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

Bradley M. Flamm

Follow this and additional works at: https://scholar.afit.edu/etd

## Recommended Citation

Flamm, Bradley M., "Extending Differential Fault Analysis to Dynamic S-Box Advanced Encryption Standard Implementations"
(2014). Theses and Dissertations. 564.
https://scholar.afit.edu/etd/564


# 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

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

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

## THESIS

Presented to the Faculty<br>Department of Electrical and Computer Engineering Graduate School of Engineering and Management Air Force Institute of Technology<br>Air University<br>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

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

Bradley M. Flamm, B.S.M.

Civilian

## Approved:

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

2 Sep 2014
Date

2 Sep 2014
Date

2 Sep 2014
Date


#### 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
Abstract ..... iv
Table of Contents ..... v
List of Figures ..... viii
List of Acronyms ..... xi
I. Introduction ..... 1
1.1 Motivation ..... 1
1.2 Research Objectives ..... 1
1.3 Scope and Limitations ..... 3
1.4 Approach ..... 3
1.5 Thesis Organization ..... 3
II. Background ..... 4
2.1 Cryptology ..... 4
2.1.1 Properties ..... 5
2.1.2 Attacks ..... 5
2.1.3 Algorithms ..... 6
2.2 AES ..... 7
2.2.1 Galois Field $2^{8}$ ..... 8
2.2.2 State Operations ..... 11
2.2.3 Encryption ..... 13
2.2.4 Decryption ..... 14
2.2.5 Key Expansion Algorithm ..... 16
2.3 Dynamic S-box ..... 19
2.3.1 Rotational S-box ..... 19
2.3.2 Chaotic S-box ..... 21
2.3.3 Reduction S-box ..... 21
2.3.4 Switch S-box ..... 21
2.4 Brute Force Attacks ..... 21
2.4.1 Time/Memory Trade-off ..... 22
2.4.2 Brute Force Mitigation Techniques ..... 22
2.5 Differential Fault Analysis ..... 24
2.5.1 DFA on the State ..... 25

## Page

2.5.2 DFA on the Key Schedule ..... 26
2.5.3 Round Modification Analysis ..... 27
2.5.4 DFA on the Algorithm ..... 28
2.5.5 DFA Mitigation Techniques ..... 28
2.6 Background Summary ..... 28
III. Theoretical Attack Analysis ..... 29
3.1 Problem Definition ..... 29
3.1.1 Goals and Hypothesis ..... 29
3.1.2 Approach ..... 29
3.2 Attack Targets and Sources ..... 30
3.2.1 Potential Attack Targets ..... 30
3.2.2 Attack Target Implementation ..... 31
3.2.3 Potential Attack Sources ..... 34
3.2.4 Attack Source Implementation ..... 35
3.3 Attack Analysis ..... 41
3.3.1 Rotation Step Analysis ..... 41
3.3.2 Mapping Rotate to an Operation in $\operatorname{GF}\left(2^{8}\right)$ ..... 44
3.3.3 Alternate Attack Analysis on Standard AES ..... 44
3.3.4 General Extension ..... 46
3.3.5 Reversing the Key Schedule ..... 47
3.4 Theoretical Attack Summary ..... 48
IV. Methodology ..... 49
4.1 Problem Definition ..... 49
4.1.1 Goals and Hypothesis ..... 49
4.1.2 Approach ..... 49
4.2 System Boundaries ..... 50
4.3 System Services ..... 51
4.4 Workload ..... 51
4.5 Performance Metrics ..... 51
4.6 System Parameters ..... 52
4.7 Factors ..... 52
4.8 Evaluation Technique ..... 53
4.9 Experimental Design ..... 54
4.10 Methodology Summary ..... 55
V. Analysis of Experimental Attack Results ..... 56
5.1 Existing Attack ..... 56

## Page

5.2 Attack Extension ..... 62
5.3 Design Suggestions ..... 71
5.4 Analysis Summary ..... 73
VI. Conclusion ..... 74
6.1 Impact ..... 74
6.2 Contributions ..... 74
6.3 Future Work ..... 75
Appendix A: Discussion of Rotational S-box Design Decisions ..... 77
Appendix B: RAES Validation Data ..... 87
Bibliography ..... 92

## List of Figures

Figure Page
2.1 High-Level Encryption and Decryption. ..... 4
2.2 AES Pseudocode Based on [8]. ..... 7
2.3 Generic AES State Representation with Byte Indexing. ..... 8
2.4 AES S-box. ..... 11
2.5 AES Shift Row operation. ..... 12
2.6 Mix Column Example. ..... 12
2.7 Generic Add Key Step. ..... 13
2.8 Logical AES-128 Encryption. ..... 14
2.9 AES-128 Encryption. ..... 15
2.10 Logical AES-128 Decryption. ..... 16
2.11 AES-128 Key Schedule. ..... 18
2.12 Reversal of the AES-128 Key Schedule. ..... 20
2.13 DFA on the State Fault Propagation in AES-128 [16]. ..... 25
2.14 DFA on the Key Schedule Fault Propagation in AES-128 [17]. ..... 26
2.15 DFA on the Key Schedule Fault Propagation in AES-128 [17]. ..... 27
3.1 Logical RAES-128 Encryption. ..... 32
3.2 Logical RAES-128 Decryption ..... 34
3.3 XOR of Correct and Faulty AES-128 Encryption, Rounds 8-10. ..... 36
3.4 XOR of Correct and Faulty AES-128 MC Operation. ..... 37
3.5 XOR MC Walkthrough. ..... 37
3.6 XOR of Correct and Faulty AES-128 AK Operation. ..... 38
3.7 XOR of Correct and Faulty AES-128 SB Operation. ..... 38
3.8 XOR of Correct and Faulty AES-128 SR Operation. ..... 39
Figure Page
3.9 RAES S-box ${ }_{0}$. ..... 42
3.10 RAES S-box ${ }_{1}$ ..... 42
3.11 RAES S-box 2 . ..... 42
3.12 XOR of Correct and Faulty RAES-128 Encryption, Rounds 8-10. ..... 45
3.13 XOR of Correct and Faulty AES-128 Encryption, Round 10. ..... 45
4.1 System Boundaries. ..... 50
4.2 Factors and Levels. ..... 53
5.1 Histogram of AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 57
5.2 Boxplot of AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 57
5.3 Quartiles of AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 58
5.4 Boxplot of $\log _{2}$ Transformed AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 58
5.5 Histogram of $\log _{2}$ Transformed AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 59
5.6 Histogram of AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 2 Faulty Ciphertext Round 10 Reduction. ..... 61
5.7 Histogram of AES-128 DFA Runtime. ..... 62
5.8 Boxplots of $\log _{2}$ Transformed RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 63
5.9 Tukey Multiple Comparisons of Means Test of RAES-128 Types, 1 Faulty Ciphertext Round 10 Reduction. ..... 63
5.10 Histogram of RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction ..... 64
5.11 Quartiles of RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction ..... 65
5.12 Histogram of $\log _{2}$ Transformed RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction. ..... 65
5.13 Histogram of Remaining $\mathrm{R}_{10}$, 2 Faulty Ciphertext Round 10 Reduction. ..... 68
Figure Page
5.14 Histogram of $\log _{2}$ Transformed Observed/2 RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 2 Faulty Ciphertext Round 10 Reduction. ..... 69
5.15 Histogram of Observed/2 RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 3 Faulty Ciphertext Round 10 Reduction. ..... 69
5.16 Histogram of Required Cipher Pairs for Full Key Recovery on RAES-128. ..... 70
5.17 Histogram of RAES-128 DFA Runtime. ..... 71
A. 1 AES-KDS Validation Encryptions [22]. ..... 84
B. 1 AES-128 Encryption and Expanded Key of [1] Example Data. ..... 87
B. 2 RAES-128 Type 1 Encryption and Expanded Key of [1] Example Data. ..... 88
B. 3 RAES-128 Type 2 Encryption and Expanded Key of [1] Example Data. ..... 89
B. 4 RAES-128 Type 3 Encryption and Expanded Keys of [1] Example Data. ..... 90
B. 5 RAES-128 Type 4 Encryption and Expanded Keys of [1] Example Data. ..... 91

## 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 highlevel 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 targetsource 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 AES128 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: ${ }^{i} S_{k}^{j}$. The round index is $i, j$ is the operation index and $k$ is the byte index: ${ }^{\text {round }} S_{\text {byte }}^{\text {operation }}$. A 4 x 4 matrix of bytes represents each particular state ${ }^{i} \mathrm{~S}^{j}$ with $k \in\{0,1, \ldots, 15\}$ indexed as shown below in Figure 2.3. Alternatively, $k \in\{\mathrm{R} 0, \mathrm{R} 1, \mathrm{R} 2, \mathrm{R} 3\}$ or $k \in\{\mathrm{C} 0$, $\mathrm{C} 1, \mathrm{C} 2, \mathrm{C} 3\}$ 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.

| $\mathrm{S}_{0}$ | $\mathrm{~S}_{1}$ | $\mathrm{~S}_{2}$ | $\mathrm{~S}_{3}$ |
| :---: | :---: | :---: | :---: |
| $\mathrm{~S}_{4}$ | $\mathrm{~S}_{5}$ | $\mathrm{~S}_{6}$ | $\mathrm{~S}_{7}$ |
| $\mathrm{~S}_{8}$ | $\mathrm{~S}_{9}$ | $\mathrm{~S}_{10}$ | $\mathrm{~S}_{11}$ |
| $\mathrm{~S}_{12}$ | $\mathrm{~S}_{13}$ | $\mathrm{~S}_{14}$ | $\mathrm{~S}_{15}$ |

Figure 2.3: Generic AES State Representation with Byte Indexing.

### 2.2.1 Galois Field $2^{8}$.

AES uses the Galois Field $\mathrm{GF}\left(2^{8}\right)$, a number system, for mathematical manipulations of bytes, treating them as polynomials. $\mathrm{GF}\left(2^{8}\right)$ provides unique properties for calculation of all bit manipulations, hexadecimal notation simply improves portability and ease of storage. Each bit in the byte $b_{7} b_{6} b_{5} b_{4} b_{3} b_{2} b_{1} b_{0}$, where $b_{i}$ is the $i^{\text {th }}$ bit, represents the coefficient of
the $x^{i}$ term of the polynomial

$$
b_{7} x^{7}+b_{6} x^{6}+b_{5} x^{5}+b_{4} x^{4}+b_{3} x^{3}+b_{2} x^{2}+b_{1} x^{1}+b_{0} x^{0} .
$$

For example,

$$
0 \mathrm{xA} 4=10100100=1 x^{7}+0 x^{6}+1 x^{5}+0 x^{4}+0 x^{3}+1 x^{2}+0 x^{1}+0 x^{0}=x^{7}+x^{5}+x^{2}
$$

The base 2 in $\mathrm{GF}\left(2^{8}\right)$ represents that coefficients are in modulus two. The following example highlights addition.

$$
\begin{gathered}
0 \mathrm{xA} 4+0 \mathrm{x} 86=10100100+10000110 \\
=\left(1 x^{7}+0 x^{6}+1 x^{5}+0 x^{4}+0 x^{3}+1 x^{2}+0 x^{1}+0 x^{0}\right)+\left(1 x^{7}+0 x^{6}+0 x^{5}+0 x^{4}+0 x^{3}+1 x^{2}+1 x^{1}+0 x^{0}\right) \\
=\left(x^{7}+x^{5}+x^{2}\right)+\left(x^{7}+x^{2}+x\right) \\
=2 x^{7}+x^{5}+2 x^{2}+x \\
=x^{5}+x=00100010=0 \mathrm{x} 22
\end{gathered}
$$

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

$$
10100100
$$

$\oplus \underline{10000110}$

$$
00100010=0 \times 22 .
$$

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

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

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 $\operatorname{GF}\left(2^{8}\right)$. Supposing $\beta$ is some unknown byte with the relation, $\beta+0 \times 14=0 \times 96$, it is now possible to solve for $\beta$,

$$
\beta+0 \times 14+0 \times 14=0 \times 96+0 \times 14 \Rightarrow \beta=0 \times 82
$$

Additionally, it is impossible to add together any two numbers within $\operatorname{GF}\left(2^{8}\right)$ and end up with a number outside of $\operatorname{GF}\left(2^{8}\right)$. Because of this, $\operatorname{GF}\left(2^{8}\right)$ is said to be closed under addition. Further, addition in $\mathrm{GF}\left(2^{8}\right)$ 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 $\operatorname{GF}\left(2^{8}\right)$. If multiplication achieves a power of $x$ greater than or equal to eight, a reduction occurs using the irreducible polynomial for $\mathrm{GF}\left(2^{8}\right), x^{8}+x^{4}+x^{3}+x+1$. This polynomial enables construction of $\operatorname{GF}\left(2^{8}\right)$, 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.

$$
\begin{gathered}
0 \mathrm{xA} 4 \cdot 0 \mathrm{x} 02=10100100 \cdot 00000010 \\
=\left(1 x^{7}+0 x^{6}+1 x^{5}+0 x^{4}+0 x^{3}+1 x^{2}+0 x^{1}+0 x^{0}\right) \cdot\left(0 x^{7}+0 x^{6}+0 x^{5}+0 x^{4}+0 x^{3}+0 x^{2}+1 x^{1}+0 x^{0}\right) \\
=\left(x^{7}+x^{5}+x^{2}\right) \cdot x \\
=x^{8}+x^{6}+x^{3} \\
=\left(x^{4}+x^{3}+x+1\right)+x^{6}+x^{3} \\
=x^{6}+x^{4}+2 x^{3}+x+1
\end{gathered}
$$

$$
=x^{6}+x^{4}+x+1=01010011=0 \times 53
$$

Because of the relation $x^{8}=x^{4}+x^{3}+x+1, \operatorname{GF}\left(2^{8}\right)$ is also closed under multiplication. Multiplying the reduction by an appropriate power of $x$ reduces powers greater than 8. For example, $x^{11}=x^{3} \cdot x^{8}=x^{3}\left(x^{4}+x^{3}+x+1\right)=x^{7}+x^{6}+x^{4}+x^{3}$. 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 $\alpha$ 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 $(S-b o x(a) \neq a)$ 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 $\operatorname{S-box}(0 \times 12)=0 x C 9$.

|  | x 0 | x 1 | x 2 | x 3 | x 4 | x 5 | x 6 | x 7 | x 8 | x 9 | xa | xb | xc | xd | xe | xf |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0 x | 63 | 7 c | 77 | 7 b | f 2 | 6 b | 6 f | c 5 | 30 | 01 | 67 | 2 b | fe | d 7 | ab | 76 |
| 1 x | ca | 82 | c 9 | 7 d | fa | 59 | 47 | $\mathrm{f0}$ | ad | d 4 | a 2 | af | 9 c | a 4 | 72 | c 0 |
| 2 x | b 7 | fd | 93 | 26 | 36 | 3 f | $\mathrm{f7}$ | cc | 34 | a 5 | e 5 | f 1 | 71 | d 8 | 31 | 15 |
| 3 x | 04 | c 7 | 23 | c 3 | 18 | 96 | 05 | 9 a | 07 | 12 | 80 | e 2 | eb | 27 | b 2 | 75 |
| 4 x | 09 | 83 | 2 c | 1 a | 1 b | 6 e | 5 a | a 0 | 52 | 3 b | d 6 | b 3 | 29 | e 3 | 2 f | 84 |
| 5 x | 53 | d 1 | 00 | ed | 20 | fc | b 1 | 5 b | 6 a | cb | be | 39 | 4 a | 4 c | 58 | cf |
| 6 x | d 0 | ef | aa | fb | 43 | 4 d | 33 | 85 | 45 | f 9 | 02 | 7 f | 50 | 3 c | 9 f | a 8 |
| 7 x | 51 | a 3 | 40 | 8 f | 92 | 9 d | 38 | $\mathrm{f5}$ | bc | b 6 | da | 21 | 10 | ff | f 3 | d 2 |
| 8 x | cd | 0 c | 13 | ec | 5 f | 97 | 44 | 17 | c 4 | a 7 | 7 e | 3 d | 64 | 5 d | 19 | 73 |
| 9 x | 60 | 81 | 4 f | dc | 22 | 2 a | 90 | 88 | 46 | ee | b 8 | 14 | de | 5 e | 0 b | db |
| ax | e 0 | 32 | 3 a | 0 a | 49 | 06 | 24 | 5 c | c 2 | d 3 | ac | 62 | 91 | 95 | e 4 | 79 |
| bx | e 7 | c 8 | 37 | 6 d | 8 d | d 5 | 4 e | a 9 | 6 c | 56 | $\mathrm{f4}$ | ea | 65 | 7 a | ae | 08 |
| cx | ba | 78 | 25 | 2 e | 1 c | a 6 | b 4 | c 6 | e 8 | dd | 74 | 1 f | 4 b | bd | 8 b | 8 a |
| dx | 70 | 3 e | b 5 | 66 | 48 | 03 | f 6 | 0 e | 61 | 35 | 57 | b 9 | 86 | c 1 | 1 d | 9 e |
| ex | e 1 | f 8 | 98 | 11 | 69 | d 9 | 8 e | 94 | $9 b$ | 1 e | 87 | e 9 | ce | 55 | 28 | df |
| fx | 8 c | a 1 | 89 | 0 d | bf | e 6 | 42 | 68 | 41 | 99 | 2 d | 0 f | b 0 | 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 $i^{\text {th }}$ 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, $S_{C i}$ for $i \in\{0,1,2,3\}$. Figure 2.6 shows the multiplication of this fixed matrix with the first column of the state, $M \times S_{C 0}$. Multiplication and addition are as defined in $\operatorname{GF}\left(2^{8}\right)$.

| 2 | 3 | 1 | 1 | $\times$ | $\mathrm{S}_{0}$ | $=$ | $2 \mathrm{~S}_{0}+3 \mathrm{~S}_{4}+\mathrm{S}_{8}+\mathrm{S}_{12}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 2 | 3 | 1 |  | $\mathrm{S}_{4}$ |  | $\mathrm{S}_{0}+2 \mathrm{~S}_{4}+3 \mathrm{~S}_{8}+\mathrm{S}_{12}$ |
| 1 | 1 | 2 | 3 |  | $\mathrm{S}_{8}$ |  | $\mathrm{S}_{0}+\mathrm{S}_{4}+2 \mathrm{~S}_{8}+3 \mathrm{~S}_{12}$ |
| 3 | 1 | 1 | 2 |  | $\mathrm{S}_{12}$ |  | $3 \mathrm{~S}_{0}+\mathrm{S}_{4}+\mathrm{S}_{8}+2 \mathrm{~S}_{12}$ |

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.

| $\mathrm{S}_{0}$ | $\mathrm{~S}_{1}$ | $\mathrm{~S}_{2}$ | $\mathrm{~S}_{3}$ |
| :--- | :--- | :--- | :--- |
| $\mathrm{~S}_{4}$ | $\mathrm{~S}_{5}$ | $\mathrm{~S}_{6}$ | $\mathrm{~S}_{7}$ |
| $\mathrm{~S}_{8}$ | $\mathrm{~S}_{9}$ | $\mathrm{~S}_{10}$ | $\mathrm{~S}_{11}$ |
| $\mathrm{~S}_{12}$ | $\mathrm{~S}_{13}$ | $\mathrm{~S}_{14}$ | $\mathrm{~S}_{15}$ |$+$| $\mathrm{K}_{0}$ | $\mathrm{~K}_{1}$ | $\mathrm{~K}_{2}$ | $\mathrm{~K}_{3}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $\mathrm{~K}_{4}$ | $\mathrm{~K}_{5}$ | $\mathrm{~K}_{6}$ | $\mathrm{~K}_{7}$ |
| $\mathrm{~K}_{8}$ | $\mathrm{~K}_{9}$ | $\mathrm{~K}_{10}$ | $\mathrm{~K}_{11}$ |
| $\mathrm{~K}_{12}$ | $\mathrm{~K}_{13}$ | $\mathrm{~K}_{14}$ | $\mathrm{~K}_{15}$ |
| $\mathrm{~S}_{0} \oplus \mathrm{~K}_{0}$ | $\mathrm{~S}_{1} \oplus \mathrm{~K}_{1}$ | $\mathrm{~S}_{2} \oplus \mathrm{~K}_{2}$ | $\mathrm{~S}_{3} \oplus \mathrm{~K}_{3}$ |
| $\mathrm{~S}_{4} \oplus \mathrm{~K}_{4}$ | $\mathrm{~S}_{5} \oplus \mathrm{~K}_{5}$ | $\mathrm{~S}_{6} \oplus \mathrm{~K}_{6}$ | $\mathrm{~S}_{7} \oplus \mathrm{~K}_{7}$ |
| $\mathrm{~S}_{8} \oplus \mathrm{~K}_{8}$ | $\mathrm{~S}_{9} \oplus \mathrm{~K}_{9}$ | $\mathrm{~S}_{10} \oplus \mathrm{~K}_{10}$ | $\mathrm{~S}_{11} \oplus \mathrm{~K}_{11}$ |
| $\mathrm{~S}_{12} \oplus \mathrm{~K}_{12}$ | $\mathrm{~S}_{13} \oplus \mathrm{~K}_{13}$ | $\mathrm{~S}_{14} \oplus \mathrm{~K}_{14}$ | $\mathrm{~S}_{15} \oplus \mathrm{~K}_{15}$ |

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 $\left(\mathrm{SB}^{-1}\right)$, InverseShiftRows $\left(\mathrm{SR}^{-1}\right)$, InverseMixColumns $\left(\mathrm{MC}^{-1}\right)$ and AddRoundKey (AK). Figure 2.10 shows the logical flow of decryption, bottom to top.

- $\mathbf{S B}^{-1}$. The inverse S-box reverses the lookup process.
- $\mathbf{S R}^{-1}$. The $i^{\text {th }}$ row rotates $i$ bytes to the right, $i \in[0,3]$.
- $\mathbf{M C}{ }^{-1}$. Matrix multiplication with the inverse of the constant matrix used in MC.
- AK. XOR is its own inverse, thus $\mathrm{AK}^{-1}=\mathrm{AK}$, 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 $4 \times 4$ representation into a $4 \times 44$, or $114 \times 4$ round keys. Similarly, AES-192 expands $4 \times 6$ to $4 \times 52$, and AES- 256 from $4 x 8$ to $4 \times 60$. 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 $E F)=A B D C E F 01)$.
- 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 $\mathrm{GF}\left(2^{8}\right)$, rcon(i) $=\mathrm{x}^{i-1}$. Rounds 1-10 use an rcon value. Table 2.1 shows the computation of each of these values.

| $\operatorname{rcon}(1)=x^{0}$ | 1 | $=00000001$ |
| ---: | :--- | :--- | :--- |
| $\operatorname{rcon}(2)=x^{1}=$ | $x$ | $=00000010$ |
| $\operatorname{rcon}(3)=x^{2}=$ | $x^{2}$ | $=00000100$ |
| $\operatorname{rcon}(4)=x^{3}=$ | $x^{3}$ | $=00001000$ |
| $\operatorname{rcon}(5)=x^{4}=$ | $x^{4}$ | $=00010000$ |
| $\operatorname{rcon}(6)=x^{5}=$ | $x^{5}$ | $=00100000$ |
| $\operatorname{rcon}(7)=x^{6}=$ | $x^{6}$ | $=01000000$ |
| $\operatorname{rcon}(8)=x^{7}=$ | $x^{7}$ | $=0000000$ |
| $\operatorname{rcon}(9)=x^{8}=x^{4}+x^{3}+x+1$ | $=00011011$ |  |
| $\operatorname{rcon}(10)$ | $=x^{9}=x^{5}+x^{4}+x^{2}+x$ | $=00110110$ |

Table 2.1: Calculation of Rcon Values 1-10.

By convention, $W$ is the expanded round key matrix with $W[i][j]$ denoting the $i^{\text {th }}$ column [0-43], $j^{\text {th }}$ byte [0-3]. As with storing the plaintext in ${ }^{0} \mathrm{~S}^{0}$, 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 $\operatorname{SW}(\operatorname{RW}(W[i-1])) \oplus$ $[\operatorname{rcon}(r), 0,0,0]$. From this, round $i$ key ${ }^{i} K=[W(4 i)-W(4 i+3)]$. Figure 2.11 depicts the AES-128 key schedule.

$$
\text { Encryption Key }=\mathrm{K}_{0} \mathrm{~K}_{1} \mathrm{~K}_{2} \mathrm{~K}_{3} \mathrm{~K}_{4} \mathrm{~K}_{5} \mathrm{~K}_{6} \mathrm{~K}_{7} \mathrm{~K}_{8} \mathrm{~K}_{9} \mathrm{~K}_{10} \mathrm{~K}_{11} \mathrm{~K}_{12} \mathrm{~K}_{13} \mathrm{~K}_{14} \mathrm{~K}_{15}
$$



Figure 2.11: AES-128 Key Schedule.

Knowing the last round key, ${ }^{10} \mathrm{~K}$, 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 $\mathrm{K}_{10}$ 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 $\mathrm{GF}\left(2^{8}\right)$ 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 Sboxes 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 $\operatorname{GF}\left(2^{8}\right)$, 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 $\operatorname{GF}\left(2^{8}\right), 30$ 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 $\mathrm{E}(\mathrm{P})=\mathrm{C}$, 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:

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

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

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

- 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 $1 \times 10^{-15}$ seconds and a target algorithm that has $2^{80}$ keys, recovery requires a maximum of:

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

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:

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

### 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 ${ }^{7} \mathrm{~S}_{0}^{2}$. 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 $r^{\text {th }}$ 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 $r^{\text {th }}$ 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 tradeoffs 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 $\mathrm{SBR}^{-1}$ 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 ${ }^{8} \mathrm{~S}_{0}^{2}$, obtaining a faulty encryption $\overline{\mathrm{E}}$ of plaintext P , using key K . The single byte fault propagates to corrupt the entire ciphertext. $\overline{\mathrm{S}}$ and $\overline{\mathrm{C}}$ represent the faulty state and ciphertext. Figure 3.3 represents the XOR of the last three
rounds of E and $\overline{\mathrm{E}} . \Delta \mathrm{S}$ and $\Delta \mathrm{C}$ 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 $\operatorname{GF}\left(2^{8}\right)$,

$$
\operatorname{MC}\left({ }^{8} S^{2}\right) \oplus \operatorname{MC}\left({ }^{8} \bar{S}^{2}\right)=\operatorname{MC}\left({ }^{8} S^{2} \oplus{ }^{8} \bar{S}^{2}\right)=\operatorname{MC}\left({ }^{8} \Delta S^{2}\right)
$$

| ${ }^{8} \Delta \mathrm{~S}^{2}$ |  |  |  |  | ${ }^{8} \Delta \mathrm{~S}^{3}$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| a | 0 | 0 | 0 |  | 2a | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | MC | a | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |  | a | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |  | 3 a | 0 | 0 | 0 |

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


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 $\overline{\mathrm{E}}$. Thus, as shown in Figure 3.6, every byte of ${ }^{8} \Delta \mathrm{~K}$ is 0 . Because XOR, or addition in $\operatorname{GF}\left(2^{8}\right)$ is commutative, the order of performing this XOR does not matter, and so ${ }^{8} \Delta \mathrm{~S}^{4}=\mathrm{AK}\left({ }^{8} \Delta \mathrm{~S}^{3}\right)={ }^{8} \Delta \mathrm{~S}^{3} \oplus{ }^{8} \Delta \mathrm{~K}={ }^{8} \Delta \mathrm{~S}^{3}$. The equations below demonstrates this relationship.

$$
\begin{aligned}
{ }^{8} \Delta \mathrm{~S}_{0}^{4} & =\left({ }^{8} \mathrm{~S}_{0}^{3} \oplus{ }^{8} \mathrm{~K}_{0}\right) \oplus\left({ }^{8} \overline{\mathrm{~S}}_{0}^{3} \oplus{ }^{8} \mathrm{~K}_{0}\right) \\
& =\left({ }^{8} \mathrm{~S}_{0}^{3} \oplus{ }^{8} \overline{\mathrm{~S}}_{0}^{3}\right) \oplus\left({ }^{8} \mathrm{~K}_{0} \oplus{ }^{8} \mathrm{~K}_{0}\right) \\
& =\left({ }^{8} \mathrm{~S}_{0}^{3} \oplus{ }^{8} \overline{\mathrm{~S}}_{0}^{3}\right) \oplus(0) \\
& ={ }^{8} \Delta \mathrm{~S}_{0}^{3}
\end{aligned}
$$



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

- SB. ${ }^{9} \Delta \mathrm{~S}^{1}=\mathrm{SB}\left({ }^{9} \mathrm{~S}^{0}\right) \oplus \mathrm{SB}\left({ }^{8} \overline{\mathrm{~S}}^{0}\right)$. Distributing SB over XOR is not possible.

$$
\begin{gathered}
\mathrm{SB}(00) \oplus \mathrm{SB}(01)=63 \oplus 7 \mathrm{C}=1 \mathrm{~F} \\
\mathrm{SB}(00 \oplus 01)=\mathrm{SB}(01)=7 \mathrm{C} \neq 1 \mathrm{~F} \\
\mathrm{SB}(02) \oplus \mathrm{SB}(03)=77 \oplus 7 \mathrm{~B}=0 \mathrm{C} \neq 1 \mathrm{~F}
\end{gathered}
$$

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.

| ${ }^{9} \Delta \mathrm{~S}^{1}$ |  |  |  |  | ${ }^{9} \Delta \mathrm{~S}^{2}$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| b | 0 | 0 | 0 |  | b | 0 | 0 | 0 |
| c | 0 | 0 | 0 | SR | 0 | 0 | 0 | c |
| d | 0 | 0 | 0 |  | 0 | 0 | d | 0 |
| e | 0 | 0 | 0 |  | 0 | e | 0 | 0 |

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 $\overline{\mathrm{C}}$, and thus $\Delta \mathrm{C}$. Knowing AK has no effect on $\Delta \mathrm{S}$ values, ${ }^{10} \Delta \mathrm{~S}^{2}=\Delta \mathrm{C}$. Again, SR does not affect $\Delta \mathrm{S}$ values, and so ${ }^{10} \Delta \mathrm{~S}^{1}=\mathrm{SR}^{-1}(\Delta \mathrm{C})$. Thus with C and $\overline{\mathrm{C}}$, Eve also knows ${ }^{10} \Delta \mathrm{~S}^{1}$. This attack exploits the known relations that exist in ${ }^{10} \Delta S^{0}$, and the possible ${ }^{10} \mathrm{~K}$ that enable ${ }^{10} \Delta \mathrm{~S}^{1}$ to step back and satisfy these relations. The set of DFE to represent this for ${ }^{10} \Delta S_{C 0}^{0}$ follow.

$$
\begin{align*}
& 2 \mathrm{~b}=\mathrm{SB}^{-1}\left(\mathrm{C}_{0} \oplus{ }^{10} \mathrm{~K}_{0}\right) \oplus \mathrm{SB}^{-1}\left(\overline{\mathrm{C}}_{0} \oplus{ }^{10} \mathrm{~K}_{0}\right)  \tag{3.1}\\
& \mathrm{b}=\mathrm{SB}^{-1}\left(\mathrm{C}_{7} \oplus{ }^{10} \mathrm{~K}_{7}\right) \oplus \mathrm{SB}^{-1}\left(\overline{\mathrm{C}}_{7} \oplus{ }^{10} \mathrm{~K}_{7}\right) \\
& \mathrm{b}=\mathrm{SB}^{-1}\left(\mathrm{C}_{10} \oplus{ }^{10} \mathrm{~K}_{10}\right) \oplus \mathrm{SB}^{-1}\left(\overline{\mathrm{C}}_{10} \oplus{ }^{10} \mathrm{~K}_{10}\right) \\
& 3 \mathrm{~b}=\mathrm{SB}^{-1}\left(\mathrm{C}_{13} \oplus{ }^{10} \mathrm{~K}_{13}\right) \oplus \mathrm{SB}^{-1}\left(\overline{\mathrm{C}}_{13} \oplus{ }^{10} \mathrm{~K}_{13}\right)
\end{align*}
$$

Since $b$ can be any value a byte can hold, except $0, b$ is in $\{1,2, \ldots, 255\}$. Were $b$ zero, fault injection failed, and thus $\mathrm{C}=\overline{\mathrm{C}}$, so there is nothing to exploit. Examining the first equation of the set, regardless of what values $\mathrm{C}_{0}$ and $\overline{\mathrm{C}}_{0}$ hold, of the 255 possible values of 2b, 128 yield $0{ }^{10} \mathrm{~K}_{0}$ key hypothesis which satisfy Equation $3.1,126$ yield 2 and 1 yields 4 . Iterating over all possible values of $\mathrm{C}_{0}, \overline{\mathrm{C}}_{0}$ and ${ }^{10} \mathrm{~K}_{0}$ reveal this. Thus, on average for any one particular value of 2 b , there exists $(2 \times 126+4) / 255=256 / 255$ just over 1 valid ${ }^{10} \mathrm{~K}_{0}$ hypothesis. Considering all 255 possible values of 2 b yields an expected $255 \times \frac{256}{255}=256$ ${ }^{10} \mathrm{~K}_{0}$ hypotheses. This result is not a reduction yet since ${ }^{10} \mathrm{~K}_{0}$ 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 $\left\{{ }^{10} K_{0},{ }^{10} K_{7},{ }^{10} K_{10},{ }^{10} K_{13}\right\}$ 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 $\left(2^{8}\right)^{4}$ to $2^{8}$.

A set of equations like those seen above exist for each of the four columns of ${ }^{10} \Delta \mathrm{~S}_{0}$, 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 $2^{128}=\left(\left(2^{8}\right)^{4}\right)^{4}$ reduces to $\left(2^{8}\right)^{4}=2^{32}$ 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 ${ }^{9} S_{C 0}^{0}$ 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 ${ }^{10} \mathrm{~K}$ keyspace to 1 requires additional $\mathrm{C}, \overline{\mathrm{C}}$ pairs. This produces two independently reduced ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ 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 $\left(2^{8}\right) \times 16=2^{12}$ since stepping each ${ }^{10} \mathrm{~K}_{i}$ 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 ${ }^{9} S_{C 0}^{0}$ requires calculating ${ }^{9} \mathrm{~K}$ which relies on ${ }^{10} \mathrm{~K}$. Individually checking all $2^{32}$ possible keys has a reduction power of $2^{-24}$. If Eve can only obtain one C, $\overline{\mathrm{C}}$ 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 $\left.\mathrm{C}_{i}\right) \times\left(\right.$ possible $\left.\overline{\mathrm{C}}_{i}\right) \times\left(\right.$ possible $\left.{ }^{10} \mathrm{~K}_{i}\right) \times($ storage cost). Each calculation needs to store 4 bytes, $\mathrm{C}^{i}, \overline{\mathrm{C}}^{i},{ }^{10} \mathrm{~K}_{i}$, 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) $\times\left(\right.$ possible $\left.\mathrm{C}_{i}\right) \times\left(\right.$ possible $\left.\overline{\mathrm{C}}_{i}\right) \times\left(\right.$ possible $\left.{ }^{10} \mathrm{~K}_{i}\right) \times($ storage cost). Each calculation needs to store 5 bytes, $\mathrm{r}_{10}, \mathrm{C}_{i}, \overline{\mathrm{C}}_{i},{ }^{10} \mathrm{~K}_{i}$ and b . Thus roughly $256 \times 256 \times 256 \times 256 \times 5 \approx 2^{34.32}$ 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, Sbox $_{0}$. Looking up $0 x 02$ : $S-\operatorname{box}_{0}(0 x 02)=0 x 77$. Rotating $S-$ box $_{0}$ by one results in $S$-box ${ }_{1}$, $\operatorname{SBR}\left(1\right.$, S-box $\left._{0}\right)=$ S-box $_{1}$, Figure 3.10 displays this new rotated S-box. Again looking up $0 x 02$ : $S-$ box $_{1}(0 x 02)=0 x 7 b$. Looking up $0 x 03$ in $S-$ box $_{0}$ achieves this same result. Rotating S-box ${ }_{1}$ by one results in S-box ${ }_{2}, \operatorname{SBR}\left(1\right.$, S-box $\left._{1}\right)=$ S-box ${ }_{2}$. Figure 3.11 visualizes this twice rotated $S$-box. Looking up $0 x 02$ again: $S-$ box $_{2}(0 x 02)=0 x f 2$.

| - | 0x6 | 0x7c | 0 | 0x7b | 0xf2 | 0x6b | 0x6f | 0xc5 | 0x30 | 0 | 0x67 | 0x2b | 0xfe | 0xd7 |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0x1 | 0x | 0x82 | 0xc9 | 0x7d | 0xfa | 0x59 | 0x47 | 0xf0 | 0x | 0x | 0x | 0x | 0x | 4 | 2 |  |
| $0 \times 2$ | 0xb7 | 0xfd | 0x93 | 0x26 | 0x36 | 0x3f | 0xf7 | 0xc | $0 \times 3$ | 0xa5 | 0xe5 | 0xf1 | 0x71 | 0xd8 | $0 \times 31$ | 0x15 |
| 0x3 | 0x04 | 0xc7 | 0x23 | 0xc3 | 0x18 | 0x96 | 0x05 | 0x9a | 0x07 | 0x12 | 0x80 | 0xe2 | 0xeb | 0x27 | 0xb2 | 0x75 |
| $0 \times 4$ | 0x09 | 0x83 | 0x2 | 0x1a | 0x1b | 0x6e | 0x5a | 0xa0 | 0x52 | 0x36 | 0xd6 | 0xb3 | 0x29 | 0xe3 | 0x2f | 0x84 |
| $0 \times 5$ | 0x53 | 0xd1 | 0x00 | 0xe | 0x20 | 0xfc | 0xb1 | 0x5b | 0x6a | 0xcb | 0xbe | 0x39 | 0x4a | 0x4c | $\times 58$ | 0xcf |
| 0x6 | 0xde | 0xef | 0xaa | 0xfb | 0x43 | 0x4d | 0x33 | 0x85 | 0x45 | 0xf9 | 0x02 | 0x7f | 0x50 | 0x3c | x9f | 0xa8 |
| 0x7 | 0x51 | 0xa3 | 0x40 | 0x8f | 0x92 | 0x9d | 0x38 | 0xf5 | 0xbc | 0xb6 | 0xda | 0x21 | $0 \times 10$ | 0xff | 0xf3 | 0xd2 |
| 0x8 | 0xc | 0x0c | 0x13 | 0xec | 0x5f | 0x97 | 0x44 | 0x17 | 0xc4 | 0xa7 | 0x7e | 0x3d | 0x64 | 0x5d | x19 | 0x73 |
| 0x9 | 0x60 | 0x81 | 0x4f | 0xdc | 0x22 | 0x2a | 0x90 | 0x88 | 0x46 | 0xee | 0xb8 | 0x14 | 0xde | 0x5e | $x 0 b$ | 0xdb |
| 0xa | 0xe | 0x32 | 0x3a | 0x0a | 0x49 | 0x06 | 0x24 | 0x5c | 0xc2 | 0xd3 | 0xac | 0x62 | 0x91 | 0x95 | xe4 | -x79 |
| 0xb | 0xe7 | 0xc8 | 0x37 | 0x6d | 0x8d | 0xd5 | $0 \times 4 \mathrm{e}$ | 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 | x8a |
| 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 | x16 |

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


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 $0 \times 4$ | $0 x 2 \mathrm{c} 0 \times 1 \mathrm{a} 0 \times 1 \mathrm{~b}$ 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 $2, \operatorname{SBR}\left(2, S-\right.$ box $\left._{0}\right)$ also computes S -box ${ }_{2}$. In fact, any number of rotations reduce to just one rotation of S-box : $_{0}$
$\operatorname{SBR}\left(r_{n}, \operatorname{SBR}\left(\cdots, \operatorname{SBR}\left(r_{1}, \operatorname{SBR}\left(r_{0}, \operatorname{S}-\operatorname{box}_{0}\right)\right) \cdots\right)\right)=\operatorname{SBR}\left(\left(r_{0}+r_{1}+\cdots+r_{n}\right) \% 256, \operatorname{S}-\operatorname{box}_{0}\right)$.

Addition here is over the integers, not $\mathrm{GF}\left(2^{8}\right)$ 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 mod 256 with $\boxplus$. As previously noted, looking up $0 \times 02$ in S-box ${ }_{1}$ is also $0 \times 03$ in S-box ${ }_{0}$. Adjusting the lookup index of $0 \times 02$ by an increase of 1 has the same effect of rotation. This adjustment is exactly $0 x 02 \boxplus 0 x 01$ since the lookup indicies are the incoming byte nibbles. Again, looking up $0 x 02$ in $S$-box 2 is also $0 x 04$ in $S$-box $_{0}$. Adjusting the lookup index by an increase of 2 produces this same effect. This adjustment is exactly $0 x 02 \boxplus 0 x 02$. In fact, this property holds for all possible rotation values. Similarly, this manipulation holds over the inverse $\mathrm{SBR}^{-1}$ as well.

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

$$
{ }^{10} \Delta S_{0}^{1}=\left({ }^{10} S_{0}^{0} \boxplus{ }^{10} R\right) \oplus\left({ }^{10} \bar{S}_{0}^{0} \boxplus{ }^{10} R\right)
$$

Assuming $\boxplus$ distributes over addition $(\oplus)$ in $\mathrm{GF}\left(2^{8}\right)$, then ${ }^{10} \Delta \mathrm{~S}_{0}^{1}={ }^{10} \mathrm{R} \boxplus\left({ }^{10} \mathrm{~S}_{0}^{0} \oplus{ }^{10} \bar{S}_{0}^{0}\right)=$ ${ }^{10} R \boxplus$ 2b. Similarly, ${ }^{10} \Delta S_{4}^{1}=\left({ }^{10} S_{4}^{0} \boxplus{ }^{10} R\right) \oplus\left({ }^{10} \bar{S}_{4}^{0} \boxplus{ }^{10} R\right)={ }^{10} R \boxplus\left({ }^{10} S_{4}^{0} \oplus{ }^{10} \bar{S}_{4}^{0}\right)={ }^{10} R \boxplus$ b. Now assuming $\boxplus$ distributes over multiplication in $\operatorname{GF}\left(2^{8}\right)$, then ${ }^{10} \Delta S_{0}^{1}=2\left({ }^{10} \mathrm{R} \boxplus \mathrm{b}\right)$ and ${ }^{10} \Delta S_{4}^{1}=\left({ }^{10} R \boxplus b\right)$. Letting $\left({ }^{10} R \boxplus b\right)=v$, the final result is ${ }^{10} \Delta S_{0}^{1}=2 \mathrm{v},{ }^{10} \Delta \mathrm{~S}_{4}^{1}=\mathrm{v}$. 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 $\operatorname{GF}\left(2^{8}\right)$. Testing these assumptions with discrete values shows $\boxplus$ does not distribute over either addition or multiplication in $\operatorname{GF}\left(2^{8}\right)$, so the initial fault propagation remains unexploitable.

$$
\begin{aligned}
& 1 \boxplus(2 \times 3)=1 \boxplus(6)=7 \neq 12=3 \times 4=(1 \boxplus 2) \times(1 \boxplus 3) \\
& 1 \boxplus(2 \oplus 3)=1 \boxplus(1)=2 \neq 7=3 \oplus 4=(1 \boxplus 2) \oplus(1 \boxplus 3)
\end{aligned}
$$

### 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 ${ }^{10} \Delta \mathrm{~S}$ 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 $\left\{{ }^{10} \Delta \mathrm{~S}_{0}^{0}\right\}$. Thus, the likelihood of a random value in $\{1,255\}$ being in $\left\{{ }^{10} \Delta \mathrm{~S}_{0}^{0}\right\}$ is $\frac{127}{255}$. This likelihood is also true for ${ }^{10} \Delta \mathrm{~S}_{4}^{0},{ }^{10} \Delta \mathrm{~S}_{8}^{0}$, and ${ }^{10} \Delta \mathrm{~S}_{12}^{0}$. So, for a given $2 \mathrm{~b} \in\left\{{ }^{10} \Delta \mathrm{~S}_{0}^{1}\right\}$, the probability of $2 \mathrm{~b} \in\left\{{ }^{10} \Delta \mathrm{~S}_{0}^{1}\right\}, \mathrm{b} \in\left\{{ }^{10} \Delta \mathrm{~S}_{4}^{1}\right\}, \mathrm{b} \in\left\{{ }^{10} \Delta \mathrm{~S}_{8}^{1}\right\}$, and $3 \mathrm{~b} \in\left\{{ }^{10} \Delta \mathrm{~S}_{12}^{1}\right\}$ is $1 \times \frac{127}{255} \times \frac{127}{255} \times \frac{127}{255}$. Since there are $1272 b \in\left\{{ }^{10} \Delta S_{0}^{0}\right\}$, the number of $2 b$ expected to satisfy the above relation and be in each set is $127 \times\left(\frac{127}{255}\right)^{3}$.

The 256 possible ${ }^{10} \mathrm{~K}_{0}$ key byte values step back to $127 \mathrm{2b}$ values. So, each valid 2 b value averages to an expected $\frac{256}{127}$ keyspace for that byte. Thus, the average keyspace for a valid $2 \mathrm{~b}, \mathrm{~b}, \mathrm{~b}, 3 \mathrm{~b}$ column is $\left(\frac{256}{127}\right)^{4}$.

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

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

The expected keyspace per valid $2 b$ and the expected number of valid $2 b$ 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: $\left(\frac{256^{4}}{255^{3}}\right)^{4} \approx 2^{32.0677}$.

### 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 2 b or the expected keyspace per valid 2 b 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 $10,{ }^{10} \mathrm{R}$ can be any value in $\{0,1, \ldots, 255\}, 256$ of these $2^{32.0677}$ keyspaces exist. The total keyspace remaining after stepping back to ${ }^{10} \Delta \mathrm{~S}^{0}$ is approximately $2^{32.0677} \times 2^{8}=2^{40.0677}$.

The existing attack reduces in round 9 by stepping back each possible remaining ${ }^{10} \mathrm{~K}$. 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 $\mathrm{K}_{0}$ 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 S-box-1R, 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 ${ }^{-1} \mathrm{r}$ is valid out of 256 , and 256 options are checked, ${ }^{-1} \mathrm{r}$ should reduce to one possibility.
- Key Schedule 2. In key schedule 2, ${ }^{-2} \mathrm{r}$ is the XOR of all encryption key bytes ${ }^{0} \mathrm{~K}$ and ${ }^{0} \mathrm{~K}={ }^{0} \mathrm{RK}$. Expanding this out to the expanded rotation key allows computation of ${ }^{-1} \mathrm{r}$. Checking this ${ }^{-1} \mathrm{R}$ 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 ${ }^{0} \mathrm{~K}$ remain after this ${ }^{-1} \mathrm{R}$ check, two more reduction checks are possible. First, rebuilding the expanded rotation key and calculating is associated ${ }^{10} \mathrm{R}$ value enables a check that this matches the ${ }^{10} \mathrm{R}$ used to create ${ }^{10} \mathrm{~K}$. If multiple possibilities still remain, checking ${ }^{9} \Delta \mathrm{~S}^{0}$ 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 ${ }^{10} \mathrm{~K}$ and few possible ${ }^{9} \mathrm{R}$ 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 AES128 implementations as defined in Section 3.2.2 with the extended attack as described in Section 3.3.4. Specifically, focusing on ${ }^{10} \mathrm{~K}$ keyspace reductions and reductions of valid ${ }^{10} \mathrm{R}$ 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 ${ }^{8} \mathrm{~S}_{0}^{2}$. 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 | 20112.8 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 $\left({ }^{10} \mathrm{~K},{ }^{10} \mathrm{R}\right)$ as part of the reduced keyspace. This pairing counts two identical ${ }^{10} \mathrm{~K}$ recovered with different ${ }^{10} \mathrm{R}$ as separate valid ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ with two ${ }^{10} \mathrm{R}$ values always made this keyspace of two, not two ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ keyspace to one and correctly reverse to the encryption key ${ }^{0} \mathrm{~K}$. Notable data for analysis captured at each stage includes the size of the remaining keyspace, the valid ${ }^{10} \mathrm{R}$ 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) $\times$ (\#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 $2^{128}$ plaintexts $) \times\left(2^{128}\right.$ 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 ${ }^{8} \mathrm{~S}_{0}^{2}$ 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 ${ }^{10} \mathrm{R}$ 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 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction.


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


Figure 5.4: Boxplot of $\log _{2}$ Transformed AES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction.


Figure 5.5: Histogram of $\log _{2}$ Transformed AES-128 ${ }^{10} \mathrm{~K}$ 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 $\left(127 \times\left(\frac{127}{255}\right)^{3} \times 2^{4}\right)^{4}=3,970,610,628 \approx 2^{31.8867}$, 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 $127 \times\left(\frac{127}{255}\right)^{3} \approx 15.6889$. However, discrete values cannot be decimal creating slightly higher and lower values. If each column has 15 valid relations, $2^{31.8867}$ becomes
$\left(15 \times 2^{4}\right)^{4}=3,317,760,000 \approx 2^{31.6275}$, and at 16 becomes $\left(16 \times 2^{4}\right)^{4}=4,294,967,296=$ $2^{32}$. Combinations of 15 and 16 valid relation columns fall between these values and closer to the average (e.g., $\left(15 \times 2^{4}\right)^{2} \times\left(16 \times 2^{4}\right)^{2}=3,774,873,600 \approx 2^{31.8137}$ ). 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 ${ }^{10} \mathrm{~K}$ values and a third had 16 possible ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ Keyspace, 1 Faulty Ciphertext Round 10 Reduction.


Figure 5.12: Histogram of $\log _{2}$ Transformed RAES-128 ${ }^{10} \mathrm{~K}$ 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 $)=\left(\left(S-\right.\right.$ box $_{0}$ reduced keyspace $)+\left(S-\right.$ box $_{1}$ reduced keyspace $)+\ldots+\left(\right.$ S-box $_{255}$ reduced keyspace $\left.)\right)=\left(\right.$ 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 $\left(2^{35} \times 2^{8}=2^{43}\right)$. 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 ${ }^{10} \mathrm{R}$ values, thus one ${ }^{10} \mathrm{~K}$ and two ${ }^{10} \mathrm{R}$ 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 ${ }^{10} \mathrm{R}$ and one ${ }^{10} \mathrm{~K}$. 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, $\mathrm{b} \boxplus 128 \boxplus 128=\mathrm{b} \boxplus 256=\mathrm{b}$. This addition of $128 \bmod 256$ flips the leftmost
bit which is exactly an XOR of 128. Thus, $\operatorname{SBR}(128)=\boxplus 128=\oplus 128$. This one case of $\mathrm{R}=128$ is a special case of the general extension tried in Section 3.3.2. Analyzing this property in the additive $\operatorname{SBR}$ 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.

$$
\begin{aligned}
& \operatorname{SBR}\left({ }^{10} \mathrm{~S}_{0}^{0}, \theta\right) \oplus \operatorname{SBR}\left({ }^{10} \overline{\mathrm{~S}}_{0}^{0}, \theta\right)={ }^{10} \mathrm{~S}_{0}^{1} \oplus{ }^{10} \overline{\mathrm{~S}}_{0}^{1}={ }^{10} \Delta \mathrm{~S}_{0}^{1} \\
& \operatorname{SBR}\left({ }^{10} \mathrm{~S}_{0}^{0}, \theta \boxplus 128\right) \oplus \operatorname{SBR}\left({ }^{10} \bar{S}_{0}^{0}, \theta \boxplus 128\right)=\operatorname{SBR}\left(\operatorname{SBR}\left({ }^{10} \mathrm{~S}_{0}^{0}, \theta\right), 128\right) \oplus \operatorname{SBR}\left(\operatorname{SBR}\left({ }^{10} \bar{S}_{0}^{0}, \theta\right), 128\right) \\
&=\left(\operatorname{SBR}\left({ }^{10} \mathrm{~S}_{0}^{0}, \theta\right) \oplus 128\right) \oplus\left(\operatorname{SBR}\left({ }^{10} \bar{S}_{0}^{0}, \theta\right) \oplus 128\right) \\
&=\left(\operatorname{SBR}\left({ }^{10} \mathrm{~S}_{0}^{0}, \theta\right) \oplus \operatorname{SBR}\left({ }^{10} \overline{\mathrm{~S}}_{0}^{0}, \theta\right)\right) \oplus(128 \oplus 128) \\
&={ }^{10} \Delta \mathrm{~S}_{0}^{1} \oplus 0={ }^{10} \Delta \mathrm{~S}_{0}^{1}
\end{aligned}
$$

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 ${ }^{10} \mathrm{R}$ not 128 apart with the same ${ }^{10} \mathrm{~K}$ which possibly creates an overcount of remaining ${ }^{10} \mathrm{~K}$. 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 ${ }^{10} \mathrm{R}$ values after two ciphertext reductions. Every single attack reduced to two ${ }^{10} \mathrm{R}$ values at this point meaning just the correct ${ }^{10} \mathrm{R}$ and
${ }^{10} \mathrm{R} \oplus 128$ 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 $\mathrm{R}_{10}$, 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 $4{ }^{10} \mathrm{~K}$ values. Application of the key schedule reversal reductions would likely reduce these to just $1{ }^{10} \mathrm{~K}$ value, however this attack implementation did not attempt that reduction.


Figure 5.14: Histogram of $\log _{2}$ Transformed Observed/2 RAES-128 ${ }^{10} \mathrm{~K}$ Keyspace, 2 Faulty Ciphertext Round 10 Reduction.


Figure 5.15: Histogram of Observed/2 RAES-128 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$ after multiple faulty ciphertexts compared to the theoretical expected value or the existing attack on AES. Since only the two ${ }^{10} \mathrm{R} 128$ 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 ${ }^{10} \mathrm{~K}$ 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 ${ }^{10} \mathrm{~K}$. This new scheme uses a rotation value for each round of the encryption key schedule effectively creating a $\left(2^{8}\right)^{10}$ work factor to reverse ${ }^{10} \mathrm{~K}$ 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 $\left({ }^{10} \mathrm{~K}\right.$ enables decryption to the end of round 9 , ${ }^{9} \mathrm{~K}$ 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 AES128 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 RAES128 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,

$$
256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256=2^{80 \prime}
$$

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 Sbox 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 " $\backslash$ 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 ' $\backslash \backslash$ 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;
$\vdots$
create_s_box(s_box, rotate);
key_expansion(expanded_key,key,s_box);

Type 2

```
unsigned char key[16]=1234567890ABCDEF;
\vdots
create_s_box(s_box,rotate);
key_expansion(expanded_key1,key,s_box);
\vdots
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.
$15(\mathrm{Hex})=06^{\wedge} \mathrm{AC}^{\wedge} \mathrm{B} 4^{\wedge} 7 \mathrm{D}^{\wedge} 58^{\wedge} 8 \mathrm{~A}^{\wedge} 9 \mathrm{E}^{\wedge} \mathrm{D} 8^{\wedge} 37^{\wedge} \mathrm{D} 5^{\wedge} 0 \mathrm{E}^{\wedge} 92^{\wedge} 3 \mathrm{C}^{\wedge} 40^{\wedge} 55^{\wedge} \mathrm{B} 5$ (`symbol used for XOR)

The resulting byte value $15(\mathrm{Hex})$ 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;
```

$\vdots$
for (round=1; round<=9; round++)
\{
rotate $=(\text { expanded_key [round } * 4]^{\wedge}$ expanded_key[round ${ }^{*} 4+1$ ]
^expanded_key[round*4+2]^expanded_key[round*4+3])\&mask;
$\vdots$
\}

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 $4 \times 44$ 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]^ ... ^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 shift $1=0$ as the initialization. Thus, on the first pass, this line has no effect. The third line, shift $1=$ shift $1+8$, 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 ${ }^{\wedge}$ 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 $0\left(x^{\wedge} \mathrm{x}^{\wedge} \mathrm{x}^{\wedge} \mathrm{x}=0\right.$ 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 |
| :--- | :--- | :--- |
| $\mathbf{1}$ | 000000000000000000 | B2 43 B5 85 CA DD F4 4E F5 |
|  | 00000000000000 | E6 6E D1 D7 08 B3 0B |
| $\mathbf{2}$ | 000000000000000000 | 37141049 D7 DE 2D 56 CC |
|  | 00000000000001 | 74666 B CF 89 4C 95 |
| $\mathbf{3}$ | 000000000000000000 | 64 E2 C1 53 C4 79 78 DF FC |
|  | 00000000000101 | 873515 9F 4C 39 76 |
| $\mathbf{4}$ | 100000000000000000 | 2F 1D C4 1D DB 52 D6 9D A5 |
|  | 00000000000000 | 749969 9B 16 31 9E |

Table.1. Plaintext \& Ciphertext samples
Key = ADF278565E262AD1F5DEC94A0BF25B28

| SN | Plaintext | Ciphertext |
| :--- | :--- | :--- |
| $\mathbf{1}$ | 000000000000000000 | 2155 66 73 D8 BE 4F 9D 98 55 |
|  | 00000000000000 | 68 D0 06 DC E6 35 |
| $\mathbf{2}$ | 000000000000000000 | 3267 6F 89 15 6E 88 80 D0 82 |
|  | 00000000000001 | 07 9A 0E E2 35 07 |
| $\mathbf{3}$ | 000000000000000000 | 25 AE 71 C2 4F E0 7F A3 AD 23 |
|  | 00000000000101 | 35 84 31 47 2B 9E |
| $\mathbf{4}$ | 100000000000000000 | C9 89 71 71 A1 F0 9D 4A 80 A9 |
|  | 00000000000000 | 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] $\mathbf{3}$ 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 [0255] 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 $4 \times 4$ matrix, by design it is to be stored by filling the rows. When the plaintext is stored as a $4 \times 4$ 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. $2 x$ 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. $2 x$ 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



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

|  |  |  |  |  | F－${ }_{\text {B }}$ |  | －－\％ape |  | \％ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | 践路然 |  |  |  | 号篤め |  | ＊\％\％\％ |  |
|  |  |  | を繇： |  | 畄点品 |  |  | 号管等 |  |  |
|  |  |  |  |  |  |  | －¢ |  |  | ※ |
| 渻 | 渻 | 澳 | 澜 | 澳 | 夣 | 澜 | 谣 | － | 漓 |  |
|  |  | 傌馬長 | F「⿹\zh26灬y |  | 或すご出 |  | \％ | Э忽氜 | Fip ${ }^{\text {\％}}$ |  |
|  | \％ |  |  | \％\％\％${ }_{\text {号忥 }}$ |  | N0ّ＊＊＊ |  | 笭品。 | ま第家 |  |
|  | 蟹枵 | 或鴚苟 |  | \％${ }^{\text {ºn }}$ |  | （10） |  | Э゙ธư |  |  |
|  |  |  | 8ักี้ㄹ |  |  | gE® |  |  | ¢ |  |
| ！ | 畄 | 漓 | 畄 | ！ | 治 | 㽞 | 治 | 滛 | 畄 |  |
|  | ¢¢ ¢ |  | 馬等年 |  | 5＇sx | きがす |  |  | －${ }^{\text {mexw }}$ |  |
|  | \％ | วร์\％\％ | \％\％ |  |  |  | f気号年 | \％\％ |  |  |
| 或緱呂 | กี้ํx | \％ | g＇sự | 第等品真 |  | ※\％ャッ | \％\％\％\％ |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| ！ | ： | 治 | ：${ }^{\text {a }}$ | ： | 策 | 洸 | 治 | ！${ }^{\text {a }}$ | ：${ }^{\text {a }}$ |  |
| \％${ }^{\text {W\％}}$ |  |  | 臓臣 |  | 河易哥 | \％\％ |  |  | －5\％ |  |
|  |  |  |  | \％\％ | ミ以～0\％ |  | 氙～苓号 | － | －\％¢ ¢ |  |
|  |  | ＋解が䓵 |  |  | （\％） | \％ |  |  | \％ |  |
|  |  |  |  |  |  |  | ※匈気気离 |  |  |  |
|  | ${ }^{13}$ | ${ }^{1}$ | ${ }^{1}$ | － | － | \％ | ！${ }_{\text {¢ }}$ | － | ${ }^{\sim}$ |  |
|  | \％\％\％ |  | － |  |  | \％\％\％ibeg |  | 发気き， | 确距 |  |
|  | 気品䒺 | \％\％\％¢ ¢ |  | （10\％ |  | \％\％\％ | 気号： |  |  |  |
|  |  | \％ | 湂号苟 |  | \％\％ | \％＂呂 |  | 跕品㷚 |  |  |
|  |  |  |  | 20愛氬 | 馬気気包 | ²： 2 e |  | 픈ํํํ |  |  |
|  | ， | ， | ＊ |  | ${ }^{\text { }}$ | ， | ！ | ＊ | ＊ | $\cdots$ |
|  |  | ¢ |  |  |  | 5xe | Fixe |  |  |  |
| \％\％wis |  | 韩気呂 | \％ |  |  |  |  |  | 2゙\％${ }^{\text {²\％}}$ | ）\％\％\％ |
|  | \％ |  |  |  |  |  | 各思： |  |  |  |
|  |  |  |  |  |  |  | ¢ ¢ ¢ ¢ ¢ | 䋵䒺式 |  | ๓ะั์ |
|  |  |  |  |  |  | \％\％ |  |  |  |  |
|  |  |  |  | 营呺䟩 |  | 㫛呺罢 |  |  |  |  |
|  |  |  | 等品䋨 |  |  | \％\％\％\％\％ |  | ） |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | 或品品路产 |  | － |  |  | － | 速 |  | － |  |
|  | 20 |  | Anown |  |  |  |  |  | 気呺気耉 |  |
|  |  |  |  |  |  | \％\％\％\％ |  |  |  |  |
|  | － |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 䂞 |  |
| 唁 | 镸 | ${ }^{6}$ | ${ }^{\circ}$ | 仁 | 郘 | 镸 | 嵩 | 镸 | 嵩 | 嵩 |

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

|  |  |  |  |  |  |  |  |  | 感刃刃心呩 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ＊＊＊ | 枵気気 |  | Now |  | \％${ }^{\text {¢ }}$ | 気上気 |  |  | ＊＊Nす。 |  |
|  | 氙䓵큥 | \％ |  |  | 点呺苟 |  |  |  | 倖吉品 |  |
|  |  | 氙贰の烒 | E̊9氝葛 |  | 䪅式氮 | ＊ |  |  |  | 天゙ป苞 |
| 告 | 知 | 㴞 | 告 | 告 | 告 | 告 | 告 | 告 | 告 |  |
|  |  | 可䓵忒の |  |  |  | F－ |  | 馬驓 |  |  |
| 気気気 | 䓌哭忥忥 |  |  |  |  | \％ |  |  | の㷬三苕 |  |
|  |  |  | \％ | 苓出呺 | 运㻤罟 | 気い㐓 | Э \％刃io |  |  |  |
|  |  |  |  |  |  |  |  | 気気氨苞 |  |  |
| 海 | 海 | 治 | ！${ }^{\text {m }}$ | ！${ }^{\text {m }}$ | ！${ }^{\text {a }}$ | 治 | ！${ }_{\text {号 }}$ | 告 | 嫘 |  |
|  |  | 枵易忒 | の－9．0． |  | 軗刃 |  | 弱呺馬 | 聝ずすくす | い寧 |  |
| 加点䓵 |  | 嵒気気 | 岕氨出兌 |  | 点出薥 |  |  |  | 洓总菏 |  |
|  |  |  |  |  |  |  | Now | ， |  |  |
|  |  | 菢氯菑 | 式気気葉 |  | 式匆気苞 | ～ |  |  |  |  |
| 号 | 㐌 | 号 | $\vdots$ | ：${ }^{\text {a }}$ | ！${ }^{\text {a }}$ | ！ | ：${ }^{\text {\％}}$ | ！${ }_{\text {号 }}$ | ！ |  |
| －易哥易 |  |  | 式芴哥 |  | NT\％ |  |  | 政島馬 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| ค＊ |  |  | 忥呺敄品 | \％\％¢ ¢ | 馬훙․․ | 出忒 | － | 亿゙すが |  |  |
|  |  |  |  | 畸気薥 |  |  |  |  |  |  |
|  | ！${ }^{\text {² }}$ | ！${ }^{\text {ºn }}$ | $\cdots$ | ！${ }^{\text {ºn }}$ | ！${ }^{\text {ºn }}$ | ！${ }_{\text {º }}$ | ！${ }^{\text {ºn }}$ | ！${ }^{\text {ºn }}$ | $\vdots$ |  |
|  | 式㰤馬岢 |  | 鵋気 |  | 馬馬苞 |  |  | ～W゙心w |  |  |
|  | ＊ |  |  |  | ～N0 |  |  | \％otw | \％ |  |
|  | －馬きさ |  |  |  |  | 可缺䓵 | ※ֹ\％ | \％ |  |  |
|  |  | 気家兑䒺 |  | 苞苞苞苞 | 禹苞葛㟋 | \％OUCUC気 | 家氮葛苞 |  | 已苍运薥 |  |
| ） | $\therefore$ | ！ | $\cdots$ | $\cdots$ | $\therefore$ ： | $\cdots$ | $\cdots$ | $\vdots$ | $\cdots$ | －${ }^{\text {x }}$ |
|  |  |  | 芻馬馬 |  | 式鶭 | 忥氝岛 | 島馬哥品 | 㦴可気哥 |  | TVoud |
|  | ＊） |  |  | ～～\％\％\％¢ |  | ฐ゙ّ | 枵碞 |  |  | ＊ |
|  |  | 氙愛～呂 | \％ |  |  |  | 寅気恶 | H |  |  |
|  |  |  |  | Eٌ＊氙藻 |  |  | ＊ | 苞矛気苞 |  |  |
|  | E～u | 750 |  | 20 |  |  |  |  |  | N下可品 |
|  |  |  | 古の敬宗 |  |  |  |  |  |  |  |
|  |  |  | 尔かow |  |  | N0\％ |  |  | 出き気缶号 |  |
|  |  |  |  |  |  |  |  |  |  |  |
| \＃woud | Fow |  |  | 20泡君 |  | 氙氙哥䃄 |  |  |  | N0 |
|  |  |  | ton敬召 |  |  | 显苟忒䂞 | 容芴気玺召 |  |  |  |
| 忥呺葸蓳 |  |  |  | \％真忥踪 |  |  |  |  | 出き話等 |  |
| 聝矛佘苞鿬 | ＊ |  |  |  |  |  |  |  |  |  |
| $\begin{aligned} & \text { 䉙 } \\ & \stackrel{\rightharpoonup}{4} \end{aligned}$ |  | $\begin{aligned} & \text { 震 } \\ & \stackrel{y}{*} \end{aligned}$ | $\begin{gathered} \text { H } \\ \frac{0}{3} \\ 0 \\ 0 \end{gathered}$ | $\begin{gathered} 4_{0}^{0} \\ 0 \\ 0 \\ 40 \\ \hline 0 \end{gathered}$ | $\begin{aligned} & \text { 震 } \\ & \stackrel{\rightharpoonup}{5} \end{aligned}$ | 嵒 菏 | $\begin{aligned} & \text { 箬 } \\ & \stackrel{\rightharpoonup}{0} \end{aligned}$ | $\begin{aligned} \text { 蘦 } \\ 0 \end{aligned}$ | 铝 | 嵒 |
| シ $\begin{array}{r}\text { 䂞 } \\ \text { H } \\ 0\end{array}$ | 式碼 | ¢ ${ }_{\text {® }}^{\text {辰 }}$ |  |  | い帬 | ¢ ${ }_{\text {g }}^{\text {g }}$ | \％${ }^{\text {椞 }}$ |  | \％ $\begin{array}{r}\text { g } \\ \text { d } \\ \text { d }\end{array}$ |  |
| $\begin{aligned} & \stackrel{0}{2} \\ & \text { S } \end{aligned}$ | $\stackrel{\circ}{\text { ¢ }}$ | $\stackrel{\circ}{5}$ | $\stackrel{\circ}{\text { ¢ }}$ | $\stackrel{\circ}{\text { ¢ }}$ | $\stackrel{\circ}{5}$ | $\stackrel{\circ}{\text { ¢ }}$ | $\stackrel{\circ}{\text { ¢ }}$ | $\stackrel{\circ}{8}$ | $\stackrel{\circ}{5}$ | $\stackrel{\circ}{5}$ |
| ${ }_{6}$ | 号 | 号 | 宮 | 号 | 号 | 号 | 号 | 号 | S | 号 |

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

| F⿹\zh26灬out |  |  |  | W9\％ow |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 気笏号。 |  |  |  |  |  | ฐّữ |  | ＊＊\％\％\％ |  |
|  | 紹葸等 |  |  | \％\％\％\％ | 或気品 | （\％） | （10ّ0\％ | \％ّ\％ |  |  |
| 㻤预包鳥 | ะับํ ex |  |  |  |  |  |  |  |  | 込ะ遈 |
| ，弾 | ，濁 | 畄 | 蜕 | 韣 | 澢 | ，弾 | 弾 | \％ | 湻 |  |
|  | EEx．os |  |  |  |  | F－馬 |  |  | 工気哥 |  |
| 5ッチ |  | （10\％ | ）\％¢¢ | （\％） | ¢ ¢ \％\％ | \％\％\％\％\％ | \％ |  | 島品尔 |  |
|  |  |  |  |  | 気古号。 | 趗沾萄 |  |  | 縎縎 |  |
|  | 気気気㫄 | －ٌx ex |  |  | 号萗实 |  |  |  |  |  |
| ： | ： | 滛 | 溪 | ：${ }_{\text {告 }}$ | ： | 治 | 号 | 滛 | 涭 |  |
| Eepow |  |  |  |  | TEF易 | \％${ }^{\text {Wexp }}$ | 辱馬氩 | T\％${ }^{\text {Tow }}$ | 戸気気咸 |  |
| 年 |  |  | \％\％\％ |  | \％Wow | 島ฐฐ\％ |  | 鯌出菏 |  |  |
|  |  |  | \％\％ | \％ | \％\％\％ep |  | 5\％ |  |  |  |
| 酸気氬 |  | 包䋑家 | 気絶き |  | 噶运总 | 巡包熈 |  |  |  |  |
| ！${ }_{\text {\％}}$ | ： | ：${ }^{\text {a }}$ | ！${ }^{\text {号 }}$ | 洸 | 号 | ： | ，\％ | ！${ }^{\text {c }}$ | ：${ }^{\text {a }}$ |  |
|  |  | \％\％\％ |  |  |  |  |  |  |  |  |
| シ\％ |  |  |  |  |  |  | 比够き |  |  |  |
|  | \％ |  |  | 敬哭号 |  |  | \％ |  |  |  |
|  | 를뿌․․․ |  |  | こ̇E®̃ |  | \％ |  |  |  |  |
|  | $\stackrel{i}{3}^{\text {i }}$ | $\stackrel{\text { in }}{ }$ | $\stackrel{\text { in }}{ }$ | i | is | ！${ }_{\text {\％}}$ | 汿 | 号 | 㶦 |  |
|  |  | 敢弱気 | \％ |  | 河気忥 |  |  |  |  |  |
|  |  |  |  |  |  |  |  | 或㿾号 |  |  |
|  | － |  | 気你馬品 | 風気菏 |  | （\％） |  | 2易気 | 2＊＊ |  |
|  | 葒気家 |  |  |  |  |  | 薥纆真 |  |  |  |
| ， | ！ | ！ | ， | ！ | ！ | ！ | ！ | ！ | ！ | \％ |
|  |  | 跔察品 |  |  | －¢ |  | EEN菏 |  |  |  |
|  | 緿安喿 |  |  | － |  |  |  | \％\％w |  |  |
|  |  |  |  |  | \％\％\％\％ |  | ¢ | （10 | \％̆\％ |  |
| 玉気気葇 |  |  |  |  |  | 家萗呺品 |  |  |  | ตะั์ |
| －¢0\％ |  |  |  |  | F『To | FFep |  | 7－9］ |  | 20 ${ }^{\text {ax }}$ |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | \％\％\％\％m |  | Ef解： |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | F⿹\zh26灬⿹\zh26灬o |  |  | 践氝可碻 |  |  |
|  | 20w |  | toraw |  |  | 号菏哥 |  | 䢒 |  |  |
| 呺絻缶 |  |  |  |  |  | กั\％\％\％ |  |  | 㮽繙 |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 閏 |
| ${ }_{5}^{5}$ | ${ }_{5}$ | ${ }_{5}^{5}$ | ${ }_{5}^{5}$ | ${ }_{5}^{5}$ | 5 | ${ }_{5}^{5}$ | ${ }_{5}^{5}$ | ${ }_{5}$ | ${ }_{5}^{5}$ | 管 |

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

|  |  |  |  |  |  | FFep |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ＊＊ |  |  | \％ัฐor |  | 畄累䒺 |  |  | \％ัّ\％\％\％ | ＊\％\％\％${ }^{\text {a }}$ |  |
|  |  |  | \％ |  | い㽞亳 |  | 可 |  |  |  |
|  |  |  |  |  |  |  | 䞸馬気 | ฐัะะะํํ |  | ミニッ |
| 鲍 | ，斶 | 涀 | 鲾 | 溾 | 渔 | 朆 | 溾 | 潧 | 澢 |  |
|  | \％${ }^{\text {\％}}$ | Fจ⿹\zh26\％ | 䢻馬 |  |  | －90 |  | F\％ | き䛗氝 |  |
|  | 気够： |  | 点安谷 | 号解兌 |  | ）¢ ¢ ¢ \％\％ |  |  |  |  |
| ¢上镸 |  | ¢ | 気虫忥 | 喀驄 |  | 部喜 | ¢ | 敬点 |  |  |
|  | 気気気包 |  |  | \％\％¢ \％ |  |  |  |  | 気気気気凫 |  |
| 畄 | 畄 | 畄 | 畄 | ： | 治 | 浩 | 畄 | 缶 | ！ |  |
|  |  | －${ }_{\text {axow }}$ |  |  |  | E5ぎ |  | 或気忥馬 |  |  |
|  |  |  | \％$\ddagger$ \％ | 馬～啝高 | ごった | 隹 | ¢ | \％ |  |  |
|  |  |  |  |  | 2030 | 込 | ま呺呂 | ）\％\％ |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| 只 | 漓 | 号 | 治 | ！ | 汾 | 号 | 管 | 漓 | 号 |  |
|  |  |  | －9\％ |  | \％ |  |  | 呺気品品 | \％${ }^{\text {gex }}$ |  |
|  |  | ¢ |  |  | ～ |  | \％ | \％\％¢ \％\％ |  |  |
|  | 込 | 萢 | 第号苞 |  |  |  |  | \％ |  |  |
|  |  |  |  |  | 2 |  | き綧逐 |  |  |  |
|  | $i^{\text {in }}$ | $\square^{\text {\％}}$ |  | $\stackrel{1}{3}^{\text {² }}$ | ${ }^{\text {² }}$ | if | ！${ }^{\text {a }}$ | i | ！${ }^{\text {a }}$ |  |
|  |  | ¢ \％\％\％¢ | F䛗 |  | －鴙易 |  | \％${ }^{\text {¢ }}$ |  |  |  |
|  | ¢ั \％\％ | \％ | \％\％\％ |  |  | 9\％ّ\％ |  |  | \％\％\％¢ |  |
|  | F\％${ }^{\text {¢ }}$ |  | 管 | 路号怘 | \％ |  |  | 䫆出号 |  |  |
|  | 匊氝： |  | 氙玸： |  |  |  |  |  | ฐ®® |  |
| ， | ！ | － | 误 | $\cdots$ | $\cdots$ | ： | ： | ！ | ${ }^{\text {L }}$ |  |
|  |  |  |  | 或㹎品 |  |  |  |  |  |  |
|  | （ |  |  | ） | क¢ 6 ¢0 | 式縸 |  | \％ |  |  |
|  |  |  |  |  | 吅気虫， |  |  | （ |  |  |
|  |  | 迨気気曷 |  |  |  |  |  |  |  | ๓ะั์ ${ }^{\text {¢ }}$ |
|  |  | Ftumb |  |  | F『p |  |  |  |  | 주ํ ${ }^{\text {¢ }}$ |
|  |  |  | 郘品号䇾 |  |  |  |  |  |  |  |
|  |  | \％${ }^{\text {\％}}$ |  |  | \％\％\％\％w |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 或 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | \％\％\％w |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| ${ }^{\circ}$ | $\begin{aligned} & \text { 蓶 } \\ & \hline \end{aligned}$ | 鄙 管 |  | $\begin{aligned} & \text { 营 } \\ & \frac{10}{6} \end{aligned}$ |  |  |  |  | 菏 |  |
|  | 啫 |  |  |  |  |  | 部宕 | 管號 |  |  |
| 号 | 吕 | 亳 | 吕 | 䓂 | 咢 | 吕 | 亳 | 号 | 吕 | 占 |

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 $\mathcal{E}$ Test in Europe Conference $\mathcal{E}$ 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.


