SECURITY OF
BLOCK CIPHERS
SECURITY OF BLOCK CIPHERS
FROM ALGORITHM DESIGN TO HARDWARE IMPLEMENTATION

Kazuo Sakiyama
The University of Electro-Communications, Japan

Yu Sasaki
NTT Secure Platform Laboratories, Japan

Yang Li
Nanjing University of Aeronautics and Astronautics, China
Contents

Preface xi
About the Authors xiii

1 Introduction to Block Ciphers 1
  1.1 Block Cipher in Cryptology 1
      1.1.1 Introduction 1
      1.1.2 Symmetric-Key Ciphers 1
      1.1.3 Efficient Block Cipher Design 2
  1.2 Boolean Function and Galois Field 3
      1.2.1 INV, OR, AND, and XOR Operators 3
      1.2.2 Galois Field 3
      1.2.3 Extended Binary Field and Representation of Elements 4
  1.3 Linear and Nonlinear Functions in Boolean Algebra 7
      1.3.1 Linear Functions 7
      1.3.2 Nonlinear Functions 7
  1.4 Linear and Nonlinear Functions in Block Cipher 8
      1.4.1 Nonlinear Layer 8
      1.4.2 Linear Layer 11
      1.4.3 Substitution-Permutation Network (SPN) 12
  1.5 Advanced Encryption Standard (AES) 12
      1.5.1 Specification of AES-128 Encryption 12
      1.5.2 AES-128 Decryption 19
      1.5.3 Specification of AES-192 and AES-256 20
      1.5.4 Notations to Describe AES-128 23
Further Reading 25

2 Introduction to Digital Circuits 27
  2.1 Basics of Modern Digital Circuits 27
      2.1.1 Digital Circuit Design Method 27
      2.1.2 Synchronous-Style Design Flow 27
      2.1.3 Hierarchy in Digital Circuit Design 29
2.2 Classification of Signals in Digital Circuits
2.2.1 Clock Signal 29
2.2.2 Reset Signal 30
2.2.3 Data Signal 31

2.3 Basics of Digital Logics and Functional Modules
2.3.1 Combinatorial Logics 31
2.3.2 Sequential Logics 32
2.3.3 Controller and Datapath Modules 36

2.4 Memory Modules
2.4.1 Single-Port SRAM 40
2.4.2 Register File 41

2.5 Signal Delay and Timing Analysis
2.5.1 Signal Delay 42
2.5.2 Static Timing Analysis and Dynamic Timing Analysis 45

2.6 Cost and Performance of Digital Circuits
2.6.1 Area Cost 47
2.6.2 Latency and Throughput 47

Further Reading

3 Hardware Implementations for Block Ciphers 49
3.1 Parallel Architecture
3.1.1 Comparison between Serial and Parallel Architectures 49
3.1.2 Algorithm Optimization for Parallel Architectures 50

3.2 Loop Architecture
3.2.1 Straightforward (Loop-Unrolled) Architecture 51
3.2.2 Basic Loop Architecture 53

3.3 Pipeline Architecture
3.3.1 Pipeline Architecture for Block Ciphers 55
3.3.2 Advanced Pipeline Architecture for Block Ciphers 56

3.4 AES Hardware Implementations
3.4.1 Straightforward Implementation for AES-128 58
3.4.2 Loop Architecture for AES-128 61
3.4.3 Pipeline Architecture for AES-128 65
3.4.4 Compact Architecture for AES-128 66

Further Reading

4 Cryptanalysis on Block Ciphers 69
4.1 Basics of Cryptanalysis
4.1.1 Block Ciphers 69
4.1.2 Security of Block Ciphers 70
4.1.3 Attack Models 71
4.1.4 Complexity of Cryptanalysis 73
4.1.5 Generic Attacks 74
4.1.6 Goal of Shortcut Attacks (Cryptanalysis) 77

4.2 Differential Cryptanalysis
4.2.1 Basic Concept and Definition 78
### 4.2.2 Motivation of Differential Cryptanalysis

- Probability of Differential Propagation
- Deterministic Differential Propagation in Linear Computations
- Probabilistic Differential Propagation in Nonlinear Computations
- Probability of Differential Propagation for Multiple Rounds
- Differential Characteristic for AES Reduced to Three Rounds
- Distinguishing Attack with Differential Characteristic
- Key Recovery Attack after Differential Characteristic
- Basic Differential Cryptanalysis for Four-Round AES
- Advanced Differential Cryptanalysis for Four-Round AES
- Preventing Differential Cryptanalysis

### 4.3 Impossible Differential Cryptanalysis

- Basic Concept and Definition
- Impossible Differential Characteristic for 3.5-round AES
- Key Recovery Attacks for Five-Round AES
- Key Recovery Attacks for Seven-Round AES

### 4.4 Integral Cryptanalysis

- Basic Concept
- Processing \( P \) through Subkey XOR
- Processing \( P \) through SubBytes Operation
- Processing \( P \) through ShiftRows Operation
- Processing \( P \) through MixColumns Operation
- Integral Property of AES Reduced to 2.5 Rounds
- Balanced Property
- Integral Property of AES Reduced to Three Rounds and Distinguishing Attack
- Key Recovery Attack with Integral Cryptanalysis for Five Rounds
- Higher-Order Integral Property
- Key Recovery Attack with Integral Cryptanalysis for Six Rounds

### 5 Side-Channel Analysis and Fault Analysis on Block Ciphers

- Introduction
- Intrusion Degree of Physical Attacks
- Passive and Active Noninvasive Physical Attacks
- Cryptanalysis Compared to Side-Channel Analysis and Fault Analysis

- Basics of Side-Channel Analysis
- Side Channels of Digital Circuits
- Goal of Side-Channel Analysis
- General Procedures of Side-Channel Analysis
- Profiling versus Non-profiling Side-Channel Analysis
- Divide-and-Conquer Algorithm

- Side-Channel Analysis on Block Ciphers
- Power Consumption Measurement in Power Analysis
- Simple Power Analysis and Differential Power Analysis
5.3.3 General Key Recovery Algorithm for DPA 164
5.3.4 Overview of Attack Targets 169
5.3.5 Single-Bit DPA Attack on AES-128 Hardware Implementations 181
5.3.6 Attacks Using HW Model on AES-128 Hardware Implementations 186
5.3.7 Attacks Using HD Model on AES-128 Hardware Implementations 192
5.3.8 Attacks with Collision Model † 199
5.4 Basics of Fault Analysis 203
5.4.1 Faults Caused by Setup-Time Violations 205
5.4.2 Faults Caused by Data Alternation 208
5.5 Fault Analysis on Block Ciphers 208
5.5.1 Differential Fault Analysis 208
5.5.2 Fault Sensitivity Analysis † 215
Acknowledgment 223
Bibliography 223

6 Advanced Fault Analysis with Techniques from Cryptanalysis 225
6.1 Optimized Differential Fault Analysis 226
6.1.1 Relaxing Fault Model 226
6.1.2 Four Classes of Faulty Byte Positions 227
6.1.3 Recovering Subkey Candidates of $sk_{10}$ 228
6.1.4 Attack Procedure 230
6.1.5 Probabilistic Fault Injection 231
6.1.6 Optimized DFA with the MixColumns Operation in the Last Round † 232
6.1.7 Countermeasures against DFA and Motivation of Advanced DFA 236
6.2 Impossible Differential Fault Analysis 237
6.2.1 Fault Model 238
6.2.2 Impossible DFA with Unknown Faulty Byte Positions 238
6.2.3 Impossible DFA with Fixed Faulty Byte Position 244
6.3 Integral Differential Fault Analysis 245
6.3.1 Fault Model 246
6.3.2 Integral DFA with Bit-Fault Model 247
6.3.3 Integral DFA with Random Byte-Fault Model 251
6.3.4 Integral DFA with Noisy Random Byte-Fault Model † 254
6.4 Meet-in-the-Middle Fault Analysis 260
6.4.1 Meet-in-the-Middle Attack on Block Ciphers 260
6.4.2 Meet-in-the-Middle Attack for Differential Fault Analysis 263
Further Reading 268

7 Countermeasures against Side-Channel Analysis and Fault Analysis 269
7.1 Logic-Level Hiding Countermeasures 269
7.1.1 Overview of Hiding Countermeasure with WDDL Technique 270
7.1.2 WDDL-NAND Gate 272
7.1.3 WDDL-NOR and WDDL-INV Gates 273
7.1.4 Precharge Logic for WDDL Technique 273
7.1.5 Intrinsic Fault Detection Mechanism of WDDL 276
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>Logic-Level Masking Countermeasures</td>
<td>277</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Overview of Masking Countermeasure</td>
<td>277</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Operations on Values with Boolean Masking</td>
<td>278</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Re-masking and Unmasking</td>
<td>278</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Masked AND Gate</td>
<td>279</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Random Switching Logic</td>
<td>281</td>
</tr>
<tr>
<td>7.2.6</td>
<td>Threshold Implementation</td>
<td>283</td>
</tr>
<tr>
<td>7.3</td>
<td>Higher Level Countermeasures</td>
<td>285</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Algorithm-Level Countermeasures</td>
<td>286</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Architecture-Level Countermeasures</td>
<td>289</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Protocol-Level Countermeasure</td>
<td>290</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>291</td>
</tr>
</tbody>
</table>

Index 293
Preface

The main purpose of this book is to offer a fundamental understanding of security and its implementation of block ciphers. Nowadays, research fields in computer science and engineering have a vast scope and cryptology deals with various topics in information security. In order to understand the cutting-edge technology and science that underlies cryptology, block cipher is one of the best-suited targets both from theoretical and practical points of view. In order to offer the learning materials to fill the gap between theory and practice of the security of block ciphers, our focus goes to cryptanalysis, side-channel analysis, and fault analysis against block ciphers rather than covering all the security issues of block ciphers. AES is currently one of the most researched block ciphers in academia and widely used both in government and in commerce. Considering this fact, the explanations in this book are mainly oriented to the security of AES. In addition, AES is one of the best choices to build up all the discussions from algorithm design to hardware implementation, which is very helpful for readers to follow and to understand the basic ideas that can apply to other block ciphers.

Book Organization

This book is intended as a textbook for undergraduate and graduate students to have a big picture understanding of block ciphers from algorithm to implementations. The contents also include essential knowledge that is useful for cryptographers who are not familiar with hardware, and hardware researchers who are not familiar with the security of block ciphers. This book consists of seven chapters, and each chapter is written by the main authors listed in Table 1.

<table>
<thead>
<tr>
<th>Chapter Number: Chapter Title</th>
<th>KS</th>
<th>YS</th>
<th>YL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Introduction to Block Ciphers</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2: Introduction to Digital Circuits</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3: Hardware Implementations for Block Ciphers</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4: Cryptanalysis on Block Ciphers</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5: Side-Channel Analysis and Fault Analysis on Block Ciphers</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6: Advanced Fault Analysis with Techniques from Cryptanalysis</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>7: Countermeasures against Side-Channel Analysis and Fault Analysis</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
For the purpose of helping readers to understand the chapters, we have prepared several exercises. Some exercises are easy, and suitable for testing the comprehension of each individual learner. Some exercises are moderately difficult, and therefore readers might consider working in a small group as they would on a mini project.

There are several (sub)sections whose titles have a mark “†” at the end. They require knowledge about advanced-level techniques to understand and implement the analysis methods. Readers who find it difficult to follow them are recommended to skip them at the first reading, and focus on understanding the essential concepts of cryptanalysis and side-channel analysis from other sections.

We hope that the readers will enjoy the world of block cipher security and open new horizons through this fantastic field of study.

*Kazuo Sakiyama
Yu Sasaki
Yang Li*
About the Authors

Kazuo Sakiyama is currently a faculty member in the Department of Informatics at the University of Electro-Communications, Tokyo. He received his Ph.D. degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium in 2007. From 1996 to 2004, he was with the Semiconductor and IC Division, Hitachi, Ltd., and engaged in designing system-on-chip LSIs. His current research interests include information security, hardware security, and security analysis of cryptographic modules.

Yu Sasaki received his Ph.D. degree in engineering from the University of Electro-Communications, Tokyo, in 2010. He is currently a member of NTT Secure Platform Laboratories. He has been working with NTT from 2005. His current research interests include cryptography, especially for design and security analysis of symmetric-key cryptography.

Yang Li received his Ph.D. degree in engineering from the Faculty of Informatics and Engineering of the University of Electro-Communications, Tokyo, in 2012. He is currently an associated professor in College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics, China. His main research interests include security evaluation and improvement for cryptographic hardware and embedded systems.
1

Introduction to Block Ciphers

1.1 Block Cipher in Cryptology

1.1.1 Introduction

Information includes our private data that we desire to protect from unwilling leakage depending on the application. Cryptology is a field of research that offers appropriate solutions for the data protection by exploring how to construct a secure communication for fair information exchange. Modern cryptology often deals with digitalized data rather than analog data that cannot be expressed simply with a series of 0s and 1s. In our daily life, information is exchanged by digital devices such as radio frequency identification (RFID) tags, smart cards, and smart phones, where a computational resource is limited. Therefore, it is one of the most important challenges in cryptology to realize an efficient implementation of cryptosystems.

1.1.2 Symmetric-Key Ciphers

There are various ways to realize encryption that is a kind of computational process for information to be protected. In a symmetric-key cipher, information is encrypted with a secret key, and it is expected that the owner of the secret key can decrypt the encrypted information correctly. For instance, let us see the situation, where Alice would like to send a message to Bob in a secure way. If the secret key, $K$, is shared only with Alice and Bob, only Bob can decrypt the message from the encrypted message. The original and the encrypted messages are called plaintext and ciphertext, respectively. Figure 1.1 illustrates the encryption and decryption processes.

The encryption by Alice can be written as

$$ C = E_K(P). $$

(1.1)

The ciphertext is decrypted by Bob as

$$ P = D_K(C). $$

(1.2)

Only Bob can decrypt and read the message, and Eve, who does not own the secret key, cannot decrypt it.
Alice and Bob need to compute the cryptographic operations based on the functions, $E_K(\cdot)$ and $D_K(\cdot)$. The simpler the functions are, the more efficiently they can compute. For instance, the Vernam cipher, invented in 1917, uses just XOR operations as

$$C = P \oplus K, \quad P = C \oplus K$$

(1.3)

to convert plaintext and ciphertext. The XOR operation is explained in Section 1.2.1.

However, in order to guarantee the security, that is, in order that Eve cannot obtain any information of message from $C$, the secret key needs to be refreshed with a random number for each encryption/decryption. In other words, in order to communicate securely with the Vernam cipher, a very long key, which is the same size as $M$, is required. This is significantly inefficient. In general, encryption and decryption processes are based on the trade-offs between cost, performance, and security.

### 1.1.3 Efficient Block Cipher Design

The fundamental idea to achieve an efficient encryption scheme is designing a fixed-input size encryption scheme, and iteratively applying this scheme to encrypt arbitrary length messages. Such a fixed-input size encryption scheme is called block cipher, and the group of bits with the fixed-input size is called block. If the unit of operation is small enough, for example, 1 bit or 1 byte, such a symmetric-key cipher is called stream cipher. As block ciphers are expected to compute encryption and decryption efficiently, they have an iterated structure, and repeat the same function several times. Such a function is called round function. The iterated structure contributes to achieving a small program code in software and implementing a compact circuit design in hardware.

Modern block ciphers are mainly categorized into two kinds: Feistel structure and substitution-permutation network (SPN) structure. Feistel structure was employed in data encryption standard (DES) block cipher proposed in 1977. Including FEAL and Camellia, the Feistel structure has been employed by many block ciphers.
On the contrary, **Advanced Encryption Standard** (AES) employed SPN structure. AES is the main target of this book as it is one of the most widely used block ciphers, and it contains fundamental ideas of SPN structure. The basic mathematics to understand SPN structure and AES specification will be explained later in this chapter.

### 1.2 Boolean Function and Galois Field

**Boolean functions** are used in most of the block ciphers including AES. A Boolean function, \( f \), is described as

\[
  f : \{0, 1\}^n \rightarrow \{0, 1\},
\]

where \( \{0, 1\} \) is called **Boolean domain** and \( \{0, 1\}^n \) is the set of all \( n \)-tuples \((x_1, \ldots, x_n)\), where \( x_1, \ldots, x_n \) are all in Boolean domain.\(^1\)

#### 1.2.1 INV, OR, AND, and XOR Operators

The most simple Boolean function is **inversion** or the INV operation that is a bit complement. It operates as

\[
  \neg x = \begin{cases} 
  1 & (x = 0), \\
  0 & (x = 1),
\end{cases}
\]

where \( \neg \) is used for representing the INV operation. Alternatively, the logic symbol, \( \neg \), is also used for INV. In this book, we allow both usage, that is, \( \neg x = \bar{x} \).

For the case of \( n = 2 \), representative Boolean functions are **OR**, **AND**, and **XOR**. OR is defined as

\[
  x \lor y = \begin{cases} 
  0 & (x = y = 0), \\
  1 & (\text{else}).
\end{cases}
\]

Likewise, AND and XOR are defined, respectively, as

\[
  x \land y = \begin{cases} 
  1 & (x = y = 1), \\
  0 & (\text{else}),
\end{cases}
\]

\[
  x \oplus y = \begin{cases} 
  0 & (x = y), \\
  1 & (x \neq y).
\end{cases}
\]

“\( \lor \)”, “\( \land \)”, and “\( \oplus \)” are used for representing OR, AND, and XOR operations.

The **truth table** for OR, AND, and XOR is described in Table 1.1.

#### 1.2.2 Galois Field

**Finite field** or **Galois field** deals with a finite number of elements. Over a Galois field, addition, subtraction, multiplication, and division are defined. Galois field with the smallest order is

\(^1\)For the case \( n = 0 \), Boolean function denotes a constant, 0 or 1.
Security of Block Ciphers

Table 1.1  Truth table for basic operators

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( x \lor y )</th>
<th>( x \land y )</th>
<th>( x \oplus y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1.2  Operations over \( GF(2) \)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( x + y )</th>
<th>( x \times y )</th>
<th>( -x )</th>
<th>( x^{-1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

called a binary field or \( GF(2) \). For instance, addition, multiplication, additive inverse, and multiplicative inverse over \( GF(2) \) are defined in Table 1.2.

As can be found from Tables 1.1 and 1.2, addition and multiplication over \( GF(2) \) are realized, respectively, with XOR and AND.

Exercise 1.1  Complete Table 1.3, that is, for addition, multiplication, additive inverse, and multiplicative inverse over \( GF(5) \).

1.2.3  Extended Binary Field and Representation of Elements

Binary field, \( GF(2) \), can be extended to a large field size called extended binary field, \( GF(2^n) \), where \( n \) is a positive integer. Especially, in the case of AES, operations in \( GF(2^8) \) are of special interest. The number of elements of \( GF(2^n) \) is \( 2^n \). There are several different representations for the elements, which affect the cost and speed performance of software and hardware implementations.

1.2.3.1  Polynomial Basis Representation

As the number of elements of \( GF(2^n) \) is a power of 2, each bit of the binary representation can be used for each coefficient of a polynomial whose degree is \( n - 1 \). Any element in \( GF(2^n) \) can be expressed with the so-called polynomial basis as

\[
a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_0,
\]  

(1.9)
Table 1.3  Operations over $GF(5)$

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>$x + y$</th>
<th>$x \times y$</th>
<th>$-x$</th>
<th>$x^{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

where $a_i \in \{0, 1\}$. For instance, 16 elements in $GF(2^4)$ can be expressed with the binary representation, $(a_3, a_2, a_1, a_0)_2$. By assigning each bit to the coefficient of a polynomial of $x$, we have $a_3x^3 + a_2x^2 + a_1x + a_0$. Addition of two field elements, for example, $(x + 1) + (x^3 + 1)$, can be calculated as

$$(x + 1) + (x^3 + 1) = x^3 + x,$$  \hspace{1cm} (1.10)

as $1 + 1 = 0$ over $GF(2)$.

Multiplication of the two field elements, for example, $(x + 1)(x^3 + 1)$, needs modular reduction with an irreducible polynomial, for example, $x^4 + x^3 + 1$, which specifies the field.\(^2\) Therefore, the multiplication result becomes as

$$(x + 1)(x^3 + 1) \equiv x^4 + x^3 + x + 1 \equiv x \pmod{(x^4 + x^3 + 1)}. \hspace{1cm} (1.11)$$

1.2.3.2 Normal Basis Representation

Alternatively, elements in $GF(2^n)$ are described using normal basis as

$$b_{n-1}\alpha^{2^{n-1}} + b_{n-2}\alpha^{2^{n-2}} + \cdots + b_0\alpha^0, \hspace{1cm} (1.12)$$

\(^2\) In this case, we also use the expression, $GF(2)[x]/(x^4 + x^3 + 1)$. 
where \( b_i \in \{0, 1\} \) and \( \alpha \) are roots of an irreducible polynomial, \( P(x) \), that is,

\[
P(\alpha) = 0. \tag{1.13}
\]

Furthermore,

\[
\alpha^{2^{n-1}} \equiv 1 \pmod{P(\alpha)}. \tag{1.14}
\]

This can be confirmed by Fermat little theorem.

For the case of \( GF(2^4) \), suppose that \( P(x) = x^4 + x^3 + 1 \), that is, \( P(\alpha) = \alpha^4 + \alpha^3 + 1 = 0 \). Addition in the normal basis representation of \( \alpha^7 + \alpha^{11} \) can be calculated simply by XORing each coefficient of two elements in the form of Equation (1.12). That is,

\[
\alpha^7 + \alpha^{11} = (\alpha^8 + \alpha^4) + (\alpha^4 + \alpha^2) = \alpha^8 + \alpha^2 = \alpha^{10}, \tag{1.15}
\]

where the normal basis representations of \( \alpha^7 \) and \( \alpha^{11} \) can be found in Table 1.4.

This is correct as \( \alpha^7 + \alpha^{11} = \alpha^7(1 + \alpha^3) = \alpha^{10} \). By using the fact of \( \alpha^{15} = 1 \), multiplication in \( GF(2^4) \), for example, \( \alpha^7\alpha^{11} \) is calculated as

\[
\alpha^7\alpha^{11} = \alpha^{18} = \alpha^3. \tag{1.16}
\]

The most advantageous point to use the normal basis representation lies in the fact that squaring is easy to compute in \( GF(2^n) \). As can be found in Table 1.4, squaring for \((b_3, b_2, b_1, b_0)\) is \((b_2, b_1, b_0, b_3)\). More precisely, in squaring, the elements in the normal basis representation are derived as

\[
(b_{n-1}\alpha^{2^{n-1}} + b_{n-2}\alpha^{2^{n-2}} + \cdots + b_0\alpha^{2^0})^2 = b_{n-1}\alpha^{2^n} + b_{n-2}\alpha^{2^{n-1}} + \cdots + b_0\alpha^{2^0}
\]

\[
= b_{n-2}\alpha^{2^{n-1}} + \cdots + b_0\alpha^{2^1} + b_{n-1}\alpha^{2^0}. \tag{1.19}
\]

### Table 1.4 Representations of elements for irreducible polynomial \( x^4 + x^3 + 1 \) in \( GF(2^4) \)

<table>
<thead>
<tr>
<th>Binary ((a_3, a_2, a_1, a_0)_2)</th>
<th>Bit concatenation</th>
<th>Hex.</th>
<th>Polynomial basis</th>
<th>Power of ( \alpha )</th>
<th>Normal basis ((b_3, b_2, b_1, b_0))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0, 0, 0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(0, 0, 0, 1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(0, 0, 1, 0)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(0, 1, 0, 0)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>(1, 0, 0, 0)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>(1, 0, 0, 1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>(1, 0, 1, 1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>(1, 1, 1, 1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>f</td>
</tr>
<tr>
<td>(0, 1, 1, 1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>(1, 1, 1, 0)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>e</td>
</tr>
<tr>
<td>(0, 0, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>(1, 0, 1, 0)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>(1, 1, 0, 1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>(0, 0, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(0, 1, 1, 0)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>(1, 1, 0, 0)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>c</td>
</tr>
</tbody>
</table>
This merit is often used in both software and hardware implementations. However, in general, implementing modular multiplication in the normal basis requires more computation than that in the polynomial basis. Hereafter, we mainly use polynomial basis representation.

1.3 Linear and Nonlinear Functions in Boolean Algebra

1.3.1 Linear Functions

Addition and multiplication by a constant are linear functions in $GF(2^n)$. Suppose that $A(x) = a_{n-1}x^{n-1} + \cdots + a_0$ and $B(x) = b_{n-1}x^{n-1} + \cdots + b_0$, where $a_i, b_i \in \{0, 1\}$. Addition of $A(x)$ and $B(x)$ is

$$A(x) + B(x) = (a_{n-1} \oplus b_{n-1})x^{n-1} + \cdots + a_0 \oplus b_0.$$  (1.20)

From the fact that $a_i \oplus b_i \in \{0, 1\}$, it is confirmed that addition in $GF(2^n)$ is a linear function.

For multiplication by a constant $B$, there exist $c_{n-1}, \ldots, c_0 \in \{0, 1\}$ such that

$$A(x) \times B = c_{n-1}x^{n-1} + \cdots + c_0.$$  (1.21)

Therefore, we know that such multiplication in $GF(2^n)$ is also a linear function. It can be easily understood considering the fact that multiplication by a constant can be computed with multiple additions of $A(x)$ in $GF(2^n)$.

**Exercise 1.2** Suppose that $A(x) = x^3 + x^2$ and $B(x) = x^3 + x$ are represented in the polynomial basis. Calculate $A(x) + B(x)$, $2A(x)$, and $3B(x)$ in $GF(2^4)$ when the irreducible polynomial is $x^4 + x^3 + 1$. Note that 2 and 3 are hexadecimal representations of $x$ and $x + 1$, respectively.

**Exercise 1.3** Confirm that modular additive inverse is a linear function.

1.3.2 Nonlinear Functions

On the contrary, (normal) modular multiplication and multiplicative inverse in $GF(2^n)$ are nonlinear functions. The AES block cipher uses a nonlinear function in a part of the design that is based on modular multiplicative inversion in $GF(2)[x]/x^8 + x^4 + x^3 + x + 1$. The multiplicative inverse computation can be done with Fermat’s (little) theorem as

$$a^{-1} \equiv a^{2^{8-2}} \equiv a^{254},$$  (1.22)

for $a \neq 0$. In AES, multiplicative inverse of 0 is mapped to 0.
One of the most optimal ways to compute the inversion is to find addition chain. On the basis of the Itoh–Tsujii algorithm, the computation can be performed with four multiplications and seven modular squarings as

\[
\begin{align*}
(a^2)^2 &= a^2, \\
(a^2a)^2 &= a^8, \\
(a^3)^2^2 &= a^{12}, \\
a^{12}a^3 &= a^{15}, \\
(a^{15})^{2^2} &= a^{240}, \\
a^{240}a^2a^{12} &= a^{254}.
\end{align*}
\]

Equations (1.23a)–(1.23f)

Itoh–Tsujii algorithm utilizes the relationship of

\[a^{2^t - 1} = (a^{2^t - 1})^{2^t - 1}(a^{2^t - 1}).\]

Equation (1.24)

### 1.4 Linear and Nonlinear Functions in Block Cipher

As discussed in Section 1.3, logical operations are classified into linear operations and nonlinear operations. Composition of linear operations is also linear. Hence, if all the cipher’s operations are linear, the resulting cipher is also linear, which is insecure. In order to break the linearity of the cipher, nonlinear operations need to be introduced. However, in general, the cost of implementing nonlinear operations is more expensive than the one for linear operations.

The strategy of the block cipher design is alternately applying nonlinear and linear operations several times. To avoid the heavy cost, nonlinear operation is designed to be weak but its cost is small. In many cases, a nonlinear operation is designed to be operated on a smaller size than the block size, and the operation is applied in parallel to all the data. Then, in order to compensate the weak nonlinear computations, a linear operation mixes the entire block. The strategy is depicted in Figure 1.2. In the following, each of the nonlinear layer and linear layer is further detailed.

#### 1.4.1 Nonlinear Layer

In order to reduce the implementation cost, a nonlinear operation is designed to work on a fraction of the data. Typical choices of the size are 64 bits, 32 bits, 8 bits (called byte),

![Figure 1.2](Image)

**Figure 1.2** Block cipher design strategy. Nonlinear operations and linear operations are alternately applied
### Table 1.5 An example of 4-bit to 4-bit S-box, $S(\cdot)$

<table>
<thead>
<tr>
<th>Input $x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output $S(x)$</td>
<td>c</td>
<td>0</td>
<td>f</td>
<td>a</td>
<td>2</td>
<td>b</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>3</td>
<td>d</td>
<td>7</td>
<td>1</td>
<td>e</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

All values are described in the hexadecimal format.

4 bits (called nibble), and 1 bit. The size of the nonlinear operation is determined depending on the following two aspects.

- type of nonlinear operation
- target platform in which the cipher is implemented.

#### 1.4.1.1 Modular Operation

When the cipher is designed for being used in high-end CPUs, the implementation cost is not a big issue but the operation should be optimized for instructions adopted in such a CPU. Currently, many CPUs operate on 64 or 32 bits, thus the size of the nonlinear operation is also adjusted to 64 or 32 bits. The high-end CPUs can perform the modular addition or subtraction efficiently. The nonlinearity is often introduced by addition or subtraction on modulo $2^{64}$ or $2^{32}$.

#### 1.4.1.2 Substitution Table (S-box)

When the cipher is designed for more resource-constrained hardwares such as micro-controllers, the balance of the implementation cost and the computation efficiency is important. When the CPU register size is smaller than 32 bits, the 32- or 64-bit modular addition cannot be performed efficiently. The hardware implementation also faces some problems for those operations. Typical choices of the size of the nonlinear operation are 8 or 4 bits. Because the size is small, using the substitution table is a popular approach to introduce the nonlinearity. The substitution table, or S-box, is a pre-specified mapping from the input values to the output values. An example of 4-bit to 4-bit S-box is given in Table 1.5.

### Exercise 1.4

Answer the output value of the following computations.

1. $S(2)$
2. $S(a)$
3. $S(2) \oplus a$
4. $S(2 \oplus a)$
5. $S(2) \oplus S(a)$
6. $S(S(2) \oplus S(a))$

### Exercise 1.5

Prove that any 1-bit to 1-bit bijective S-box is a linear mapping rather than nonlinear mapping.
In this S-box, the input value 5 is transformed to b according to the table. A 4-bit to 4-bit S-box is implemented only with $16 \times 4 = 64$ bits of memory, which is very small. An 8-bit to 8-bit S-box is implemented only with $256 \times 8 = 2048$ bits of memory, which is bigger than the 4-bit to 4-bit S-box but can mix the data faster than the 4-bit to 4-bit S-box.

1.4.1.3 Boolean Function

A Boolean function is the smallest tool to introduce the nonlinearity. By using an AND or OR operation, the nonlinearity is introduced in 1 bit. When the cipher is designed to be a very resource constraint environment such as RFID, a Boolean function is a typical choice as a source of the nonlinearity. A Boolean function can also fit the high-end CPUs. Thirty two-bit CPUs can operate bit-wise for each of the 32 bits in parallel. If this is combined with modular additions (not bit-wise), the nonlinearity can be introduced quickly.

It is also a popular approach to specify the input and output correspondence of some Boolean functions as an S-box. If the cipher is implemented with some memory, the S-box can be implemented, and the nonlinearity of several bits can be introduced with 1 table look-up. If the cipher is implemented with small hardware, the logic of the Boolean function is implemented to minimize the implementation cost.

1.4.1.4 Balanced Choice

Unfortunately, there is no obvious choice that shows the overwhelming performance in any implementation environment. When the cipher is designed in multi-platforms, that is, both the high- and low-end environment, an S-box maybe chosen as the source of nonlinearity that shows a relatively good performance in both the environments. The popular choices of the nonlinear operations are summarized in Figure 1.3.

Note that the data is mixed by alternately applying a nonlinear operation and a linear operation. The choice of the nonlinear operation also depends on the choice of the linear operation.

**Figure 1.3** Substitution-permutation network. Popular choices of size and type of nonlinear operations
### 1.4.2 Linear Layer

The purpose of the linear layer is mixing all the output data from the nonlinear layer in which the data is updated in a small part independently. The linear layer is required to be performed efficiently and implemented lightly.

One of the simplest linear operations is XOR. A part of the nonlinear layer output is XORed to another part to mix the data from different parts. The XOR operation can be performed several times between different parts to mix the data more.

The bit-rotation and bit-shift are also simple linear operations. For example, by applying the 1-bit rotation to the entire data, 1-bit from each part will be moved to the next part. The XOR, bit-shift, and bit-rotation can be implemented efficiently in various platforms, thus they are suitable for the block cipher design.

Another important example is a multiplication over a finite field or modular multiplication. Suppose that the size of the nonlinear operation is $n$ bits and each bit of $n$-bit value represents each coefficient of a polynomial whose degree is $n - 1$. As explained in Section 1.3, multiplication over a finite field with some irreducible polynomial $P(x)$ is a linear function. Suppose that the entire data consists of $m$ parts of $n$-bit data, that is, its size is $mn$ bits. The purpose of the linear function is mixing $m$ independent outputs from the nonlinear layer. In order to mix all the $m$ outputs, $m \times m$ matrices are often used.

For instance, when $m = 4$, four $n$-bit values $x_0, x_1, x_2, x_3$ are updated to four $n$-bit values $y_0, y_1, y_2, y_3$ by the following matrix operation:

\[
\begin{bmatrix}
c_0 & c_4 & c_8 & c_{12} \\
c_1 & c_5 & c_9 & c_{13} \\
c_2 & c_6 & c_{10} & c_{14} \\
c_3 & c_7 & c_{11} & c_{15}
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
y_3
\end{bmatrix},
\]

where each $c_i$ is a constant number.

Any combination of linear operations is a linear operation. A popular design approach is combining different types of light linear operations to introduce a strong mixing effect. An example of the linear layer is depicted in Figure 1.4.

---

**Figure 1.4** An example of linear layer consisting of three linear operations. Nonlinear layer is supposed to update data in eight parts independently.
1.4.2.1 Maximum Distance Separable Matrix (MDS Matrix)

A maximum distance separable matrix (in short MDS matrix) is a matrix with some special property useful for block cipher’s design. Considering the usage in block cipher AES, only the case with the same input and output size is discussed here. Let $x$ be the $m$-component input to the matrix, $M$, and $y$ be the $m$-component output from the matrix, that is, $y = Mx$. The matrix $M$ is called MDS if no distinct input-output pairs $(x, y)$ collide in $m$ or more components.

For the application to cryptology, the fact that at least $m + 1$ components differ in distinct pairs of $(x, y)$ is important. In other words, the MDS matrix guarantees a certain amount of change in different input and output values. For instance, suppose that the value of $x$ is slightly modified to $x'$, which differs only 1 bit from $x$, and the corresponding output value $y'$ is computed. The multiplication by the MDS matrix can guarantee that all the $m$ components of the outputs $y$ and $y'$ have different values, meaning that the 1-bit change of the input always changes all the $m$ components of the output.

1.4.3 Substitution-Permutation Network (SPN)

Substitution-permutation network, which is often called SPN, is a design approach to mix a fixed-length input data. SPN is a special form of the iterative application of nonlinear and linear computations.

The substitution layer (or S-layer), which applies a nonlinear operation, is supposed to be an S-box application in a small size. The permutation layer (or P-layer) applies a linear operation to mix the results of the S-layer efficiently.

The SPN structure is adopted in many block ciphers. AES, which is a main target of this book, also adopts the SPN structure.

1.5 Advanced Encryption Standard (AES)

AES is the most widely used block cipher in present time in both governmental and commercial purposes. AES is standardized internationally, and a lot of academic researches and industrial developments have been proposed about AES. This section explains the specification of AES.

The block cipher AES supports three different key sizes: 128 bits, 192 bits, and 256 bits. The corresponding AES algorithms are called AES-128, AES-192, and AES-256, respectively. AES supports a fixed block size: 128 bits. That is to say, when the key is determined, AES provides a bijective map from 128-bit plaintext to 128-bit ciphertext, that is, for a key $K$, AES-128$_{K}$, AES-192$_{K}$, AES-256$_{K}$: $\{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ (Figure 1.5).

1.5.1 Specification of AES-128 Encryption

In high level, the 128-bit key $K$ is expanded to eleven 128-bit subkeys $sk_0, sk_1, \ldots, sk_{10}$ according to the key schedule function, or KSF.

1. The 128-bit key $K$ is set to the first 128-bit subkey $sk_0$.
2. The KSF is computed to update 128-bit subkey $sk_0$ to another 128-bit subkey $sk_1$. 
Similarly, the KSF is iterated nine times. In each time, 128-bit subkey \( sk_{i-1} \) is updated to another 128-bit subkey \( sk_i \) for \( i = 2, 3, \ldots, 10 \).

Then, a plaintext is encrypted to a ciphertext as follows:

1. An XOR of the plaintext and the first subkey \( sk_0 \) is computed, and this value is set to a 128-bit internal state value \( state_1 \). This operation is often called \textit{whitening}.
2. The 128-bit internal state value \( state_1 \) is updated to \( state_2 \) by computing a round function, which also takes as input subkey \( sk_1 \). This operation is called \textbf{round 1} or \textbf{the first round}.
3. The round function is iterated nine times to update the internal state value \( state_2 \) to \( state_3, state_4, \ldots, state_{11} \). In round \( i \), where \( i = 2, 3, \ldots, 10 \), the round function takes as input \( (state_i, sk_i) \) and outputs \( state_{i+1} \). Note that the round function in the last round is slightly different from the other rounds. The last state that is \( state_{11} \) is the ciphertext.

The computation structure of AES-128 in a function level is described in Figure 1.6.

In practice, it is not necessary to compute all the 11 subkeys at the very beginning. For example, the last subkey will not be used until the very end of the encryption process. Thus, generating the last subkey and keeping it in a register is a waste of computation resource. In order to minimize the computation resource, the KSF and the round function updates are computed in parallel round by round. The AES-128 encryption algorithm in the function level can be described as Algorithm 1.1.

1.5.1.1 Preliminaries to Describe Computation Details

In AES, \textbf{byte} represents 8-bit values. AES is a byte-oriented cipher. All operations are defined at byte level. Let \( v \) be a byte value and \( v_7 \| v_6 \| v_5 \| v_4 \| v_3 \| v_2 \| v_1 \| v_0 \) be its bit-wise representation, of which the corresponding vector representation is \( (v_7 v_6 v_5 v_4 v_3 v_2 v_1 v_0)_2 \). In AES, each bit of a byte represents coefficients of polynomial of \( GF(2^8) \):

\[
v_7 x^7 + v_6 x^6 + v_5 x^5 + v_4 x^4 + v_3 x^3 + v_2 x^2 + v_1 x + v_0.
\]  

(1.26)

A byte value can be represented in hexadecimal. For example, the byte \texttt{9b} represents the polynomial \( x^7 + x^4 + x^3 + x + 1 \). 

![Figure 1.5 Three algorithms of AES](image-url)
Figure 1.6  High-level computation structure of the encryption of AES-128. RF and KSF denote the round function and KSF, respectively. RF<sub>last</sub> is the last round function, which is different from the other rounds.

Algorithm 1.1 AES-128 Encryption Algorithm in the Function Level

**Input:** Plaintext P, 128-bit key K, round function RF, the last round function RF<sub>last</sub>, key schedule function KSF

**Output:** Ciphertext C

1: \( sk_0 \leftarrow K \);
2: \( state_1 \leftarrow P \oplus sk_0 \);
3: for \( i = 1, 2, \ldots, 9 \) do
4: \( sk_i \leftarrow KSF(sk_{i-1}) \);
5: \( state_{i+1} \leftarrow RF(state_i, sk_i) \);
6: end for
7: \( C \leftarrow RF_{last}(state_{10}, sk_{10}) \);
8: return \( C \);