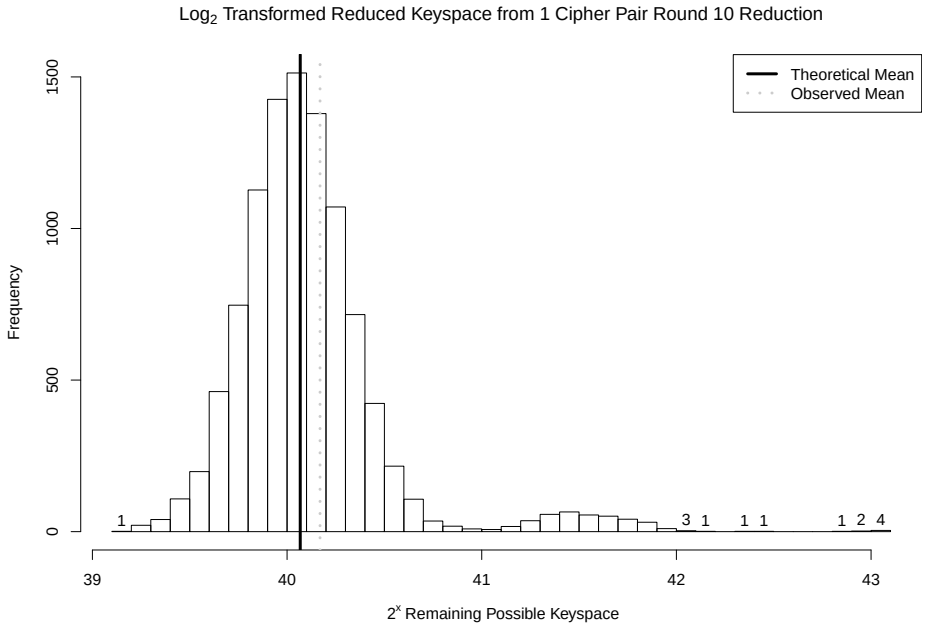

Figure 5.12: Histogram of Log_2 Transformed RAES-128 ^{10}K Keyspace, 1 Faulty Ciphertext Round 10 Reduction.

The normal distribution hot spot near 2^{40} fits with the theoretical analysis. The secondary distribution around $2^{41.5}$ and the lack of tertiary and quaternary reduction pockets as seen in the existing analysis need further investigation. The existing attack had those hot spots from the unlikely 'high' number of key byte stepbacks (i.e., the 4 key byte b values). Now for an attack to deviate from the rest, each of the 256 S-boxes used need to hit the 'high' key byte stepbacks. This property has an averaging effect making the values seen more consistent: (total reduced keyspace) = ((S-box$_0$ reduced keyspace) + (S-box$_1$ reduced keyspace) + ... + (S-box$_{255}$ reduced keyspace)) = (Average S-box$_i$ reduced keyspace) $\times 2^8$. As discussed in the analysis of the existing attack, the number of valid b relations has a much smaller effect on keyspace size than the number of valid byte keys per valid b. Examining the maximum value of $2^{43.0384}$, if every S-box had the same stepback properties as S-box$_0$, only each of the 256 S-box stepbacks resulting in about a 2^{35} reduced keyspace would achieve such a large remaining keyspace ($2^{35} \times 2^8 = 2^{43}$). Since Figure 5.5 holds only four attacks with keyspaces near 2^{35}, seeing this 256 times for a single discrete attack would be extremely unlikely. With reasonable certainty, the other S-box's do not have such uniform stepback properties. This variability may be due to S-box construction using multiplicative inverses, but rotating the S-box removes this property. Thus a histogram like Figure 5.5 for an alternate S-box would likely have a much greater variance with high density hot spots spaced more sporadically and more extreme outlier values.

As mentioned in Section 4.8, the keyspace calculation depends on ^{10}R values, thus one ^{10}K and two ^{10}R create an observed keyspace of two. Pilot studies revealed this duplicity is exactly what happens. These studies revealed that, regardless the number of faulty ciphertexts used, Round 10 column reductions maximumly reduce the keyspace to two ^{10}R and one ^{10}K. The experimental data shows these two possible rotation 10 values are always 128 apart. A rotation of 128 is a special case because a half rotation is its own inverse, b $\boxplus$ 128 $\boxplus$ 128 = b $\boxplus$ 256 = b. This addition of 128 `mod` 256 flips the leftmost

bit which is exactly an XOR of 128. Thus, SBR(128) = $\boxplus$ 128 = $\oplus$ 128. This one case of R = 128 is a special case of the general extension tried in Section 3.3.2. Analyzing this property in the additive SBR fault propagation model outlined in Figure 3.12 from ${}^{10}\Delta S^0$ to ${}^{10}\Delta S^1$, if ${}^{10}R = \theta$ is valid, then ${}^{10}R = \theta \boxplus 128$ is also valid.

$$\mathtt{SBR}({}^{10}S_0^0, \theta) \oplus \mathtt{SBR}({}^{10}\bar{S}_0^0, \theta) = {}^{10}S_0^1 \oplus {}^{10}\bar{S}_0^1 = {}^{10}\Delta S_0^1$$

$$\begin{aligned}
\mathtt{SBR}({}^{10}S_0^0, \theta \boxplus 128) \oplus \mathtt{SBR}({}^{10}\bar{S}_0^0, \theta \boxplus 128) &= \mathtt{SBR}(\mathtt{SBR}({}^{10}S_0^0, \theta), 128) \oplus \mathtt{SBR}(\mathtt{SBR}({}^{10}\bar{S}_0^0, \theta), 128) \\
&= (\mathtt{SBR}({}^{10}S_0^0, \theta) \oplus 128) \oplus (\mathtt{SBR}({}^{10}\bar{S}_0^0, \theta) \oplus 128) \\
&= (\mathtt{SBR}({}^{10}S_0^0, \theta) \oplus \mathtt{SBR}({}^{10}\bar{S}_0^0, \theta)) \oplus (128 \oplus 128) \\
&= {}^{10}\Delta S_0^1 \oplus 0 = {}^{10}\Delta S_0^1
\end{aligned}$$

The theoretical expected average keyspace of Section 3.3.4 overlooks this ($\boxplus$ 128) equivalence. Including this additional information, the theoretical mean after one faulty ciphertext drops to $2^{39.0677}$. Accounting for this equivalence in the observed data, the observed mean is $2^{39.1693}$. This adjustment does not consider ${}^{10}R$ not 128 apart with the same ${}^{10}K$ which possibly creates an overcount of remaining ${}^{10}K$. However, any such overcount is likely negligible. The increased observed mean remaining keyspace over the theoretical existing AES attack is likely due to an inherent skew in the data not fully or properly captured in the theoretical analysis. Although the theoretical mean reductions are underestimations, they are still great estimates of the magnitude of the remaining keyspace as evidence in the close $\log_2$ transformed means. In most areas of work, an error of $2^{39.1693} - 2^{39.0677} = 42,071,374,371$ is not considered negligible, or even acceptable. However, the difference of the $\log_2$ of the means is only 0.1016. For purposes of knowing the general work factor and computing power necessary to perform an attack, the theoretical analysis is more than sufficient.

Figure 5.13 shows the number of valid ${}^{10}R$ values after two ciphertext reductions. Every single attack reduced to two ${}^{10}R$ values at this point meaning just the correct ${}^{10}R$ and

^{10}R $\oplus$ 128 remain. Since these share the same reduced keyspaces, the observed remaining keyspace overcounts the actual remaining keyspace by a factor of 2. Figure 5.14 displays the $\log_2$ transformed histogram of the true remaining keyspace accounting for this double count.

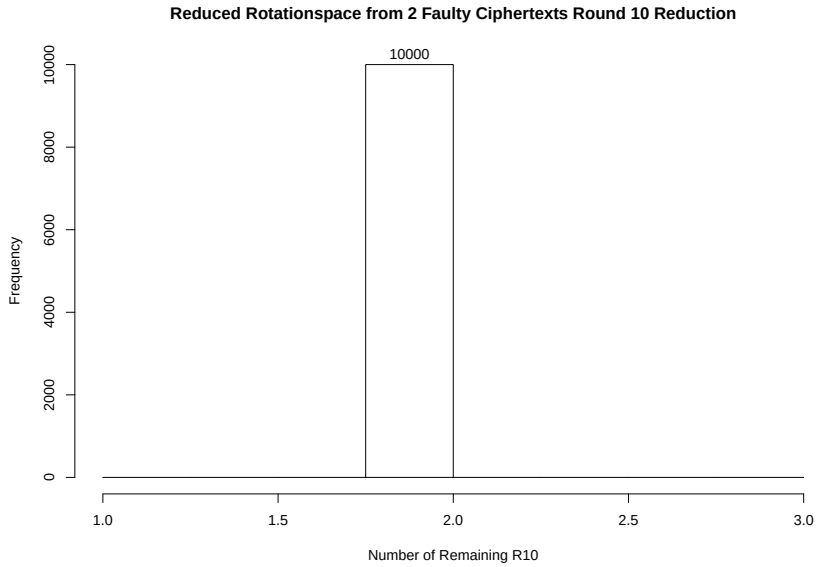

Figure 5.13: Histogram of Remaining R_{10}, 2 Faulty Ciphertext Round 10 Reduction.

The vast majority of the time, specifically 9,263 of 10,000 attacks, a reduction to one valid ^{10}K occurs after two faulty ciphertexts. However, compared to the attack on AES, a larger possible remaining keyspace of 64 is possible after this double reduction, with 2 and 4 much more common. The updated theoretical expected remaining keyspace accounting for the double count is $1+2^{-49.8646}$. The observed mean remaining ^{10}K keyspace is 1+.1893. As with the existing attack, proper explanation of this disparity is not possible without further data and analysis. Figure 5.15 shows that reducing with a third faulty ciphertext leaves just 8 total attacks that still require reduction at 2 and 4 ^{10}K values. Application of the key schedule reversal reductions would likely reduce these to just 1 ^{10}K value, however this attack implementation did not attempt that reduction.

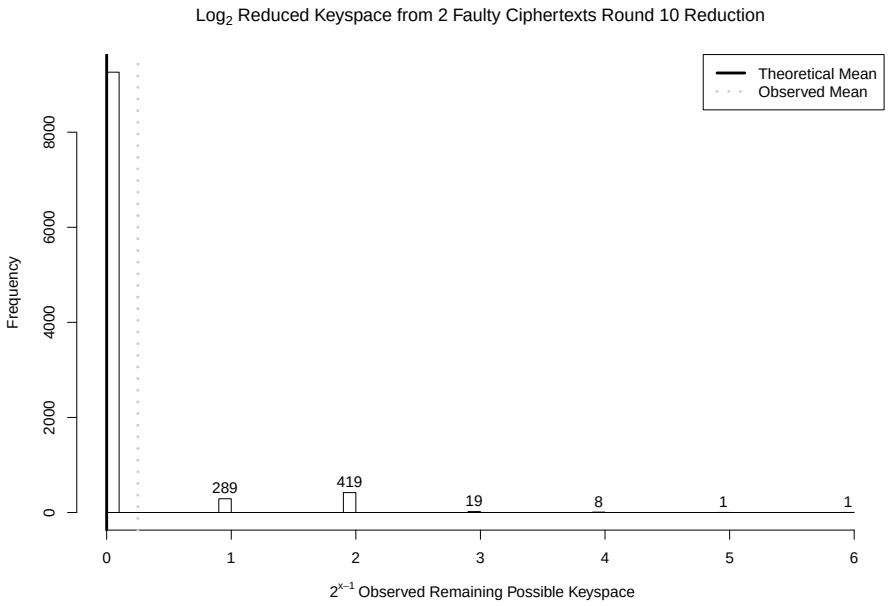

Figure 5.14: Histogram of Log_2 Transformed Observed/2 RAES-128 ^{10}K Keyspace, 2 Faulty Ciphertext Round 10 Reduction.

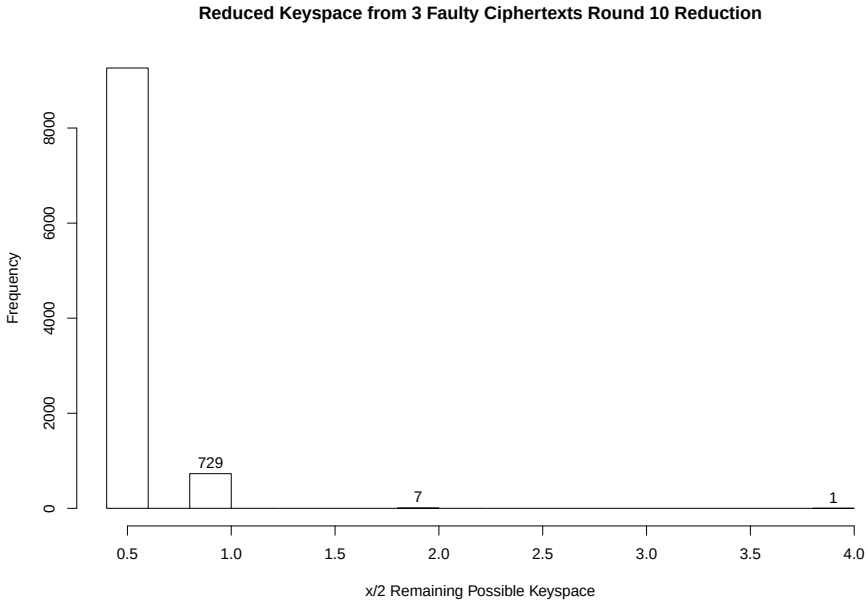

Figure 5.15: Histogram of Observed/2 RAES-128 ^{10}K Keyspace, 3 Faulty Ciphertext Round 10 Reduction.

Figure 5.16 shows that total keyspace reduction requires a maximum of four cipher pairs, thus no further keyspace histograms are necessary for analysis. The increased remaining keyspace of $2^{39.1693}$ after 1 faulty ciphertext does not explain the increased number of remaining valid ^{10}K after multiple faulty ciphertexts compared to the theoretical expected value or the existing attack on AES. Since only the two ^{10}R 128 apart remain after two or more faulty ciphertexts and these share the same keyspace, effectively only the keyspace associated with one S-box remains. Were the reductions between all S-boxes uniform, Figure 5.14 would be indistinguishable from a $\log_2$ transformation of Figure 5.6. Therefore, the less evenly distributed key byte densities of rotated S-boxes, which do not change the average reduction, impact discrete cases by increasing overlapping matches between reductions. Figure 5.17 displays the runtime required for full key recovery. Meaningful in depth analysis of runtimes is not valid. However, this increased average runtime of 11.38 seconds - over 60 times the runtime required for the existing attack on AES - provides context for the work factor of the attack.

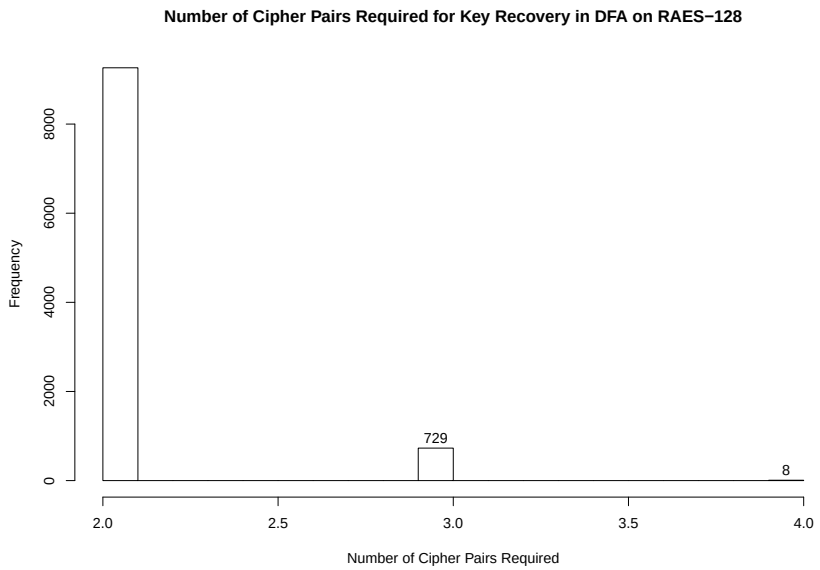

Figure 5.16: Histogram of Required Cipher Pairs for Full Key Recovery on RAES-128.

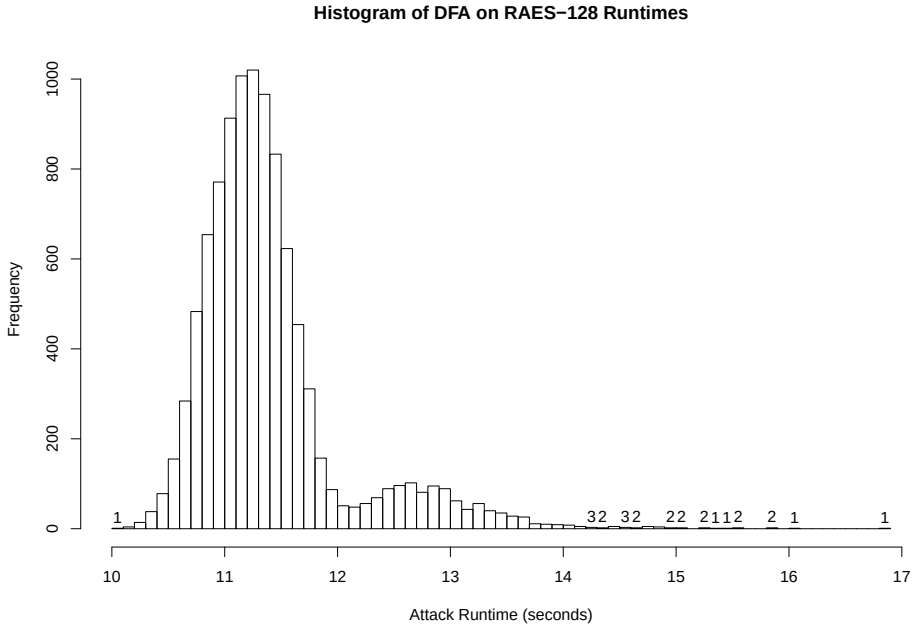

Figure 5.17: Histogram of RAES-128 DFA Runtime.

5.3 Design Suggestions

Recovery of the round 10 encryption key is the crux of existing DFA attacks. RAES implementations create more complexity in reversing ^{10}K to the encryption key. Since the key schedule does not need to be invertible (both encryption and decryption logically step forward through the key schedule, only optimization possibly uses invertibility), adding further complexity to this algorithm such that reversal is infeasible would weaken these attacks. Changing the key schedule entirely, creating a one way function would also accomplish this goal, but modifying the existing key schedule requires less alterations and would take less work to vet since over a decade of research and use well establish the foundations of the existing key schedule. A modified key schedule which creates three expanded keys follows.

- **Key Schedule 3.** The RAES key schedule 2 creates two expanded keys. As intended, the first is the expanded rotation key. However, the second is the expanded key

schedule rotation key. XORing all 16 bytes of each round creates expanded key schedule rotation values. These values define the S-box$_i$ used in the SW operation of the third key expansion. This third expanded key built is the expanded encryption key.

Previously, the expansion of the expanded encryption key only relied on one S-box$_i$ creating 256 possible ways to rebuild the encryption key from ^{10}K. This new scheme uses a rotation value for each round of the encryption key schedule effectively creating a $(2^8)^{10}$ work factor to reverse ^{10}K to the encryption key. Now, even with a successful DFA on the State, reversal through the key schedule is computationally infeasible. Instead, Eve needs to perform this DFA attack on each round (^{10}K enables decryption to the end of round 9, ^{9}K enables decryption to the end of round 8, etc.) until the work factor is computationally feasible. This attack method increases the resources and access required of an attacker by increasing the number of faulty ciphertext pairs required. DFA on the Key Schedule could possibly bypass some of this work factor or leverage a more powerful fault injection, but since each round rotates by a separate value, full reversal would still likely be infeasible or highly costly with few faults. Following is an updated list of implementations which includes this new key schedule.

- **RAES Type 1.** Key Schedule 1 and Rotation Reduction 1
- **RAES Type 2.** Key Schedule 1 and Rotation Reduction 2
- **RAES Type 3.** Key Schedule 2 and Rotation Reduction 1
- **RAES Type 4.** Key Schedule 2 and Rotation Reduction 2
- **RAES Type 5.** Key Schedule 3 and Rotation Reduction 1
- **RAES Type 6.** Key Schedule 3 and Rotation Reduction 2

5.4 Analysis Summary

The mean observed remaining keyspace after applying the existing attack to AES-128 is higher than expected. The theoretical analysis explains the range and relative densities of values seen, but does not fully account for the high outliers. However, this analysis still provides a good estimate of the general expected complexity and reduction power. Analysis of the observed extension data reveals that the S-box rotation only introduces a complexity of 128, not 256 for DFA attacks because $\boxplus\ 128 = \oplus\ 128$. Updating the theoretical analysis to include this information reduces the expected mean to $2^{39.0677}$. This theoretical mean also underestimates the observed remaining keyspace, but again the theoretical analysis explains the range and relative densities of the values seen, while not fully accounting for the high outliers. This analysis still provides a good estimate of the general expected complexity and reduction power. The increase in attack time from AES-128 to RAES-128 highlights the increased complexity RAES introduces. Altering the RAES key schedule creates an effectively one way expanded encryption key expansion which mitigates DFA on the State.

VI. Conclusion

Overall this research shows initial progress into the extension of applying existing DFA techniques to Dynamic S-box AES implementations. This research uses RAES-128 for non-trivial simplicity, proof of concept, and flexibility in initial exploratory attempts. Analysis produced a reasonable attack requiring one fault on the State in round 8, with full key recovery possible with two or more faulty ciphertexts. The theoretical analysis of the existing attack on AES slightly overestimates the observed average reduction power. Analysis of the extension's experimentation data revealed additional information allowing the theoretical analysis to reduce to $2^{39.0677}$. This value also slightly overestimates the observed average reduction power.

6.1 Impact

This research reveals RAES-128 Types 1-4 are nominally more secure than AES-128 against DFA attacks on the State. Therefore, these RAES implementations should still incorporate current DFA mitigation techniques when securing high value data. The proposed implementations, RAES-128 Types 5-6, should make key reversal more difficult and costly, protecting the encryption key. However, DFA still enables full decryption with sufficient resources. Therefore, RAES-128 Types 5-6 should still incorporate current DFA mitigation techniques when securing high value data. Despite the potential of RAES-128 Types 5-6, this paper still recommends use of non-proprietary best practice AES implementations following the guidelines established in [2] because these platforms are transparent, well established and regularly publicly reviewed and updated.

6.2 Contributions

This research made several contributions to cryptology.

- **This research determines extension of current DFA attacks to Dynamic S-box AES variants is possible.** Chapter 3 extends DFA on the State to RAES-128 implementations.

- **This research reveals expected keyspace reduction power of DFA extensions.** Section 3.3.4 expects DFA on the State of all four RAES-128 implementations to have a reduction power of $2^{-88.9323}$.

- **This research builds functional attacks which demonstrate full key and plaintext recovery.** Chapter 5 discusses how the (R)AES-DFA attack simulation platform successfully attacks AES-128 and all RAES-128 implementations.

- **This research provides an easy to follow and self-contained resource which walks through the mechanics and analysis of DFA attacks.** Chapters 2, 3 & 5 create a thorough introduction to DFA on the State.

- **This research appreciably adds to the overall security analysis of Dynamic S-box AES variants.** Chapters 3 & 5 provide insight on DFA concerns.

- **This research contributes to the literature of theoretical analysis.** Chapter 3 updates the existing analysis of DFA on the State of AES-128, and yields new analysis of DFA on the State of RAES-128. Chapter 5 compares theoretical analysis to experimental data.

- **This research helps inform and shape future discussions of cryptographic standards and algorithmic design decisions.** An irreversible key schedule would significantly mitigate DFA attack power.

6.3 Future Work

This research extends to a simple non-trivial Dynamic S-box AES variant. As such, many vectors of improvement, expanded scope, and future work exist. Most directly,

this manifests in leveraging the round 9 relations with a reasonable work factor and extending DFA on the State to RAES-192 and RAES-256 implementations. Also, the current theoretical analysis models need further attention to fully and more accurately capture the expected reduction power of attacks. Other work includes implementing, testing, and attacking the proposed RAES implementations, Types 5 and 6, or one similar which theoretically makes key reversal computationally infeasible from the last round key. Extending other DFA attacks, specifically DFA on the Key Schedule to RAES implementations is another area of interesting future work. The last area of future work is extending all DFA attacks to other Dynamic S-box variants.

Appendix A: Discussion of Rotational S-box Design Decisions

AES-KDS as described in [22] leaves several implementation details open to interpretation. This appendix first presents higher level questions, then discusses case and type specific ambiguities.

The most significant unknown is directly related to the S-box rotation. The round rotation value "...is used to rotate the S-box. The resulting S-box is used during the *SubBytes* operation." And "each round AES-KDS S-box can have 256 possible entries. Totally there are 10 rounds. So total number of possible S-boxes is given by,

$$256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 = 2^{80}\text{"}.$$

These passages make clear each round rotates the S-box, but do not define if this is the standard AES S-box or the previous round's S-box. That is, it is not well defined if the application of `RotSBox` is iterative. The only clue is provided by the pseudo code. Below is the section of the pseudo code for Case 2 encryption rounds 1 through 9:

```
for(round=1;round<=9;round++)
{
    rotate=(expanded_key[round*4]^expanded_key[round*4+1]
    ^expanded_key[round*4+2]^expanded_key[round*4+3])&mask;
    create_s_box(s_box,rotate);
// function to rotate S-box to left by a value equal to rotate
    substitute_bytes(state,s_box);
    shift_row(state);
    mix_column(state);
    add_round_key(round*4,state,expanded_key);
}
```

For this code to work as expected and described in [22], these function calls must be by reference. Otherwise, `create_s_box(s_box,rotate)` would create a newly rotated S-box that is never used in `substitute_bytes(state,s_box)` and further the state would never be updated. Thus, given only this information, an iterative S-box rotation is the most logical conclusion.

The other high-level problem is that the provided pseudo code does not provide sufficient detail to make data structures, variables, and operations well defined. Several instances of this lack of definition are now provided.

The first case of non-explicit definition is the pseudo code function `create_s_box(s_box,rotate)`, which is described only by the comment "\\ function to rotate S-box to left by a value equal to rotate". This follows the logical explanation of the `RotSBox` step, however no explicit definition or pseudo code is provided. This allows for the ambiguity of an iterative S-box to remain. As such, only a logical application of the description within the paper and pseudo code can be applied.

A similar problem exists with `key_expansion(expanded_key,key,s_box)`. This pseudo code function is only described by the comment '\\ as in original AES'. The pseudo code snippet below shows the context of `key_expansion` as used in the key schedule section of the algorithm:

```
rotate=temp;
create_s_box(s_box,rotate);
key_expansion(expanded_key1,key,s_box);
// as in original AES
for(i=0;i<44;i++)
    fprintf(ky1,"%lx ",expanded_key1[i]);
```

On a high level, this code appears to set a rotate value and rotate the S-box by this value.

Then, the encryption key is expanded using the standard AES key expansion algorithm with the exception that S-box lookups use this rotated S-box. The resulting expanded key is saved as expanded_key1. No explicit definition or pseudo code is provided for key_expansion, so logical interpretation is necessary.

Variable specific problems are also present in the pseudo code. For both Type 1 and 2 key schedules, the variable key receives a hard coded value. Below are pertinent snippets of code from each of these:

Type 1

```
unsigned char key[16]=1234567890ABCDEF;
⋮
create_s_box(s_box,rotate);
key_expansion(expanded_key,key,s_box);
```

Type 2

```
unsigned char key[16]=1234567890ABCDEF;
⋮
create_s_box(s_box,rotate);
key_expansion(expanded_key1,key,s_box);
⋮
create_s_box(s_box,shift);
key_expansion(expanded_key2,key,s_box);
```

Logical interpretation of this would mean key expansion and thus encryption was not dependent on the provided encryption key. If this were the case, encryption of a given plaintext with two different keys would result in the same resulting ciphertext. However, this is not the case as seen in the paper's experimental results. Therefor, this must be

interpreted as an example key value provided to clarify its structure and use in the pseudo code.

The next potential problem relates to Case 2 round rotation keys. The paper text describes the reduction from round rotation key to round rotation value as the 'XOR operation of all the bytes':

> Suppose for a particular round j, if the round key value is
>
> 06ACB47D588A9ED837D50E923C4055B5 (each byte represented by 2-Hex digits).
>
> Here XOR operation of all the bytes is taken.
>
> 15(Hex)=06^AC^B4^7D^58^8A^9E^D8^37^D5^0E^92^3C^40^55^B5 (^symbol used for XOR)
>
> The resulting byte value 15(Hex) is used to rotate the Sbox.

However, the pseudo code to accompany this does not logically perform the XOR as described. Provided is a pertinent snippet:

```
unsigned long int mask=0xff;
⋮
for(round=1;round<=9;round++)
{
    rotate=(expanded_key[round*4]^expanded_key[round*4+1]
    ^expanded_key[round*4+2]^expanded_key[round*4+3])&mask;
⋮
}
Rotate=(expanded_key[40]^expanded_key[41]
^expanded_key[42]^expanded_key[43])&mask;
```

The rotate value calculated in this pseudo code is only the XOR of 4 bytes, not all

16 bytes of the round rotation key. Each `expanded_key[i]` index must be a 4 byte value. This follows from many details. Traditionally, the expanded key of AES-128 is represented by a 4x44 matrix of bytes. This is a result of the way the key is expanded and computed columnwise. The indented `rotate` value represents `rotate` computed for rounds 1 through 9, while the second `Rotate` value represents `rotate` computed for round 10. Indices 40-43 are accessing the last four 'columns' of the expanded key with indexing starting at 0. Additionally, for four values to represent 16 bytes in total, each must represent 4 bytes. Tracing out the operation, first, four 4-byte values are XOR'ed resulting in one 4-byte value. This one 4-byte value is then bitwise AND'ed (&) with `mask=0xff`, the one byte value `1111 1111b`. Thus, depending on endianness, only the most or least significant byte is saved into `rotate`. The same logical reduction of round rotation key to round rotation value using the XOR of all bytes is again used in Case 4, "XOR operation of all bytes is taken", however no pseudo code is provided [22].

The last part of the algorithm which is not clearly defined is the Type 2 key schedule. Two rotation values are calculated, one from the encryption key as in Type 1, and the second from `expanded_key1`. How this second rotation value is logically formed is not described in the text which only notes, "These round keys are also used for finding a value for rotating the S-box, which will be used in generating [the] second set of round keys". Below is a pseudo code snippet which describes the second rotation value's calculation:

```
for(i=0;i<43;i++)
{
    expanded_key1[i+1]=expanded_key1[i]^expanded_key1[i+1];
}
for(i=0;i<=3;i++)
    for(j=0;j<=3;j++)
    {
```

```
        temp=expanded_key1[44]&mask;
        temp=temp>>shift1;
        shift1=shift1+8;
        mask=mask<<8;
        shift=shift^temp;
    }
create_s_box(s_box,shift);
key_expansion(expanded_key2,key,s_box);
```

Notably, `temp` is initialized as an `unsigned char` and therefor can hold up to a byte of information; `mask`, `shift`, and `shift1` are not initialized earlier in this code section. The first `for` loop XOR's each 'column' of the expanded key together. The first time through the loop `expanded_key1[1]` is XOR'ed with `expanded_key1[0]`. The second pass XOR's `expanded_key1[2]` with `expanded_key1[1]`. At this point `expanded_key1[1]` is `expanded_key1[1]^expanded_key1[0]`. Thus, by the last pass of the loop, `expanded_key1[43] = expanded_key1[43]^expanded_key1[42]^` $\cdots$ `^expanded_key1[1]^expanded_key1[0]`. The next nested `for` loop section is where lack of explicit details becomes problematic. Ignoring the nested loops and only examining the contents, these five lines of code appear to XOR several bytes together.

In the first line, `temp=expanded_key1[44]&mask`, `mask` is used, which is not initialized in this code section; however, `mask` is initialized in an earlier pseudo code section detailing Case 2 encryption. If the same initialization is assumed, let `mask=0xff`. Additionally, `expanded_key1[44]` is referenced, however the previous loop only iterates through `expanded_key1[43]`. As discussed previously, `expanded_key1` must be an array of length 44 (and so indices 0-43) to represent the 44 'columns' of the expanded key. To further this interpretation, Case 2 encryption pseudo code only accesses indices 0-43 of `expanded_key`. Thus, let the 44 reference be assumed to be a typo which should read

43. This then makes the first line set `temp` to the 'first' byte (most or least significant byte depending on endianness) of `expanded_key[43]`.

Moving to the next line, `temp=temp>>shift1`, `shift1` is unknown. However, as `shift1` is incremented by 8 each iteration, or one byte (as seen in the third line), and `temp` holds the value of just one byte, it is logical to assume `shift1=0` as the initialization. Thus, on the first pass, this line has no effect. The third line, `shift1=shift1+8`, as previously mentioned increments `shift1` by 8, or one byte. The fourth line bit shifts `mask` to the left by one byte. This logically agrees with the prior assumption of its initialization. The last line is the most interesting, `shift=shift^temp`. `shift` is not initialized, however, an initialization to 0 is logical. This would set `shift=temp` which is the 'first' byte of `expanded_key1[43]` on the first pass.

Tracing through subsequent passes, `temp` is set to the next byte of `expanded_key1[43]` and bit shifted by one byte so there is not a byte worth of trailing zeros. `shift1` and `mask` are appropriately updated, and `shift` is XOR'ed with this second byte. Thus `shift` is now the XOR of the first two bytes of `expanded_key1[43]`. After the first four iterations, `shift` is the XOR of all four bytes of `expanded_key1[43]`, or logically when considering the prior loop, the XOR of every byte in `expanded_key1`. The utility of the outer `for` loop is not apparent. If it is not a misprint, then the resulting value of `shift` would end up being 0 (`x^x^x^x=0` for any given value `x`). As the iterators `i` and `j` are not referenced in the content of the loops, each iteration of the outer loop would be exactly the same. Based on the other calculations performed to reduce the round keys to round rotation values in Cases 2 and 4, and the reductions of the encryption key for the first rotation values computed in the key schedules, all of which are the XOR of all bytes being handled, it is logical to assume the nested loops is a typo and this block is intended to XOR all the bytes of `expanded_key1`.

For this research, an implementation of this algorithm was written in Python 2.7 building off of the Scripting Languages Open-source Workable AES (SLOWAES) code base [8]. First the code's base functions were tested for proper functionality by using NIST sample values and walkthroughs [1], and were successfully validated. Next, the algorithm as best described by [22] and discussed above was implemented on top of this validated AES implementation. Each step was validated and carefully examined and stepped through to ensure encryption was following all expected logical flow. The rotate step specifically was vetted with the example rotated S-box as provided by [22]. Validation of this implementation of AES-KDS was reliant on the sample encryption data provided in the paper which is shown below:

Key = ADF278565E262AD1F5DEC94A0BF25B27

SN	Plaintext	Ciphertext
1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	B2 43 B5 85 CA DD F4 4E F5 E6 6E D1 D7 08 B3 0B
2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01	37 14 10 49 D7 DE 2D 56 CC 74 66 6B CF 89 4C 95
3	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01	64 E2 C1 53 C4 79 78 DF FC 87 35 15 9F 4C 39 76
4	10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	2F 1D C4 1D DB 52 D6 9D A5 74 99 69 9B 16 31 9E

Table.1. Plaintext & Ciphertext samples

Key = ADF278565E262AD1F5DEC94A0BF25B28

SN	Plaintext	Ciphertext
1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	21 55 66 73 D8 BE 4F 9D 98 55 68 D0 06 DC E6 35
2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01	32 67 6F 89 15 6E 88 80 D0 82 07 9A 0E E2 35 07
3	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 01	25 AE 71 C2 4F E0 7F A3 AD 23 35 84 31 47 2B 9E
4	10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	C9 89 71 71 A1 F0 9D 4A 80 A9 6D CF F3 EE 40 4C

Figure A.1: AES-KDS Validation Encryptions [22].

It is important to note that only sample encryptions for Case 4, Type 2 as described in [22] are provided, and no intermediate steps or key schedule data is available. Thus,

validation of this algorithm is wholly dependent on just 8 encryptions and only one Case can truly be validated. To accomplish this, encryption of the four plaintexts with the two keys was performed, however the resulting ciphertexts did not match those provided. Because of the ambiguities described in the prior section, the possibility of a misinterpretation was reasonable. Thus, iterative encryptions testing these possibilities was performed. What follows is a description of each moving part tried, and an analysis of how many combinations were tested.

– Iterative S-box encryption: Is the S-box iteratively rotated between rounds of encryption? [Yes/No] **2 possibilities**.

– Iterative S-box key schedule: Is the S-box iteratively rotated between creation of `expanded_key1` and `expanded_key2`? This is handled by a later iterator, **1 possibilities**.

– Iterative S-box key schedule to encryption: Is the S-box iteratively rotated between the key schedule and the encryption rounds? [Yes/No] **2 possibilities**.

– How is the round rotation key reduced to the round rotation value? [XOR of all bytes as logically described/XOR of most significant bytes as in pseudo code/XOR of least significant bytes as in pseudo code] **3 possibilities**.

– In the key schedule, how is the first `rotate` value calculated? [XOR of all bytes of encryption key/XOR of all bytes of hardcoded pseudo code key] Because of the importance of using the correct keys and the lack of explicit definition, all 256 rotation values [0-255] are used. **256 possibilities**.

– In the key schedule, how is the second rotate value `shift` calculated? [XOR of all bytes of `expanded_key1`] Because of the lack of explicit definition of how the rotation value is calculated, and if rotation is iterative with the first performed, all 256 rotation values [0-255] are used. **256 possibilities**.

– Expanded key altered by XOR of columns [Yes/No] **2 possibilities**.

– Next the number of encryption keys checked is discussed. To cover potential

implementation specific issues, alternate endianess representations of the two keys for architectures ranging from 16-bit to 128-bit (excessively large range for completeness) are used. Note this is only applied to the keys and not plaintext because the all 0 plaintext will still result in a match. *8 possibilities*. When the encryption key is stored as a 4x4 matrix, by design it is to be stored by filling the rows. When the plaintext is stored as a 4x4 matrix, by design it is to be stored by filling the columns. To cover any potential mixup of these details, the transpose of the key is also checked. *2x possibilities*. Totally that makes $(2 + 8) \times 2 = 20$ encryption keys. **20 possibilities**.

– Finally the number of plaintext checked is discussed. As mentioned above, the key and plaintext are stored in a different indexing. To overcome a potential mixup, the transpose of each plaintext is also encrypted. *2x possibilities*. Totally this makes $4 \times 2 = 8$ plaintext. **8 possibilities**.

When all these moving parts are checked in totality, it amounts to $2 \times 1 \times 2 \times 3 \times 256 \times 256 \times 2 \times 20 \times 8 = 251,658,240$ or approximately a quarter of a billion encryptions. All of these were checked, and no match was found. The authors of the paper were also reached out to for more validation data, intermediate calculations, or more detail and definition, but no response was received. Thus, given the thorough validation efforts, and the amount of ambiguity found in [22], an implementation was chosen which was most logical and followed most directly from the data provided in the paper. This implementation, RAES, is logically described in Section 3.2.2, with walk through encryption and key schedule examples to best facilitate repeatability and future validation and verification provided in Appendix B.

Appendix B: RAES Validation Data

```
Round  0                                                                                                                Round Key  0
[ 50, 136,  49, 224]                                                                            [ 25, 160, 154, 233]    [ 43,  40, 171,   9]
[ 67,  90,  49,  55]                                                                         AK [ 61, 244, 198, 248]    [126, 174, 247, 207]
[246,  48, 152,   7] -------------------------------------------------------------------------> [227, 226, 141,  72]    [ 21, 210,  21,  79]
[168, 141, 162,  52]                                                                            [190,  43,  42,   8]    [ 22, 166, 136,  60]

Round  1                                                                                                                Round Key  1
[ 25, 160, 154, 233]    [212, 224, 184,  30]    [212, 224, 184,  30]    [  4, 224,  72,  40]    [164, 104, 107,   2]    [160, 136,  35,  42]
[ 61, 244, 198, 248] SB [ 39, 191, 180,  65] SR [191, 180,  65,  39] MC [102, 203, 248,   6] AK [156, 159,  91, 106]    [250,  84, 163, 108]
[227, 226, 141,  72] -> [ 17, 152,  93,  82] -> [ 93,  82,  17, 152] -> [129,  25, 211,  38] -> [127,  53, 234,  80]    [254,  44,  57, 118]
[190,  43,  42,   8]    [174, 241, 229,  48]    [ 48, 174, 241, 229]    [229, 154, 122,  76]    [242,  43,  67,  73]    [ 23, 177,  57,   5]

Round  2                                                                                                                Round Key  2
[164, 104, 107,   2]    [ 73,  69, 127, 119]    [ 73,  69, 127, 119]    [ 88,  27, 219,  27]    [170,  97, 130, 104]    [242, 122,  89, 115]
[156, 159,  91, 106] SB [222, 219,  57,   2] SR [219,  57,   2, 222] MC [ 77,  75, 231, 107] AK [143, 221, 210,  50]    [194, 150,  53,  89]
[127,  53, 234,  80] -> [210, 150, 135,  83] -> [135,  83, 210, 150] -> [202,  90, 202, 176] -> [ 95, 227,  74,  70]    [149, 185, 128, 246]
[242,  43,  67,  73]    [137, 241,  26,  59]    [ 59, 137, 241,  26]    [241, 172, 168, 229]    [  3, 239, 210, 154]    [242,  67, 122, 127]

Round  3                                                                                                                Round Key  3
[170,  97, 130, 104]    [172, 239,  19,  69]    [172, 239,  19,  69]    [117,  32,  83, 187]    [ 72, 103,  77, 214]    [ 61,  71,  30, 109]
[143, 221, 210,  50] SB [115, 193, 181,  35] SR [193, 181,  35, 115] MC [236,  11, 192,  37] AK [108,  29, 227,  95]    [128,  22,  35, 122]
[ 95, 227,  74,  70] -> [207,  17, 214,  90] -> [214,  90, 207,  17] -> [  9,  99, 207, 208] -> [ 78, 157, 177,  88]    [ 71, 254, 126, 136]
[  3, 239, 210, 154]    [123, 223, 181, 184]    [184, 123, 223, 181]    [147,  51, 124, 220]    [238,  13,  56, 231]    [125,  62,  68,  59]

Round  4                                                                                                                Round Key  4
[ 72, 103,  77, 214]    [ 82, 133, 227, 246]    [ 82, 133, 227, 246]    [ 15,  96, 111,  94]    [224, 200, 217, 133]    [239, 168, 182, 219]
[108,  29, 227,  95] SB [ 80, 164,  17, 207] SR [164,  17, 207,  80] MC [214,  49, 192, 179] AK [146,  99, 177, 184]    [ 68,  82, 113,  11]
[ 78, 157, 177,  88] -> [ 47,  94, 200, 106] -> [200, 106,  47,  94] -> [218,  56,  16,  19] -> [127,  99,  53, 190]    [165,  91,  37, 173]
[238,  13,  56, 231]    [ 40, 215,   7, 148]    [148,  40, 215,   7]    [169, 191, 107,   1]    [232, 192,  80,   1]    [ 65, 127,  59,   0]

Round  5                                                                                                                Round Key  5
[224, 200, 217, 133]    [225, 232,  53, 151]    [225, 232,  53, 151]    [ 37, 189, 182,  76]    [241, 193, 124,  93]    [212, 124, 202,  17]
[146,  99, 177, 184] SB [ 79, 251, 200, 108] SR [251, 200, 108,  79] MC [209,  17,  58,  76] AK [  0, 146, 200, 181]    [209, 131, 242, 249]
[127,  99,  53, 190] -> [210, 251, 150, 174] -> [150, 174, 210, 251] -> [169, 209,  51, 192] -> [111,  76, 139, 213]    [198, 157, 184,  21]
[232, 192,  80,   1]    [155, 186,  83, 124]    [124, 155, 186,  83]    [173, 104, 142, 176]    [ 85, 239,  50,  12]    [248, 135, 188, 188]

Round  6                                                                                                                Round Key  6
[241, 193, 124,  93]    [161, 120,  16,  76]    [161, 120,  16,  76]    [ 75,  44,  51,  55]    [ 38,  61, 232, 253]    [109,  17, 219, 202]
[  0, 146, 200, 181] SB [ 99,  79, 232, 213] SR [ 79, 232, 213,  99] MC [134,  74, 157, 210] AK [ 14,  65, 100, 210]    [136,  11, 249,   0]
[111,  76, 139, 213] -> [168,  41,  61,   3] -> [ 61,   3, 168,  41] -> [141, 137, 244,  24] -> [ 46, 183, 114, 139]    [163,  62, 134, 147]
[ 85, 239,  50,  12]    [252, 223,  35, 254]    [254, 252, 223,  35]    [109, 128, 232, 216]    [ 23, 125, 169,  37]    [122, 253,  65, 253]

Round  7                                                                                                                Round Key  7
[ 38,  61, 232, 253]    [247,  39, 155,  84]    [247,  39, 155,  84]    [ 20,  70,  39,  52]    [ 90,  25, 163, 122]    [ 78,  95, 132,  78]
[ 14,  65, 100, 210] SB [171, 131,  67, 181] SR [131,  67, 181, 171] MC [ 21,  22,  70,  42] AK [ 65,  73, 224, 140]    [ 84,  95, 166, 166]
[ 46, 183, 114, 139] -> [ 49, 169,  64,  61] -> [ 64,  61,  49, 169] -> [181,  21,  86, 216] -> [ 66, 220,  25,   4]    [247, 201,  79, 220]
[ 23, 125, 169,  37]    [240, 255, 211,  63]    [ 63, 240, 255, 211]    [191, 236, 215,  67]    [177,  31, 101,  12]    [ 14, 243, 178,  79]

Round  8                                                                                                                Round Key  8
[ 90,  25, 163, 122]    [190, 212,  10, 218]    [190, 212,  10, 218]    [  0, 177,  84, 250]    [234,   4, 101, 133]    [234, 181,  49, 127]
[ 65,  73, 224, 140] SB [131,  59, 225, 100] SR [ 59, 225, 100, 131] MC [ 81, 200, 118,  27] AK [131,  69,  93, 150]    [210, 141,  43, 141]
[ 66, 220,  25,   4] -> [ 44, 134, 212, 242] -> [212, 242,  44, 134] -> [ 47, 137, 109, 153] -> [ 92,  51, 152, 176]    [115, 186, 245,  41]
[177,  31, 101,  12]    [200, 192,  77, 254]    [254, 200, 192,  77]    [209, 255, 205, 234]    [240,  45, 173, 197]    [ 33, 210,  96,  47]

Round  9                                                                                                                Round Key  9
[234,   4, 101, 133]    [135, 242,  77, 151]    [135, 242,  77, 151]    [ 71,  64, 163,  76]    [235,  89, 139,  27]    [172,  25,  40,  87]
[131,  69,  93, 150] SB [236, 110,  76, 144] SR [110,  76, 144, 236] MC [ 55, 212, 112, 159] AK [ 64,  46, 161, 195]    [119, 250, 209,  92]
[ 92,  51, 152, 176] -> [ 74, 195,  70, 231] -> [ 70, 231,  74, 195] -> [148, 228,  58,  66] -> [242,  56,  19,  66]    [102, 220,  41,   0]
[240,  45, 173, 197]    [140, 216, 149, 166]    [166, 140, 216, 149]    [237, 165, 166, 188]    [ 30, 132, 231, 210]    [243,  33,  65, 110]

Round 10                                                                                                                Round Key 10
[235,  89, 139,  27]    [233, 203,  61, 175]    [233, 203,  61, 175]                            [ 57,   2, 220,  25]    [208, 201, 225, 182]
[ 64,  46, 161, 195] SB [  9,  49,  50,  46] SR [ 49,  50,  46,   9]                         AK [ 37, 220,  17, 106]    [ 20, 238,  63,  99]
[242,  56,  19,  66] -> [137,   7, 125,  44] -> [125,  44, 137,   7] -------------------------> [132,   9, 133,  11]    [249,  37,  12,  12]
[ 30, 132, 231, 210]    [114,  95, 148, 181]    [181, 114,  95, 148]                            [ 29, 251, 151,  50]    [168, 137, 200, 166]
```

Figure B.1: AES-128 Encryption and Expanded Key of [1] Example Data.

```
Round 0                                                                                                                      Round Encryption Key   Round Rotation Key
[ 50, 136,  49, 224]                                                                              [ 25, 160, 154, 233]   [ 43,  40, 171,   9]   [ 43,  40, 171,   9]
[ 67,  90,  49,  55]                                                                           AK [ 61, 244, 198, 248]   [126, 174, 247, 207]   [126, 174, 247, 207]   Iterative Rotation Value
[246,  48, 152,   7] ---------------------------------------------------------------------------> [227, 226, 141,  72]   [ 21, 210,  21,  79]   [ 21, 210,  21,  79]             208
[168, 141, 162,  52]                                                                              [190,  43,  42,   8]   [ 22, 166, 136,  60]   [ 22, 166, 136,  60]

Round 1                                                                                                                      Round Encryption Key   Round Rotation Key
[ 25, 160, 154, 233]     [ 26, 161, 155, 234]     [162,  50,  20, 135]     [162,  50,  20, 135]     [118,   6, 128,  56]     [135, 223, 242,  67]   [241, 217, 114, 123]   [241, 217, 114, 123]
[ 61, 244, 198, 248] SBR [ 62, 245, 199, 249] SB  [178, 230, 198, 153] SR  [230, 198, 153, 178] MC  [ 95, 224, 247,  58] AK  [225, 240,  16,  18]   [190,  16, 231,  40]   [190,  16, 231,  40]   Iterative Rotation Value
[227, 226, 141,  72] --> [228, 227, 142,  73] --> [105,  17,  25,  59] --> [ 25,  59, 105,  17] --> [117, 154, 204,  31] --> [158, 163, 224, 124]   [235,  57,  44,  99]   [235,  57,  44,  99]              1
[190,  43,  42,   8]     [191,  44,  43,   9]     [  8, 113, 241,   1]     [  1,   8, 113, 241]     [  0, 187,  46, 200]     [ 35,  62,  35, 249]   [ 35, 133,  13,  49]   [ 35, 133,  13,  49]

Round 2                                                                                                                      Round Encryption Key   Round Rotation Key
[135, 223, 242,  67]     [177,   9,  28, 109]     [200,   1, 156,  60]     [200,   1, 156,  60]     [ 55,  94, 168,  91]     [133,  53, 177,  57]   [178, 107,  25,  98]   [178, 107,  25,  98]
[225, 240,  16,  18] SBR [ 11,  26,  58,  60] SB  [ 43, 162, 128, 235] SR  [162, 128, 235,  43] MC  [ 24, 149,  55,  85] AK  [101, 248, 189, 247]   [125, 109, 138, 162]   [125, 109, 138, 162]   Iterative Rotation Value
[158, 163, 224, 124] --> [200, 205,  10, 166] --> [232, 189, 103,  36] --> [103,  36, 232, 189] --> [206, 247, 115,  72] --> [ 89,  89, 241, 169]   [151, 174, 130, 225]   [151, 174, 130, 225]             42
[ 35,  62,  35, 249]     [ 77, 104,  77,  35]     [227,  69, 227,  38]     [ 38, 227,  69, 227]     [202, 122,  54,  15]     [ 90, 111,  46,  38]   [144,  21,  24,  41]   [144,  21,  24,  41]

Round 3                                                                                                                      Round Encryption Key   Round Rotation Key
[133,  53, 177,  57]     [ 70, 246, 114, 250]     [ 90,  66,  64,  45]     [ 90,  66,  64,  45]     [237,  39, 146,  37]     [ 27, 186,  22, 195]   [246, 157, 132, 230]   [246, 157, 132, 230]
[101, 248, 189, 247] SBR [ 38, 185, 126, 184] SB  [247,  86, 243, 108] SR  [ 86, 243, 108, 247] MC  [ 59,  22,  97, 250] AK  [142, 206,  51,  10]   [181, 216,  82, 240]   [181, 216,  82, 240]   Iterative Rotation Value
[ 89,  89, 241, 169] --> [ 26,  26, 178, 106] --> [162, 162,  55,   2] --> [ 55,   2, 162, 162] --> [197,  95, 127, 255] --> [203, 255,  93,  60]   [ 14, 160,  34, 195]   [ 14, 160,  34, 195]            193
[ 90, 111,  46,  38]     [ 27,  48, 239, 231]     [175,   4, 223, 148]     [148, 175,   4, 223]     [188, 114,   6, 135]     [ 15, 212, 184,  16]   [179, 166, 190, 151]   [179, 166, 190, 151]

Round 4                                                                                                                      Round Encryption Key   Round Rotation Key
[ 27, 186,  22, 195]     [ 78, 237,  73, 246]     [ 47,  85,  59,  66]     [ 47,  85,  59,  66]     [160, 123,  97, 198]     [228, 162,  60, 125]   [ 68, 217,  93, 187]   [ 68, 217,  93, 187]
[142, 206,  51,  10] SBR [193,   1, 102,  61] SB  [120, 124,  51,  39] SR  [124,  51,  39, 120] MC  [109, 252, 102,  62] AK  [  4,  77, 133,  45]   [105, 177, 227,  19]   [105, 177, 227,  19]   Iterative Rotation Value
[203, 255,  93,  60] --> [254,  50, 144, 111] --> [187,  35,  96, 168] --> [ 96, 168, 187,  35] --> [189,  89,  37,  92] --> [ 54, 114,  44, 150]   [139,  43,   9, 202]   [139,  43,   9, 202]             51
[ 15, 212, 184,  16]     [ 66,   7, 235,  67]     [ 44, 197, 233,  26]     [ 26,  44, 197, 233]     [ 89,  60,  64,  84]     [164, 103, 165,  38]   [253,  91, 229, 114]   [253,  91, 229, 114]

Round 5                                                                                                                      Round Encryption Key   Round Rotation Key
[228, 162,  60, 125]     [ 35, 225, 123, 188]     [ 38, 248,  33, 101]     [ 38, 248,  33, 101]     [210, 221,  11,  69]     [151,  65, 202,  63]   [ 69, 156, 193, 122]   [ 69, 156, 193, 122]
[  4,  77, 133,  45] SBR [ 67, 140, 196, 108] SB  [ 26, 100,  28,  80] SR  [100,  28,  80,  26] MC  [ 34, 212,  25, 123] AK  [243, 180, 154, 235]   [209,  96, 131, 144]   [209,  96, 131, 144]   Iterative Rotation Value
[ 54, 114,  44, 150] --> [117, 177, 107, 213] --> [157, 200, 127,   3] --> [127,   3, 157, 200] --> [107, 209,  60,  79] --> [204,  93, 185,   0]   [167, 140, 133,  79]   [167, 140, 133,  79]             63
[164, 103, 165,  38]     [227, 166, 228, 101]     [ 17,  36, 105,  77]     [ 77,  17,  36, 105]     [235,  46, 230, 175]     [ 43, 181, 152, 163]   [192, 155, 126,  12]   [192, 155, 126,  12]

Round 6                                                                                                                      Round Encryption Key   Round Rotation Key
[151,  65, 202,  63]     [213, 127,   8, 125]     [  3, 210,  48, 255]     [  3, 210,  48, 255]     [ 22,  87, 254,  85]     [163, 126,  22, 199]   [181,  41, 232, 146]   [181,  41, 232, 146]
[243, 180, 154, 235] SBR [ 49, 242, 216,  41] SB  [199, 137,  97, 165] SR  [137,  97, 165, 199] MC  [ 74,  36, 197, 160] AK  [ 91,  85,  55, 194]   [ 17, 113, 242,  98]   [ 17, 113, 242,  98]   Iterative Rotation Value
[204,  93, 185,   0] --> [ 10, 155, 247,  62] --> [103,  20, 104, 178] --> [104, 178, 103,  20] --> [ 73, 220,  76,  17] --> [104, 113, 100, 118]   [ 33, 173,  40, 103]   [ 33, 173,  40, 103]             62
[ 43, 181, 152, 163]     [105, 243, 214, 225]     [249,  13, 246, 248]     [248, 249,  13, 246]     [ 15,  87, 136,  62]     [ 25, 218, 123, 193]   [ 22, 141, 243, 255]   [ 22, 141, 243, 255]

Round 7                                                                                                                      Round Encryption Key   Round Rotation Key
[163, 126,  22, 199]     [ 30, 249, 145,  66]     [114, 153, 129,  44]     [114, 153, 129,  44]     [  1, 243, 157, 213]     [215,  12, 138,  80]   [214, 255,  23, 133]   [214, 255,  23, 133]
[ 91,  85,  55, 194] SBR [214, 208, 178,  61] SB  [246, 112,  55,  39] SR  [112,  55,  39, 246] MC  [192,  45,   0, 208] AK  [ 75, 215,   8, 186]   [139, 250,   8, 106]   [139, 250,   8, 106]   Iterative Rotation Value
[104, 113, 100, 118] --> [227, 236, 223, 241] --> [ 17, 206, 158, 161] --> [158, 161,  17, 206] --> [  3, 145, 155, 155] --> [168, 151, 181, 210]   [171,   6,  46,  73]   [171,   6,  46,  73]            123
[ 25, 218, 123, 193]     [148,  85, 246,  60]     [ 34, 252,  66, 235]     [235,  34, 252,  66]     [181,  98,  77, 200]     [  9,  83, 143, 245]   [188,  49, 194,  61]   [188,  49, 194,  61]

Round 8                                                                                                                      Round Encryption Key   Round Rotation Key
[215,  12, 138,  80]     [224,  21, 147,  89]     [225,  89, 220, 203]     [225,  89, 220, 203]     [244,  95,  83,  75]     [ 34, 118, 109, 240]   [214,  41,  62, 187]   [214,  41,  62, 187]
[ 75, 215,   8, 186] SBR [ 84, 224,  17, 195] SB  [ 32, 225, 130,  46] SR  [225, 130,  46,  32] MC  [106,  95, 137, 246] AK  [ 53, 250,  36,  49]   [ 95, 165, 173, 199]   [ 95, 165, 173, 199]   Iterative Rotation Value
[168, 151, 181, 210] --> [177, 160, 190, 219] --> [200, 224, 174, 185] --> [174, 185, 200, 224] --> [145, 242, 167, 250] --> [237, 136, 243, 231]   [124, 122,  84,  29]   [124, 122,  84,  29]              9
[  9,  83, 143, 245]     [ 18,  92, 152, 254]     [201,  74,  70, 187]     [187, 201,  74,  70]     [ 26,  89,  13,  10]     [ 90,  40, 190, 132]   [ 64, 113, 179, 142]   [ 64, 113, 179, 142]

Round 9                                                                                                                      Round Encryption Key   Round Rotation Key
[ 34, 118, 109, 240]     [ 92, 176, 167,  42]     [ 74, 231,  92, 229]     [ 74, 231,  92, 229]     [202, 226,  95,  86]     [143, 142,  13, 191]   [ 69, 108,  82, 233]   [ 69, 108,  82, 233]
[ 53, 250,  36,  49] SBR [111,  52,  94, 107] SB  [168,  24,  88, 127] SR  [ 24,  88, 127, 168] MC  [167, 105,  71, 128] AK  [173, 198,  69,  69]   [ 10, 175,   2, 197]   [ 10, 175,   2, 197]   Iterative Rotation Value
[237, 136, 243, 231] --> [ 39, 194,  45,  33] --> [204,  37, 216, 253] --> [216, 253, 204,  37] --> [ 16,  56,  69, 196] --> [ 52, 102,  79, 211]   [ 36,  94,  10,  23]   [ 36,  94,  10,  23]             58
[ 90,  40, 190, 132]     [148,  98, 248, 190]     [ 34, 170,  65, 174]     [174,  34, 170,  65]     [ 89, 211,  24,  59]     [ 36, 223, 167,  10]   [125,  12, 191,  49]   [125,  12, 191,  49]

Round 10                                                                                                                     Round Encryption Key   Round Rotation Key
[143, 142,  13, 191]     [114, 113, 240, 162]     [ 64, 163, 140,  58]     [ 64, 163, 140,  58]                              [ 25, 150, 235, 180]   [ 89,  53, 103, 142]   [ 89,  53, 103, 142]
[173, 198,  69,  69] SBR [144, 169,  40,  40] SB  [ 96, 211,  52,  52] SR  [211,  52,  52,  96]                           AK [ 77,   5,   7, 150]   [158,  49,  51, 246]   [158,  49,  51, 246]   Iterative Rotation Value
[ 52, 102,  79, 211] --> [ 23,  73,  50, 182] --> [240,  59,  35,  78] --> [ 35,  78, 240,  59] --------------------------> [123,  72, 252,  32]   [ 88,   6,  12,  27]   [ 88,   6,  12,  27]            227
[ 36, 223, 167,  10]     [  7, 194, 138, 237]     [197,  37, 126,  85]     [ 85, 197,  37, 126]                              [126, 226, 189, 215]   [ 43,  39, 152, 169]   [ 43,  39, 152, 169]
```

Figure B.2: RAES-128 Type 1 Encryption and Expanded Key of [1] Example Data.

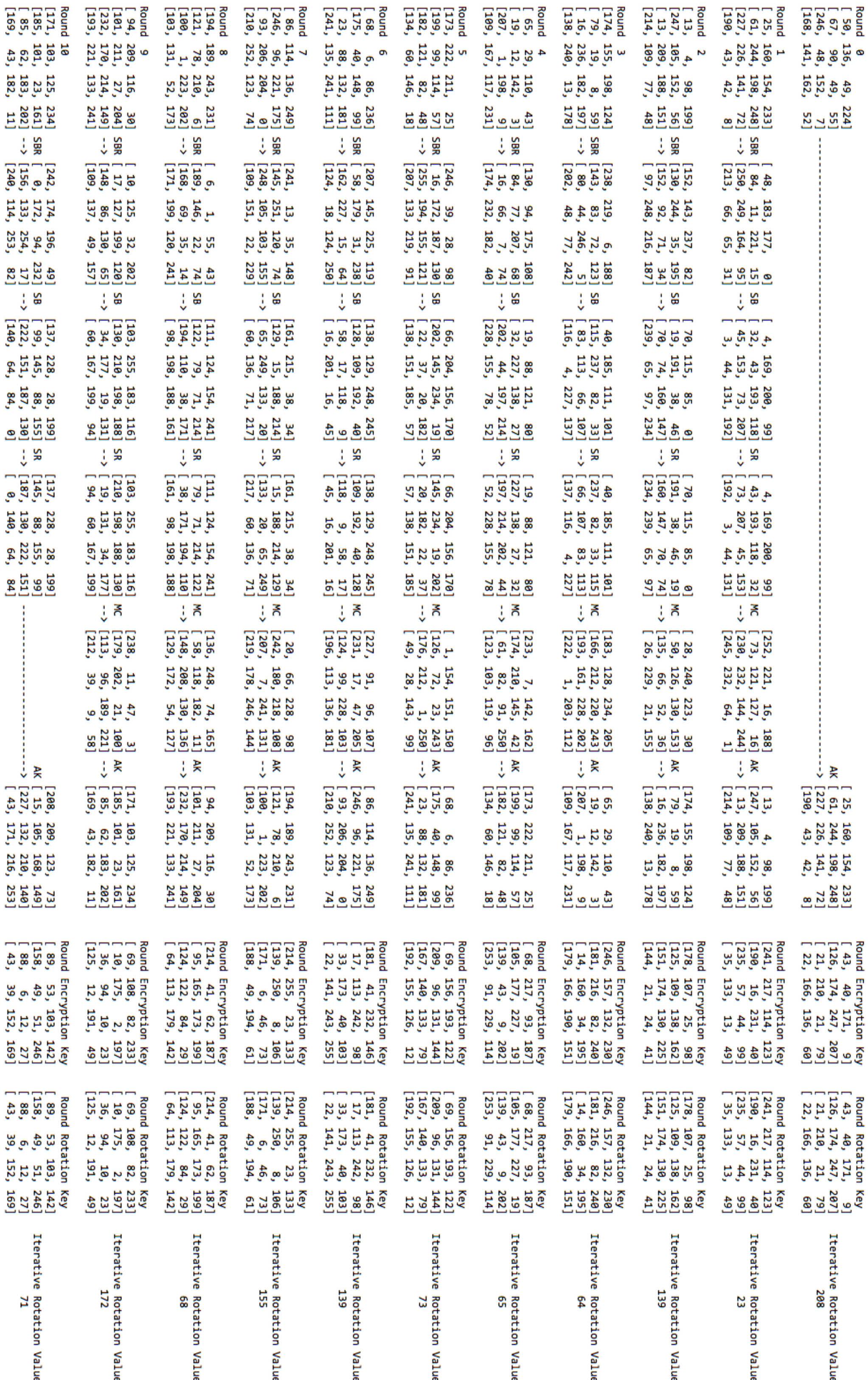

Figure B.3: RAES-128 Type 2 Encryption and Expanded Key of [1] Example Data.

```
Round  0                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 50, 136,  49, 224]                                                                                                                         [ 25, 160, 154, 233]   [ 43,  40, 171,   9]   [ 43,  40, 171,   9]
[ 67,  90,  49,  55]                                                                                                                      AK [ 61, 244, 198, 248]   [126, 174, 247, 207]   [126, 174, 247, 207]   Iterative Rotation Value
[246,  48, 152,   7] -----------------------------------------------------------------------------------------------------------------> [227, 226, 141,  72]   [ 21, 210,  21,  79]   [ 21, 210,  21,  79]            39
[168, 141, 162,  52]                                                                                                                         [190,  43,  42,   8]   [ 22, 166, 136,  60]   [ 22, 166, 136,  60]

Round  1                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 25, 160, 154, 233]     [113, 248, 242,  65]     [163,  65, 137, 131]     [163,  65, 137, 131]     [ 47, 179, 242, 240]     [ 71, 243,  25,  18]   [104,  64, 235, 226]   [241, 217, 114, 123]
[ 61, 244, 198, 248] SBR [149,  76,  30,  80] SB  [ 42,  41, 114,  83] SR  [ 41, 114,  83,  42] MC  [ 81, 217, 254,  95] AK  [ 23,  49, 225, 143]   [ 70, 232,  31, 208]   [190,  16, 231,  40]   Iterative Rotation Value
[227, 226, 141,  72] --> [ 59,  58, 229, 160] --> [226, 128, 217, 224] --> [217, 224, 226, 128] --> [ 72,  33,  42, 135] --> [166,  29,   3, 225]   [238,  60,  41, 102]   [235,  57,  44,  99]            88
[190,  43,  42,   8]     [ 22, 131, 130,  96]     [ 71, 236,  19, 208]     [208,  71, 236,  19]     [181, 223, 242,  18]     [167, 107, 206,  18]   [ 18, 180,  60,   0]   [ 35, 133,  13,  49]

Round  2                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 71, 243,  25,  18]     [200, 116, 154, 147]     [232, 146, 184, 220]     [232, 146, 184, 220]     [ 17,  68,  44, 230]     [ 19,   6, 133, 173]   [  2,  66, 169,  75]   [178, 107,  25,  98]
[ 23,  49, 225, 143] SBR [152, 178,  98,  16] SB  [ 70,  55, 170, 202] SR  [ 55, 170, 202,  70] MC  [187,  12, 182, 201] AK  [160, 255,  90, 245]   [ 27, 243, 236,  60]   [125, 109, 138, 162]   Iterative Rotation Value
[166,  29,   3, 225] --> [ 39, 158, 132,  98] --> [204,  11,  95, 170] --> [ 95, 170, 204,  11] --> [ 30,  43, 184,  27] --> [ 60,  53, 143,  74]   [ 34,  30,  55,  81]   [151, 174, 130, 225]            129
[167, 107, 206,  18]     [ 40, 236,  79, 147]     [ 52, 206, 132, 220]     [220,  52, 206, 132]     [232, 197,  82,  33]     [251,  98, 201, 186]   [ 19, 167, 155, 155]   [144,  21,  24,  41]

Round  3                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 19,   6, 133, 173]     [ 43,  30, 157, 197]     [241, 114,  94, 166]     [241, 114,  94, 166]     [ 27, 243,  36, 248]     [230,  76,  50, 165]   [253, 191,  22,  93]   [246, 157, 132, 230]
[160, 255,  90, 245] SBR [184,  23, 114,  13] SB  [108, 240,  64, 215] SR  [240,  64, 215, 108] MC  [ 91, 106,  81, 184] AK  [252,  62, 233,  60]   [167,  84, 184, 132]   [181, 216,  82, 240]   Iterative Rotation Value
[ 60,  53, 143,  74] --> [ 84,  77, 167,  98] --> [ 32, 227,  92, 170] --> [ 92, 170,  32, 227] --> [125, 250, 188,   4] --> [122, 227, 146, 123]   [  7,  25,  46, 127]   [ 14, 160,  34, 195]            24
[251,  98, 201, 186]     [ 19, 122, 225, 210]     [125, 218, 248, 181]     [181, 125, 218, 248]     [213, 134, 186, 149]     [134, 114, 213,  97]   [ 83, 244, 111, 244]   [179, 166, 190, 151]

Round  4                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[230,  76,  50, 165]     [112, 214, 188,  47]     [ 81, 246, 101,  21]     [ 81, 246, 101,  21]     [244, 220,  79,  21]     [ 99, 244, 113, 118]   [151,  40,  62,  99]   [ 68, 217,  93, 187]
[252,  62, 233,  60] SBR [134, 200, 115, 198] SB  [ 68, 232, 143, 180] SR  [232, 143, 180,  68] MC  [204, 132, 171,  22] AK  [ 79,  83, 196, 253]   [131, 215, 111, 235]   [105, 177, 227,  19]   Iterative Rotation Value
[122, 227, 146, 123] --> [  4, 109,  28,   5] --> [242,  60, 156, 107] --> [156, 107, 242,  60] --> [186, 234, 229,  99] --> [ 18,  91, 122, 131]   [168, 177, 159, 224]   [139,  43,   9, 202]            138
[134, 114, 213,  97]     [ 16, 252,  95, 235]     [202, 176, 207, 233]     [233, 202, 176, 207]     [ 78, 106, 146, 194]     [ 66, 146,   5, 161]   [ 12, 248, 151,  99]   [253,  91, 229, 114]

Round  5                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 99, 244, 113, 118]     [249, 138,   7,  12]     [153, 126, 197, 254]     [153, 126, 197, 254]     [ 91, 144,  24,  34]     [ 21, 246,  64,  25]   [ 78, 102,  88,  59]   [ 69, 156, 193, 122]
[ 79,  83, 196, 253] SBR [229, 233,  90, 147] SB  [217,  30, 190, 220] SR  [ 30, 190, 220, 217] MC  [122,  31,  15, 187] AK  [ 60, 142, 241, 174]   [ 70, 145, 254,  21]   [209,  96, 131, 144]   Iterative Rotation Value
[ 18,  91, 122, 131] --> [168, 241,  16,  25] --> [194, 161, 202, 212] --> [202, 212, 194, 161] --> [189, 208, 218,  66] --> [107, 183,  34,  90]   [214, 103, 248,  24]   [167, 140, 133,  79]            150
[ 66, 146,   5, 161]     [216,  40, 155,  55]     [ 97,  52,  20, 154]     [154,  97,  52,  20]     [ 75,  42,  34,  73]     [ 57, 160,  63,  55]   [114, 138,  29, 126]   [192, 155, 126,  12]

Round  6                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 21, 246,  64,  25]     [170, 139, 213, 174]     [172,  61,   3, 228]     [172,  61,   3, 228]     [203, 226, 221, 240]     [ 78,   1, 102, 112]   [133, 227, 187, 128]   [181,  41, 232, 146]
[ 60, 142, 241, 174] SBR [209,  35, 134,  67] SB  [ 62,  38,  68,  26] SR  [ 38,  68,  26,  62] MC  [ 75,  68,   4, 171] AK  [120, 230,  88, 226]   [ 51, 162,  92,  73]   [ 17, 113, 242,  98]   Iterative Rotation Value
[107, 183,  34,  90] --> [  0,  76, 183, 239] --> [ 99,  41, 169, 223] --> [169, 223,  99,  41] --> [ 30,  90, 126,  80] --> [206, 237,  49,   7]   [208, 183,  79,  87]   [ 33, 173,  40, 103]            149
[ 57, 160,  63,  55]     [206,  53, 212, 204]     [139, 150,  72,  75]     [ 75, 139, 150,  72]     [246, 209,  75, 176]     [ 46, 131,   4, 129]   [216,  82,  79,  49]   [ 22, 141, 243, 255]

Round  7                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 78,   1, 102, 112]     [ 32, 211,  56,  66]     [183, 102,   7,  44]     [183, 102,   7,  44]     [ 87, 174, 158, 199]     [195, 217,  82, 139]   [148, 119, 204,  76]   [214, 255,  23, 133]
[120, 230,  88, 226] SBR [ 74, 184,  42, 180] SB  [214, 108, 229, 141] SR  [108, 229, 141, 214] MC  [ 15, 139, 193, 117] AK  [207, 233, 255,   2]   [192,  98,  62, 119]   [139, 250,   8, 106]   Iterative Rotation Value
[206, 237,  49,   7] --> [160, 191,   3, 217] --> [224,   8, 123,  53] --> [123,  53, 224,   8] --> [  1,  76,  78, 235] --> [187,  65,  12, 254]   [186,  13,  66,  21]   [171,   6,  46,  73]            210
[ 46, 131,   4, 129]     [  0,  85, 214,  83]     [ 99, 252, 246, 237]     [237,  99, 252, 246]     [ 20, 188, 135,  93]     [144, 106,  30, 245]   [132, 214, 153, 168]   [188,  49, 194,  61]

Round  8                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[195, 217,  82, 139]     [ 35,  57, 178, 235]     [ 38,  18,  55, 233]     [ 38,  18,  55, 233]     [173, 186,  80,  55]     [178, 210, 244, 223]   [ 31, 104, 164, 232]   [214,  41,  62, 187]
[207, 233, 255,   2] SBR [ 47,  73,  95,  98] SB  [ 21,  59, 207, 170] SR  [ 59, 207, 170,  21] MC  [ 92, 243, 230, 102] AK  [119, 186, 145, 102]   [ 43,  73, 119,   0]   [ 95, 165, 173, 199]   Iterative Rotation Value
[187,  65,  12, 254] --> [ 27, 161, 108,  94] --> [175,  50,  80,  88] --> [ 80,  88, 175,  50] --> [162, 226,  68, 150] --> [146, 223,  59, 252]   [ 48,  61, 127, 106]   [124, 122,  84,  29]            96
[144, 106,  30, 245]     [240, 202, 126,  85]     [140, 116, 243, 252]     [252, 140, 116, 243]     [226, 162, 180, 250]     [233, 127, 240,  22]   [ 11, 221,  68, 236]   [ 64, 113, 179, 142]

Round  9                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[178, 210, 244, 223]     [ 67,  99, 133, 112]     [ 26, 251, 151,  81]     [ 26, 251, 151,  81]     [237, 196,  97, 175]     [ 37, 100, 101,  67]   [200, 160,   4, 236]   [ 69, 108,  82, 233]
[119, 186, 145, 102] SBR [  8,  75,  34, 247] SB  [ 48, 179, 147, 104] SR  [179, 147, 104,  48] MC  [230, 251, 231, 206] AK  [ 76,  24, 115,  90]   [170, 227, 148, 148]   [ 10, 175,   2, 197]   Iterative Rotation Value
[146, 223,  59, 252] --> [ 35, 112, 204, 141] --> [ 38,  81,  75,  93] --> [ 75,  93,  38,  81] --> [219, 167, 246, 215] --> [150, 215, 249, 178]   [ 77, 112,  15, 101]   [ 36,  94,  10,  23]            145
[233, 127, 240,  22]     [122,  16, 129, 167]     [218, 202,  12,  92]     [ 92, 218, 202,  12]     [110, 119,  99, 138]     [ 19, 215, 135, 130]   [125, 160, 228,   8]   [125,  12, 191,  49]

Round 10                                                                                                                                                                  Round Encryption Key    Round Rotation Key
[ 37, 100, 101,  67]     [ 95, 158, 159, 125]     [207,  11, 219, 255]     [207,  11, 219, 255]                              [219, 191, 107, 163]   [ 20, 180, 176,  92]   [ 89,  53, 103, 142]
[ 76,  24, 115,  90] SBR [134,  82, 173, 148] SB  [ 68,   0, 149,  34] SR  [  0, 149,  34,  68]                           AK [206, 184, 155, 105]   [206,  45, 185,  45]   [158,  49,  51, 246]   Iterative Rotation Value
[150, 215, 249, 178] --> [208,  17,  51, 236] --> [112, 130, 195, 206] --> [195, 206, 112, 130] ---------------------------> [155, 230,  87, 192]   [ 88,  40,  39,  66]   [ 88,   6,  12,  27]            58
[ 19, 215, 135, 130]     [ 77,  17, 193, 188]     [227, 130, 120, 101]     [101, 227, 130, 120]                              [101,  67, 198,  52]   [  0, 160,  68,  76]   [ 43,  39, 152, 169]
```

Figure B.4: RAES-128 Type 3 Encryption and Expanded Keys of [1] Example Data.

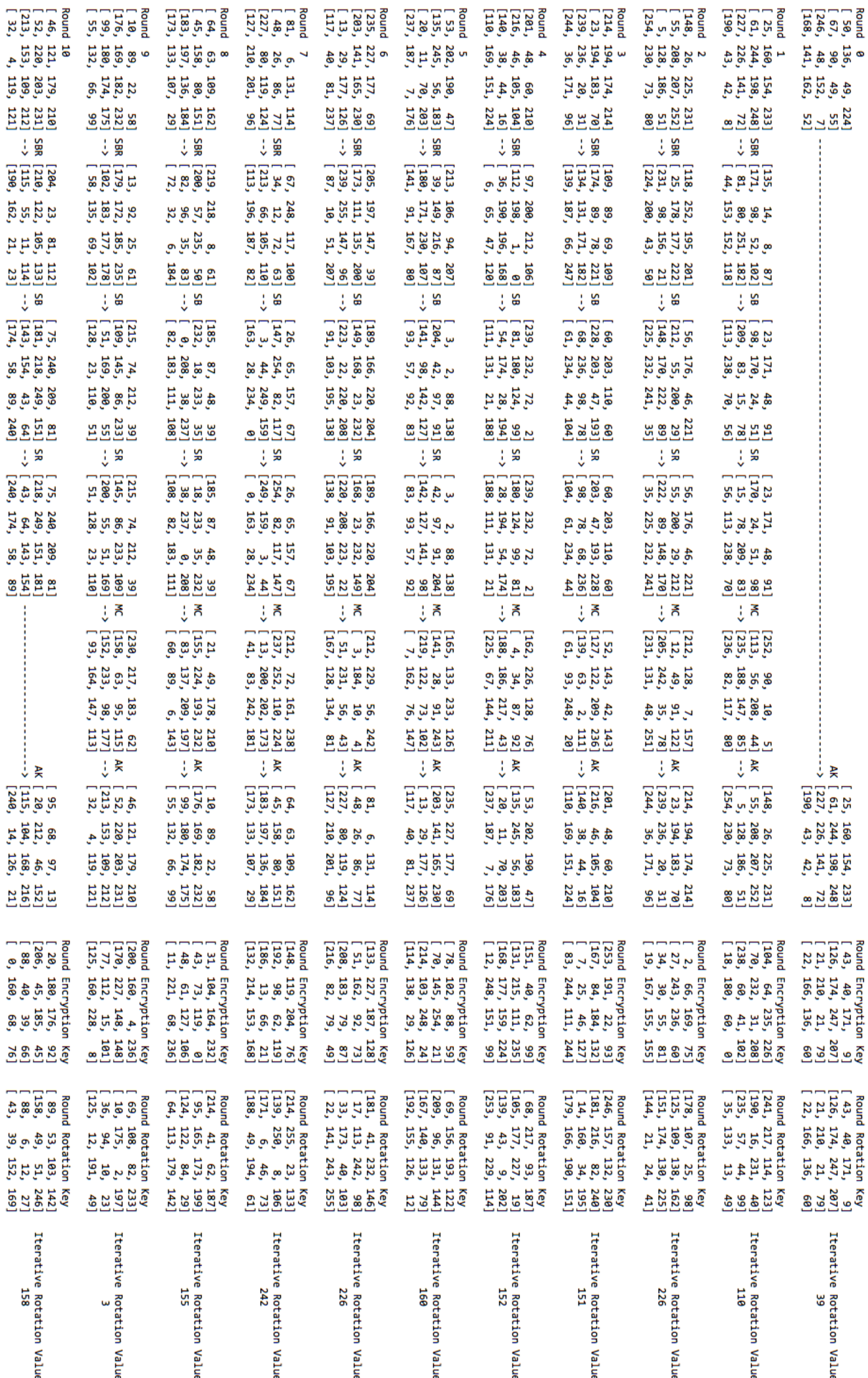

Figure B.5: RAES-128 Type 4 Encryption and Expanded Keys of [1] Example Data.

Bibliography

[1] "Advanced Encryption Standard (AES)". *Federal Information Processing Standards (FIPS) Publication 197*. National Institute of Standards and Technology, 2001.

[2] "Annex A: Approved Security Functions for FIPS PUB 140-2". *Security Requirements for Cryptographic Modules*. National Institute of Standards and Technology, 2014.

[3] "Cryptographic Algorithm Validation Program (CAVP)". National Institute of Standards and Technology (NIST), 1 Aug. 2014. URL http://csrc.nist.gov/groups/STM/cavp/#01.

[4] Bahim, E. and A. Shamir. "Differential Fault Analysis of Secret Key Cryptosystems". *Proceedings of Advances in Cryptology CRYPTO '97*, 1294:513–525, 1997.

[5] Cretu, M. and C. Apostol. "A Modified Version of Rijndael Agorithm Implemented to Analyze the Cyphertexts Correlation for Switched S-Boxes". *Proceedings of 2012 9th International Conference on Communications (COMM)*, 331–334. 2012.

[6] Das, I., S. Roy, S. Nanth, and S. Mondal. "Random S-Box Generation in AES by changing Irreducible Polynomial". *Proceedings of 2012 International Conference on Communications, Devices and Intelligent Systems (CODIS)*, 556–559. 2012.

[7] Dassance, F. and A. Venelli. "Combined Fault and Side-Channel Attacks on the AES Key Schedule". *Proceedings of 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 63–71. 2012.

[8] Davis, J. and A. Martelli. "Scripting Languages Open-source Workable AES (SLOWAES)". Google Project Hosting, 1 March 2014. URL https://code.google.com/p/slowaes/source/browse/trunk/python/aes.py.

[9] Dutertre, J., A. Mirbaha, D. Naccache, A. Ribotta, A. Tira, and R. Vaschalde. "Fault Round Modification Analysis of the Advanced Encryption Standard". *Proceedings of 2012 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 140–145. 2012.

[10] ElBadawy, E. A., W. A. El-Masry, A. Mokhtar, and A. E. S. Hafez. "A New Chaos Advanced Encryption Standard (AES) Algorithm for Data Security". *Proceedings of 2010 International Conference on Signals and Electrical Systems*, 405–408. 2010.

[11] Endo, S., Y. Li, N. Homma, K. Sakiyama, K. Ohta, and T. Aoki. "An Efficient Countermeasure Against Fault Sensitivity Analysis Using Configurable Delay Blocks". *Proceedings of 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 95–102. 2012.

[12] Feistel, H. "Cryptography and Computer Privacy". *Scientific America*, 228:15–23, 1973.

[13] Floissac, N. and Y. L'Hyver. "From AES-128 to AES-192 to AES-256, How to Adapt Differential Fault Analysis Attacks on KeyExpansion". *Proceedings of 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 43–53. 2011.

[14] Fukunaga, T. and J. Takahashi. "Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers". *Proceedings of 2009 International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 84–92. 2009.

[15] Juremi, J. "A Proposal for Improving AES S-Box with Rotation and Key-Dependent". *Proceedings of 2012 International Conference on Cyber Security, Cyber Warefare and Digital Forensic (CyberSec)*, 38–42. 2012.

[16] Kim, C. H. "Differential Fault Analysis of AES: Toward Reducing Number of Faults". *Journal of Information Sciences*, 199:43–57, 2012.

[17] Kim, C. H. "Improved Differential Fault Analysis on AES Key Schedule". *IEEE Transactions on Information Forensics and Security*, 7:41–50, 2012.

[18] Lashermes, R., G. Reymond, J. Dutertre, J. Fournier, B. Robisson, and A. Tria. "A DFA on AES Based on the Entropy of Error Distributions". *Proceedings of 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 34–43. 2012.

[19] Li, W., D. Gu, Y. Wang, J. Li, and Z. Liu. "An Extension of Differential Fault Analysis on AES". *Proceedings of International Conference on Network and System Security*, 443–446. 2009.

[20] Lomne, V., T. Roche, and A. Thillard. "On the Need of Randomness in Fault Attack Countermeasures - Application to AES". *Proceedings of 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 85–94. 2012.

[21] Luan, H., L. Luo, and Y. Wang. "An S-box Construction Algorithm based on Spatiotemporal Chaos". *Proceedings of 2010 International Conference on Communications and Mobile Computing*, 61–65. 2010.

[22] N., K. G. and V. Ramanswarmy. "Making AES Stronger: AES with Key Dependent S-Box". *IJCSNS International Journal of Computer Science and Network Security*, 8:388–398, 2008.

[23] Nassar, M., Y. Souissi, S. Guilley, and J.L. Danger. "RSM: A Small and Fast Countermeasure for AES, Secure Against 1st and 2nd-Order Zero-Offset SCAs". *Proceedings of 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1173–1179. 2012.

[24] Park, J., S. Moon, D. Choi, Y. Kang, and J. Ha. "Fault Attack for the Iterative Operation of AES S-Box". *Proceedings of 2010 International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 550–555. 2010.

[25] Radhakrishnan, S. V. and S. Subramanian. "An Analytical Approach to S-box Generation". *Proceedings of 2012 International Conference on Communications and Signal Processing (ICCSP)*, 1–5. 2012.

[26] Sakiyama, K., Y. Li, M. Iwamoto, and K. Ohta. "Information-Theoretic Approach to Optimal Differential Fault Analsysis". *IEEE Transactions on Information Forensics and Security*, 7:109–120, 2012.

[27] Stanoyevitch, A. *Introduction to Cryptography with Mathematical Foundations and Computer Implementations*. Chapman and Hall/CRC, Taylor and Francis Group, Boca Raton, FL, USA, 2011.

[28] Tunstall, M., D. Mukhapadhyay, and S. Ali. "Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault". *2011 International Federation for Information Processing*, 6633:224–233, 2011.

[29] Vervaudwhede, I., D. Karaklacjic, and J. Schmidt. "The Fault Attack Jungle - A Classification Model to Guide You". *Proceedings of 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 3–8. 2011.

[30] Zaibi, G., A. Kachouri, F. Peyrard, and D. Fournier-Prunaret. "On Dynamic Chaotic S-box". *Proceedings of 2009 Global Information Infrastucture Symposium*, 1–5. 2009.

[31] Zaibi, G., A. Kachouri, F. Peyrard, and D. Fournier-Prunaret. "A New Design of Dynamic S-box based on Two Chaotic Maps". *Proceedings of 2010 International Conference on Computer Systems and Applications (AICCSA)*, 1–6. 2010.

[32] Zhao, G., H. Yan, and F. Lu. "Research of Changeable S-Box in Block Cryptosystem Based on Chaos". *Proceedings of 2007 International Conference on Communications, Circuits and Systems*, 436–441. 2007.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
18–09–2014	Master's Thesis	Oct 2012–Sept 2014

4. TITLE AND SUBTITLE

Extending Differential Fault Analysis to Dynamic S-Box Advanced Encryption Standard Implementations

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Flamm, Bradley M., Civilian

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-T-14-S-08

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AF Cyberspace Technical Center of Excellence
Attn: Dr. Robert Mills
2950 Hobson Way
WPAFB, OH 45433-7765
(937) 255-3636, Ext. 4738, Robert.Mills@afit.edu

10. SPONSOR/MONITOR'S ACRONYM(S)

AF CyTCoE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

AES is a worldwide cryptographic standard for symmetric key cryptography. Many attacks try to exploit inherent weaknesses in the algorithm or use side channels to reduce entropy. At the same time, researchers strive to enhance AES and mitigate these growing threats. This paper researches the extension of existing DFA attacks, a family of side channel attacks, on standard AES to Dynamic S-box AES research implementations. Theoretical analysis reveals an expected average keyspace reduction of $2^{-88.9323}$ after one faulty ciphertext using DFA on the State of Rotational S-box AES-128 implementations. Experimental results revealed an average $2^{-88.8307}$ keyspace reduction and confirmed full key recovery is possible.

15. SUBJECT TERMS

Advanced Encryption Standard, AES, Dynamic S-box, Rotational S-box, RAES, Differential Fault Analysis, DFA

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Thomas E. Dube (ENG)
U	U	U	UU	107	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4613 thomas.dube@afit.edu