國立交通大學

電機與控制工程學系

博 士 論 文

循環碼、MP3 以及先進加密標準之複合式編碼研究

The Study of Hybrid Coding on Convolution Codes, MP3, and AES

研 究 生：顏志旭

指導教授：吳炳飛 教授

中 華 民 國 九 十 四 年 七 月

循環碼、MP3 以及先進加密標準之複合式編碼研究
The Study of Hybrid Coding on Convolution Codes, MP3 and AES

研 究 生：顏志旭	Student：Chih-Hsu Yen
指導教授：吳炳飛	Advisor：Bing-Fei Wu

國 立 交 通 大 學
電機與控制工程學系
博 士 論 文

A Dissertation
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Ph. D.
in

College of Electrical Engineering and Computer Science

July 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年七月

循環碼、MP3 以及先進加密標準之複合式編碼研究

學生：顏志旭

指導教授：吳炳飛

國立交通大學電機與控制工程學研究所博士班

摘　　要

本論文在循環碼、MP3 以及先進加密標準三種編碼技術上，以安全編碼為主軸，通道編碼和訊源編碼為支線，進行複合式編碼之研究，目的在於提升應用面上的編碼效能或其應用價值。我們提出跳躍式的位移暫存器，並將之取代循環碼中的位移暫存器架構，且加入亂數向量產生器，讓循環碼也具加密的效果，如此即可在進行循環編碼的同時，也完成加密功能。而在解碼部份，則修改循序解碼器的架構，使其不但能解循環碼，同時也可以解密。而在 MP3 音樂壓縮法中，我們引入串流加密器，對壓縮後的符號位元和赫夫曼碼等資料加密，以作為數位權利管理中之音樂內容保護技術。同時，針對雙核心平台，設計有效的編解碼流程，讓系統僅需多負擔 1~2%的效能，即可達到具加密功能的 MP3 編碼。最後，我們將$(n+1,n)$ CRC 編碼技巧引進先進加密標準的硬體設計，其中 n 為 4、8 或 16，讓先進加密標準的硬體能夠對抗誤差攻擊法。

The Study of Hybrid Coding on Convolution Codes, MP3 and AES

student：Chih-Hsu Yen Advisors：Dr. Bing-Fei Wu

Department of Electrical and Control Engineering
National Chiao Tung University

ABSTRACT

This dissertation presents three hybrid coding techniques on convolution code, MP3 and AES to enhance the coding efficiency of overall data coding flow or their application value. Each hybrid coding technique is based on secrecy coding, and accompanied by channel coding or source coding. First, state-hopping shift registers are proposed instead of the shift registers used in convolution code to obtain a new coding scheme. A pseudo-random vector generator (PRVG) is incorporated into the new scheme to achieve secure coding and channel coding simultaneously. The new scheme can be either a pure cryptosystem or a secrecy-channel coding by demand. If the secrecy-channel coding is chosen, the decoder of this new scheme is modified from the sequential algorithm, which is a decoding algorithm for convolution code. Finally, we apply (n+1, n) cyclic redundancy check (CRC) in the implementation of AES, where n is 4, 8 or 16 to let that implementation against differential faults attacks (DFA).

誌　　　謝

本論文雖是博士研究成果的句點著作，然而它僅是漫長的博士生涯中的一小部份。在研究的道路上，週遭許多的老師、實驗室夥伴、朋友以及家人，在所有人的幫忙與支柱，才能有今天的成果。

首先，感謝交大提供如此多元且豐富的學術資源以及優良的求學環境— 豐富的館藏、快速穩定的網路、舒適的研究環境以及態度和善的教職員，在這樣的軟硬體環境下進行研究，是愉悅的經驗。

我的指導老師— 吳炳飛教授，除了提供許多寶貴的意見，協助解決研究問題之外，並且以身示教，讓我從中學習待人處世、做事態度、思考邏輯等更珍貴的經驗。而求學期間的風雨、挫折與錯誤，吳老師也付出其耐心、鼓勵與體諒，充分感受到為何教師在中國文化上，有其崇高的地位。吳老師不僅是一個研究上的指導老師，亦是一位生活上的導師，能與吳老師結識、學習與共識，是我的榮幸。

另外，感謝博士求學過程所有共事的實驗室夥伴。歷經六屆的學弟妹交替，非常幸運的，每屆的學弟妹不但在專業領域上，展現優秀的能力，協助我研究，並且於討論中也拓展我在其他專業領域的視野。同時，友善的個性以及團隊合作的能力，讓我在實驗室的生活也能夠充滿歡笑和快樂，是我精神上的重要支柱。還有，周圍的朋友們，在此也非常感謝你們願意分擔我的苦與樂。

最後，感謝父母親、家弟和家妹的全力支持，這對博士生來說是不可或缺的一環，沒有家人的支柱，則難以無慮的進行修業。

僅以本論文

　　　　獻給家人以及所有關愛我的人

經歷

經歷(參予計畫):

1. 2004, 亞鉅電子股份有限公司 《適用於 RISC 之 MP3 編碼器技術》
2. 2003, 中山科學研究院 《具可程式化置換表之 AES ASIC 硬體電路設計研究》
3. 2003, 中山科學研究院 《具資訊保密功能之紅外線協定晶片設計之研究》
4. 2001, 中山科學研究院《The Research of A Configurable Architecture of Vector Signal Processors》

榮譽:

1. 2005, 第七屆矽智產 SIP 設計競賽 「佳作」
2. 2004, 第十八屆龍騰知識論文獎 資訊科技及應用類 「優等獎」
3. 2003, 中華民國科技管理學會 第七屆學生創新獎 「第一名」
4. 2003, 研華文教基金會 第五屆 TIC100 科技創新事業 「銀質獎」
5. 2003, 旺宏第三屆半導體設計與應用大賽 「應用組二獎」
6. 2001, 旺宏第一屆半導體設計與應用大賽 「應用組一獎」暨「最佳創意獎」

著作目錄

Journal

[1.] Chih-Hsu Yen, and Bing-Fei Wu, “An Error-Correcting Stream Cipher Design with State-Hopping Architecture,” Journal of the Chinese Institute of Engineers, Vol. 28, No. 1, pp. 9-16, 2005.

[2.] Chih-Hsu Yen, Hung-Yu Wei, and Bing-Fei Wu, “New Encryption Approaches to MP3 Compression,” WSEAS Transactions on Acoustic and Music, Issue 4, Vol. 1, pp. 165-172, Oct. 2004.

[3.] Chih-Hsu Yen, and Bing-Fei Wu, “An Efficient Implementation of a Low-Complexity MP3 Algorithm with a Stream Cipher,” Multimedia Tools and Applications, (accepted,2005)

[4.] Chih-Hsu Yen, Yu-Shiang Lin, and Bing-Fei Wu, “A Low-complexity MP3 Algorithm Using a New Rate Control and a Fast Dequantization,” IEEE Transactions on Consumer Electronics, (accepted, 2005)

[5.] 吳炳飛、顏志旭、魏宏宇、數位音樂安全編碼的方式，專利案號：092133583。

[6.] Chih-Hsu Yen, and Bing-Fei Wu, “Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard,” IEEE Transactions on Computers, (revised 1, 2005)

[7.] Chih-Hsu Yen, Tsung-Yao Pai, and Bing-Fei Wu, “Efficient Implementations of the Rijndael Algorithm with Changeable Coefficients,” submitted on IEEE Transactions on Computers, 2005.

Conference

[1.] Hung-I Chin, Chih-Hsu Yen, and Bing-Fei Wu, “An observer-based secure system with chaotic signals,” Proceedings of 1998 Conference on Industrial Automatic Control & Power Applications, Kaohsiung, pp. A2-28-A2-32, 1998.

[2.] Chih-Hsu Yen, Tsung-Yao Pai, and Bing-Fei Wu, “The Implementations of the Reconfigurable Rijndael Algorithm with Throughput of 4.9Gbps,” accepted on the 16th VLSI Design/CAD Symposium, 2005.

Book

[1.] 吳炳飛、顏志旭、林煜翔、魏宏宇、張芷燕，Audio Coding 技術手冊：MP3 篇，民 93 全華圖書

Contents

List of Figures

List of Tables

Abstract

This dissertation presents three hybrid coding techniques on convolution code, MP3 and AES to enhance the coding efficiency of overall data coding flow or their application value. Each hybrid coding technique is based on secrecy coding, and accompanied by channel coding or source coding. First, state-hopping shift registers are proposed instead of the shift registers used in convolution code to obtain a new coding scheme. A *pseudo-random vector generator* (PRVG) is incorporated into the new scheme to achieve secure coding and channel coding simultaneously. The new scheme can be either a pure cryptosystem or a secrecy-channel coding by demand. If the secrecy-channel coding is chosen, the decoder of this new scheme is modified from the sequential algorithm, which is a decoding algorithm for convolution code. Finally, we apply $(n + 1, n)$ *cyclic redundancy check* (CRC) in the implementation of AES, where n is 4, 8 or 16 to let that implementation against *differential faults attacks* (DFA).

Chapter 1

Introduction

The great pace of internet, wireless services and multimedia services have led to an increasing demand for efficient, secure, and reliable digital data-transmission systems. A typical data-transmission model may be represented by the block diagram shown in Fig. 1.1. The source encoder compresses the original information to increase the utilization of channel. The secrecy encoder encrypts the outputs of source encoder, in order to prevent the transmitting contents from being known directly by eavesdropper. The channel encoder, the end of transceiver, transfers the input into codeword to provide data-resilience transmission on a noisy channel. The receiver is the inverse operation of transceiver. The more stages are in the data-transmission model, the more processing time is needed. Hence the hybrid systems which merge heterogeneous systems, e.g., the source-channel coding, the secrecy-source coding, or the secrecy-channel coding, are developed to cut down the processing time and to achieve the efficient implementation.

Therefore, this work proposes a secrecy-channel coding scheme and a secrecy-source coding scheme. The design of secrecy-channel coding scheme is based on the convolution code and a *pseudo-random vector generator* (PRVG) [49] and it has a flexible structure. A new structure of shift registers, named *state-hopping shift register* (SHSR), is proposed in this scheme instead of general shift registers in convolution code, and the PRVG is a calculation of modulo matrix-vector multiplication. The scheme is a flexible system described by a 4-tuple (N_c, N_p, m, M), where N_c and N_p are bit length of plaintext (message) and ciphertext (codeword) respectively, m is the number of registers in one SHSR, and M is the modulus of PRVG. When the 4 tuples are given, the whole system is generated by following proposed

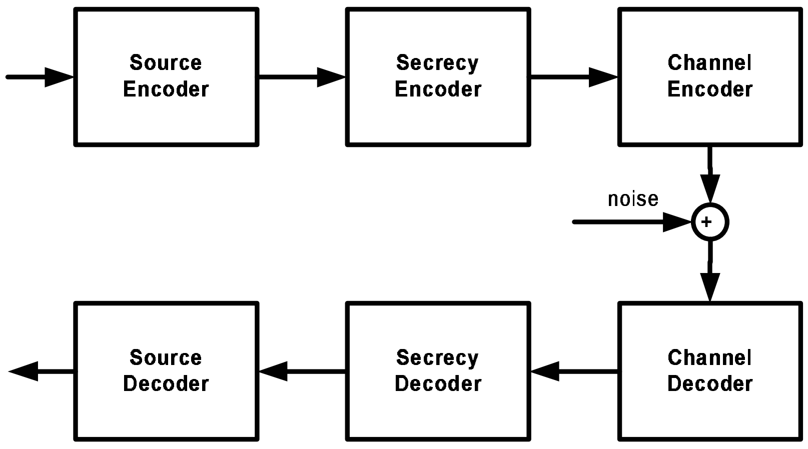


Figure 1.1: Data transmission model with source, secrecy, and channel coding

rules. If N_c equals to N_p, then the generated system is a pure cryptosystem. In this case, the decoder is simply the inverse of encoder. However, if N_c is smaller than N_p, then it is a secrecy-channel coding scheme. The decoder in secrecy-channel coding scheme is far complex than that in the pure cryptosystem, because the decoder not only decrypts the ciphertext but also decodes a codeword to a message. The stack algorithm [40], one of the sequential algorithms, is chosen, and modified to meet the requirement of hybrid decoding. The decoding steps of the modified stack algorithm are as well as those of original stack algorithm, but the fields of stack are extended by adding two new fields, the state and the tracking information.

A secrecy-source coding scheme is the second hybrid coding proposed in this work. It is a combination of MP3 codec [27] and stream ciphers. The main concept is to protect music by encrypting small amounts of data instead of an entire MP3 file; hence, only a partial part of compressed audio data or sign information are selected to be encrypted. The experimental results showed that the encryption introduced 1 2% overhead of encrypting or decrypting music. In the early stage of developing such a system, *software encryption algorithm* (SEAL) [56] is adopted and seamlessly incorporated into MP3 algorithm, but we found that it is not flexible to use the secure MP3 codec. Thereupon, based on the first structure, a more efficiency one is proposed and suitable to dual-core systems as well as to single-core system. The efficiency structure does not limit the type of the stream cipher to

SEAL, and any kind of stream cipher which generates bit sequence can be adopted.

A different kind of hybrid coding from the two kinds described above is proposed in *Advanced Encryption Standard* (AES) [47]. This coding scheme is designed for increasing the security of implementation of AES, not for enhancing the coding efficiency. Because AES is vulnerable to *differential faults analysis* (DFA) [7, 14, 53], the error detection mechanism is required in the implementation of AES against DFA. Once errors are detected, the AES circuit halts and stops outputting erroneous results to prevent the results from being analyzed by cryptanalysts. We propose a $(n + 1, n)$ *cyclic redundancy check* (CRC) as the error detection mechanism in implementation of AES, where $n \in \{4, 8, 16\}$. The parity generation and the syndrome generation of our approach only use the XOR operation over $GF(2^8)$, so the overhead of detecting errors is small. This approach is symmetrical, because encryption and decryption can share hardware of detecting errors. Moreover, it is also scalable, since it can be applied to an 8-bit, 32-bit or 128-bit implementation of AES.

Ultimately, for enhancing the security of AES usage, an implementation of AES with on-line changing coefficients is proposed. This is not about the hybrid coding scheme, nevertheless, we also put the results as an appendix of AES security in Chapter 4. In here, we implement a parameterizable Rijndael in two ways, non-pipeline (normal) and pipeline structure. Because the coefficients are changeable; hence, the chip will operate in different dual ciphers with different given coefficients. The normal structure executes one round per clock cycle on a 128-bit data block, and the pipeline structure requires six clock cycles to perform one round on a 128-bit data block. The data bus of both structures is 32-bit, and only the 128-bit key scheduler is implemented in this work. The normal structure achieves a throughput of 1.7902 Gbps and a 153.84 MHz clock, and has 83.094k gate counts. The pipeline structure has a throughput of 4.9516 Gbps with 425.53 MHz clock and 125.993k gate counts. This implementation of Rijndael is not only compatible to AES but also available to replace the coefficients in Rijndael, so it can be applied to applications that require customized security. Besides, the throughput of our implementations is over 1 Gbps; hence, the results of this work are suitable to network devices over Fast Ethernet or Giga Ethernet. In particular, the *virtual private network* (VPN) is an appropriate application, because this work can provide customized security for VPN users.

This dissertation is organized as follows. Chapter 2 is the first topic of hybrid coding scheme about secrecy-channel coding. Chapter 3 describes the second hybrid coding scheme – secure MP3 algorithm. The two security issues and their countermeasures are discussed in Chapter 4. The future works related to hybrid coding are addressed in Chapter 5.

Chapter 2

A Stream Cipher Based on Convolution Codes

A new architecture of stream cipher based on state-hopping shift registers and a *pseudorandom vector generator* (PRVG) is introduced. The proposed stream cipher merges secrecy coding and channel coding into one processing step. It could be either a pure cryptosystem or a secrecy-channel coding by demand. In aspect of cryptography, the PRVG generates the pseudo random vectors which are treated as keystreams setting up the encryption scheme. Different from the general concept in stream ciphers, state-hopping shift registers do not generate a pseudo random sequence but act as substitutions on plaintexts. From the point of channel coding, the state-hopping shift registers play as the ones in convolution code and the PRVG generates a sequence of pseudorandom vector to determine the Trellis diagram. If the system acts as a pure cryptosystem, the decoding scheme is exactly the inverse of encryption scheme. When the error-correcting ability is chosen, a modified sequential decoding is proposed to decode.

This chapter is organized as follows. The introduction of secrecy-channel coding scheme is given in Section 2.1. Section 2.2 defines the representation of the stream cipher and the most symbols used throughout this manuscript, and the invertibility of the proposed system is also explained. The significant functions of our proposed scheme are separately described in Section 2.3. The design flows of the proposed scheme are depicted in Section 2.4. The encryption scheme with and without the error correction ability, named *inverse SHSR* (iSHSR) and *state-hopping sequential algorithm* (SHS), are shown in Section 2.5. The simulation and discussion of our approach are presented in Section 2.6. The conclusions are given in Section 2.7.

2.1 Overview

The great pace of internet, wireless services and multimedia services have led to an increasing demand for efficient, secure, and reliable digital data-transmission systems. A typical data-transmission model may be represented by the block diagram shown in Fig. 1.1. The source encoder compresses the original information to increase the utilization of channel. The secrecy encoder encrypts the outputs of source encoder, in order to prevent the transmitting contents from being known directly by eavesdropper. The channel encoder, the end of transceiver, transfers the input into codeword to provide data-resilience transmission on a noisy channel. The receiver is the inverse operation of transceiver. The more stages are in the data-transmission model, the more processing time is needed. Hence the hybrid systems which merge heterogeneous systems, e.g., the source-channel coding, the secrecy-source coding, or the secrecy-channel coding, are developed to cut down the processing time and to achieve the efficient implementation.

For secrecy-source coding, the partial encryption scheme is proposed by Cheng and Li [10] to decrease the processing time by only encrypting the important data which are the low-low band information analyzed by the wavelet technique. There are many researches on *source-channel coding* (SCC) [44, 9, 21]. These approaches are to provide each priority class of information with distinct data-resilience level, then the processing time is lowered by coding the significant data only. The technique of a secure and reliable transmission of information is introduced by A. Denis and W. Kinsner [13]. The data integrity is protected by a concatenation of a *Reed-Solomon* (RS) code, interleaved with a self-orthogonal majority decodable convolution code, and the security is achieved through a probabilistic encryption scheme. The resilience and security are realized by two separate coding systems in [13], the secrecy coding and the channel coding.

McEliece [42] presented a public-key cryptosystem based on t-error correcting Goppa code. The main idea is to add a random error vector with Hamming distance $t' < t$ to the encoded message before transmission. Rao and Nam [54, 55] proposed a similar approach, a private-key cryptosystem based on algebraic code. These two schemes execute the secrecy coding and the channel coding in one step. There are two definitions of secrecy-channel coding defined by Rao [23], the *Joint Encryption and Error Correction* (JEEC) scheme and the *Secret Error-Correcting Code* (SECC) scheme. The JEEC has the trade-off problem between data secrecy and data reliability, but SECC does not. Besides, cryptanalysts are unable to correct the noises without the knowledge of key of the both schemes. However, the SECC scheme [23] is attacked by Zeng, Yang, and Rao [64] with a known-plaintext attack.

A new secrecy-channel coding is presented in this manuscript. The proposed stream cipher can be either a pure cryptosystem or a secrecy-channel coding by demand. The design of encryption scheme is based on the shift registers and the PRVG. In general, the contents of the registers and system parameters are initialized by a private key. Then, it becomes possible for encrypting the same plaintext into different ciphertexts by distinct private keys. Not elaborately designing the system will cause fatal results. There are various well-known attacks. The differential cryptanalysis introduced by Biham and Shamir is a chosen-plaintext attack [6]. The basic idea is to compare the exclusive or of two plaintexts with the exclusive or of the corresponding two ciphertexts. The *linear feedback shift registers* (LFSRs) based on stream ciphers are susceptible to various versions of the correlation attack [58, 43, 65].

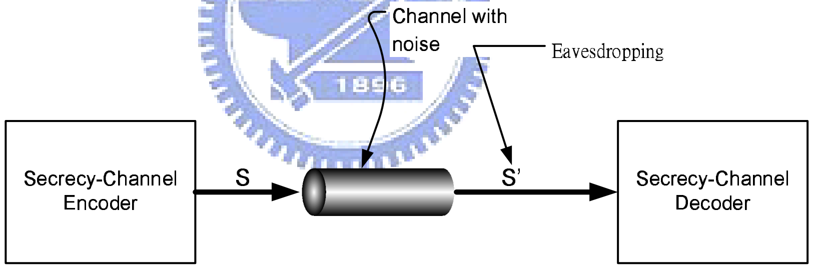


Figure 2.1: The secrecy-channel coding system.

When the pure cryptosystem is chosen, the decryption scheme is just the inverse of decryption scheme. In secrecy-channel coding, the encryption scheme is a maximum likelihood decoding. In Fig. 2.1, the error correction is impossible for lacking of the knowledge about the private key, hence the channel noise will create a more secure channel than the one provided by general systems which do secrecy coding and channel coding in two steps. Noises are removable for cryptanalyst in conventional system, since the scheme of channel decoder is known. Certainly, the security of the secrecy-channel system can not rely on the channel noise. Therefore, a new architecture, the shift register with state hopping, named *state-hopping shift register* (SHSR), is proposed. States of SHSRs can be changed by two sources, one is plaintext and the other is the partial output of the PRVG. An m-degree SHSR has $2m$ state diagrams, and each diagram will randomly appear. Given a private key, the PRVG establishes a sequence of state diagram. The output of an m-degree SHSR is the result of linear combination of registers.

2.2 Preliminary

By the description of stream ciphers in [59], the basic idea of stream ciphers is to generate a keystream $\mathbf{z} = z_1 z_2 \ldots$ and use it to encrypt a plaintext string $\mathbf{p} = p_1 p_2 \ldots$ according to the rule

$$\mathbf{c} = c_1 c_2 \ldots = e_{z_1}(p_1) e_{z_2}(p_2) \ldots,$$

where $\mathbf{c}$ is a ciphertext string and e_{z_i} is an encryption scheme with key z_i.

Definition 1 *A stream cipher is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{F}, \mathcal{E}, \mathcal{D})$, where the following conditions are satisfied [59]:*

1. *$\mathcal{P}$ is a finite set of possible plaintexts.*
2. *$\mathcal{C}$ is a finite set of possible ciphertexts.*
3. *$\mathcal{K}$, the keyspace, is a finite set of possible keys.*
4. *$\mathcal{L}$ is a finite set called the keystream alphabet.*
5. *$\mathcal{F} = (f_1, f_2, \ldots)$ is the keystream generator. For $i > 1$,*
$$f_i : \mathcal{K} \times \mathcal{P}_{i-1} \rightarrow \mathcal{L}.$$
6. *For each $z \in \mathcal{L}$, there is an encryption rule $E_z \in \mathcal{E}$ and a corresponding decryption rule $D_z \in \mathcal{D}$. $E_z : \mathcal{P} \rightarrow \mathcal{C}$ and $D_z : \mathcal{C} \rightarrow \mathcal{P}$ are functions such that $D_z(E_z(x)) = x$ for every plaintext $x \in \mathcal{P}$.*

Define $\mathbb{Z}_l^{m \times n}$ is a set of $m \times n$ matrixes whose entry belongs to the set $\{0, 1, 2, \ldots, l-1\}$. Our stream cipher $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{F}, \mathcal{E}, \mathcal{D})$ is proposed, where $\mathcal{P} = \mathbb{Z}_2^{N_p \times 1}$, $\mathcal{C} = \mathbb{Z}_2^{N_c \times 1}$, $\mathcal{K} = \mathbb{Z}_2^{1 \times N_k}$, $\mathcal{L} = \mathbb{Z}_M^{N \times 1}$, $\mathcal{F}$ is a PRVG by using the *matrix method* [49], $\mathcal{E}$ is an N_p-SHSR, and $\mathcal{D}$ is an N_p-iSHSR or an SHS algorithm.

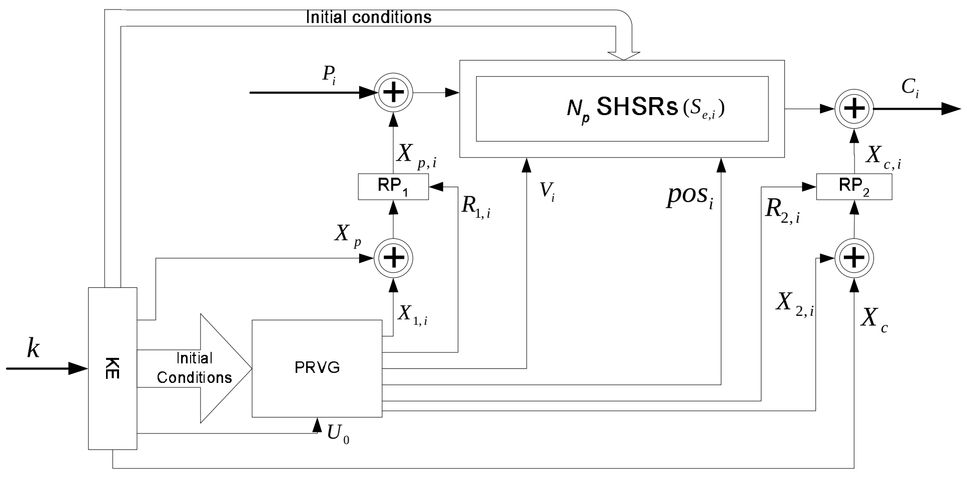


Figure 2.2: The block diagram of encryption scheme

The description of the notations is shown below. N_p and N_c are the bit length of plaintext $P_i \in \mathcal{P}$ and of ciphertext $C_i \in \mathcal{C}$, respectively. The length of the private key $k \in \mathcal{K}$ is N_k bits. M is the modulus used by PRVG and N is the dimension of PRVG. When the error correction ability is chosen for reliable transmission, N_c is greater than N_p, because of the redundancy caused by the channel encoder; otherwise, N_c equals N_p. Instead of 7-tuple presentation of the system in Def. 1, the proposed stream cipher will be represented in a simple form, a 4-tuple (N_c, N_p, m, M) throughout the whole manuscript, where m is the number of registers in one SHSR.

The block diagram of the encryption scheme is shown in Fig. 2.2, the system has three major parts: *Key Expansion* (KE), N-dimensional PRVG, and N_p SHSRs. The KE enlarges the private keyspace,

$$\text{KE} : \mathcal{K} \rightarrow \widetilde{\mathcal{K}}, \text{ where } \widetilde{\mathcal{K}} = \mathbb{Z}_2^{1\times N_{k_e}},$$

to get the expansion key $k_e \in \widetilde{\mathcal{K}}$ of bit length N_{k_e} required for initializing the system. In Fig. 2.2, exclusive-or pairs $\{X_p, X_c\}$, initial conditions of PRVG, and initial states of N_p SHSRs are initialized by $\tilde{k} \in \widetilde{\mathcal{K}}$, a key expanded from k by KE. The PRVG determines the *random permutation* (RP) by giving $R_{1,i}$ and $R_{2,i}$, exclusive-or pairs $\{X_{1,i}, X_{2,i}\}$, and transition of of state diagram of SHSRs by giving V_i and pos_i. The SHSRs take as a bit-substitution function $S_{e,i}$. For each $z_i \in \mathcal{L}$, there is a corresponding $E_{z_i} \in \mathcal{E}$, consisting of

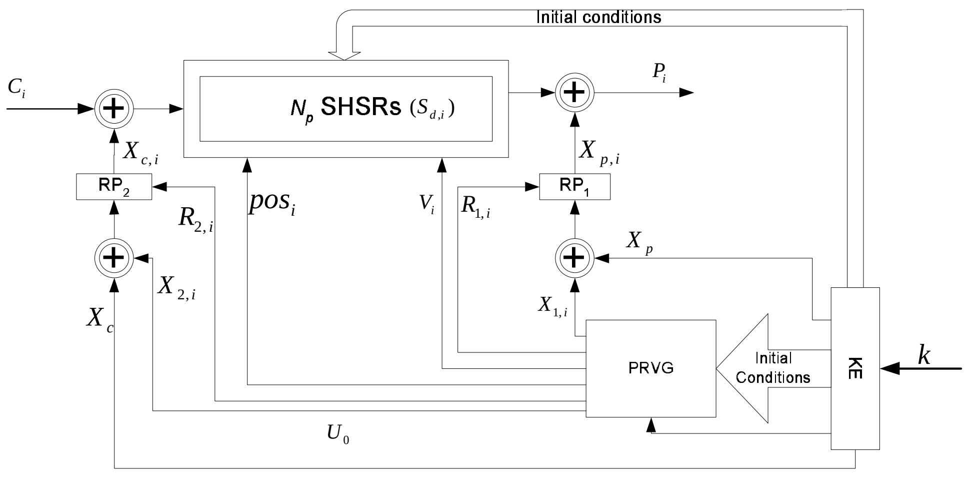


Figure 2.3: The block diagram of decryption scheme

$X_{p,i}$, $X_{c,i}$ and $S_{e,i}$, such that

$$E_{z_i} : \mathcal{P} \to \mathcal{C}.$$

The decryption scheme is obtained by reversing the procedure of encryption scheme and substituting $S_{d,i}$ into $S_{e,i}$. The block diagram of the decryption scheme is depicted in Fig. 2.3.

2.2.1 The Invertibility (without Error Correction)

Given the ith plaintext $P_i \in \mathcal{P}$, according to Fig. 2.2 and Fig. 2.3, the encryption E_z and decryption D_z are the following:

$$E_z : \mathcal{P} \to \mathcal{C},\ C_i = E_{z_i}(P_i) = S_{e,i}(P_i \oplus X_{p,i}) \oplus X_{c,i}, \quad (2.1)$$

$$D_z : \mathcal{C} \to \mathcal{P},\ P_i = D_{z_i}(C_i) = S_{d,i}(C_i \oplus X_{c,i}) \oplus X_{p,i}. \quad (2.2)$$

At the ith time, PRVG generates one subkey z_i and $X_{p,i},\ X_{c,i} \in \mathbb{Z}_2^{N_p \times 1}$ are calculated from z_i. Both $S_{e,i}$ and $S_{d,i}$ are functions of pos_i, V_i, and the past data. Assuming $P_i \in \mathbb{Z}_2^{N_p \times 1}$, the system is invertible if

$$S_{d,i}(S_{e,i}(P_i)) = P_i = \begin{bmatrix} p_{i,0} & p_{i,1} & \dots & p_{i,N_p-1} \end{bmatrix}^T. \quad (2.3)$$

Because an SHSR of degree m is a linear combination of m memories in shift registers whose content is assignable, for N_p independent SHSRs, $S_{d,i}$ will equal to $S_{e,i}$ when three arguments are correctly provided to $S_{d,i}$ and $S_{e,i}$. Thus we can use S_i to represent $S_{e,i}$ and $S_{d,i}$. In an N_p-SHSR, the S_i should be treated as N_p subsystems, $\begin{bmatrix} s_{i,0} & s_{i,1} & \dots & s_{i,N_p-1} \end{bmatrix}$, and each $s_{i,n} : \mathbb{Z}_2^{1\times 1} \to \mathbb{Z}_2^{1\times 1}$. Given $p_{i,n} \in \mathbb{Z}_2^{1\times 1}$, $s_{i,n}$ acts as following:

$$\begin{aligned} c_{i,n} &= s_{i,n}(p_i) = (A_n \otimes B_{i,n}) \oplus p_{i,n} \\ A_n &= \begin{bmatrix} a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix} \\ B_{i,n} &= \begin{bmatrix} b_{n,1} & b_{n,2} & \cdots & b_{n,m} \end{bmatrix}^T , \end{aligned} \tag{2.4}$$

where $A_n \in \mathbb{Z}_2^{1\times m}$ is the coefficient vector of function, $B_{i,n} \in \mathbb{Z}_2^{1\times N}$ is the content in SHSR at the ith time. And $p_n(x) = 1 + a_{n,1}x + a_{n,2}x^2 + \cdots + a_{n,m}x^m$ is the polynomial representation of the nth SHSR. A_n is constant for a given system, and $B_{i,n}$ is modified by randomly changing two entries at most for each $p_{i,n}$. Consider a subsystem $s_{i,n}$ in (2.3), at the ith time, then

$$\begin{aligned} s_{i,n}(s_{i,n}(p_{i,n})) &= s_{i,n}((A_n \otimes B_{i,n}) \oplus p_{i,n}) \\ &= (A_n \otimes B_{i,n}) \oplus ((A_n \otimes B_{i,n}) \oplus p_{i,n}) \\ &= p_{i,n}. \end{aligned} \tag{2.5}$$

The pictorial representation of $s_{i,n}(s_{i,n}(p_{i,n}))$ is shown in Fig. 2.4.

Substituting (2.5) into (2.3), it yields

$$S_i(P_i) = \begin{bmatrix} s_{i,0}(p_{i,0}) & s_{i,1}(p_{i,1}) & \dots & s_{i,N_p}(p_{i,N_p-1}) \end{bmatrix} . \tag{2.6}$$

According to (2.5), $S_i(S_i(P_i)) = P_i$, the cryptosystem is invertible, i.e., $P = D_k(E_k(P))$.

2.2.2 The Invertibility (with Error Correction)

In the case with error correction, the encryption scheme is similar to (2.1) except the relationship of N_p and N_c. Given a system (N_p, N_c, m, M), the encryption is

$$E_z : \mathcal{P} \to \mathcal{C} \tag{2.7}$$

, and the substitution function of N_p-SHSR is

$$S_{e,i} : \mathbb{Z}_2^{N_p\times 1} \to \mathbb{Z}_2^{N_c\times 1}. \tag{2.8}$$

Given the ith plaintext $P_i \in \mathcal{P}$, P_i is encrypted as

$$C_i = E_{z_i}(P_i) = S_{e,i}(P_i \oplus X_{p,i}) \oplus X_{c,i}, \tag{2.9}$$

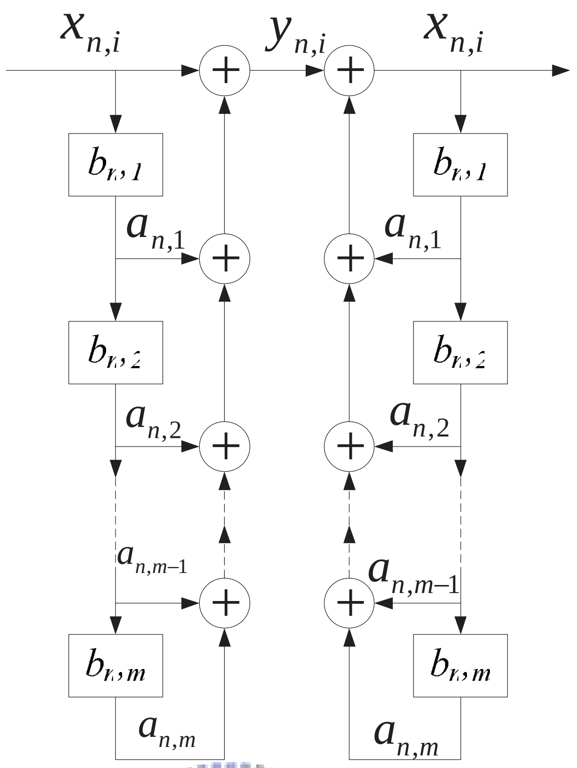


Figure 2.4: The structure of $s_{i,n}(s_{i,n}(p_{i,n}))$

and each matrix A_n in $S_{e,i}$ is

$$A_n = \begin{bmatrix} a_{0,1} & a_{0,2} & \dots & a_{0,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_n,1} & a_{r_n,2} & \dots & a_{r_n,m} \end{bmatrix}$$

, where $\frac{1}{r_n}$ is the coding rate of each SHSR, and $\sum_{n=0}^{N_p-1} \frac{1}{r_n} = N_c$. Then each $s_{i,n}$ in $S_{e,i}$ is a mapping from $\mathbb{Z}_2^{1\times 1}$ to $\mathbb{Z}_2^{r_n\times 1}$.

The decryption scheme is a *Maximum Likelihood Decoder* (MLD). Suppose that the plaintext sequence $P = (P_0,\ P_1,\ ,\dots,\ P_{l-1})$ of lN_p-bit length is encrypted into the ciphertext $C = (C_0,\ C_1,\ \dots,\ C_{l-1})$ of lN_c-bit length, and that a binary sequence $D = (D_0,\ D_1,\ \dots,\ D_{l-1})$ is received over a *Discrete Memoryless Channel* (DMC). The MLD chooses $\hat{C}$ as the transmitted ciphertext C which maximizes the log-likelihood function $\log Pr(D|C)$, where $Pr(D|C)$ is the conditional probability of an event D assuming C.

2.3 Descriptions on Main Functions

Each block in Fig. 2.2 is addressed in this section. The processing flow of this scheme can be divided into two parts: key processing and data processing. The private key k is the input in key processing of which objectives are key expansion and system initialization. In key processing part, only the KE function is involved. In data processing, PRVG generates the pseudorandom vectors or keystream based on the initial conditions given by k_e, and SHSRs substitute plaintexts.

2.3.1 Pseudorandom Vector Generator

The task in PRVG is to produce a sequence of independent and identically distributed random vectors. In our approach, the *matrix method* [49] is adopted to produce pseudorandom vectors. Also the matrix method inherits some of the drawbacks of the linear congruential method, but each vector is not in a systematic form when using it. Before the vectors are sent to next functions, they will be transformed through nonlinear functions to get the suitable bit length.

Because the state of *linear feedback shift register* (LFSR) can be analyzed so easily by *linear complexity analysis* that makes the prediction of state possible. For more complex behavior of the state flow of a shift register, a system proposed here is SHSR which is modified from a shift register to prevent the stream cipher from statistical cryptanalysis. One important duty in PRVG is to vary the state transition of SHSRs that will make intruders hard to attack the system by predicting the state trajectory.

PRVG in our proposed system is depicted in the following.

$$X_{n+1} = (G \cdot X_n + U_j)\ mod\ M \tag{2.10}$$

, where $G \in \mathbb{Z}^{N\times N}, X \in \mathbb{Z}_M^{N\times 1}, U \in \mathbb{Z}_M^{N\times 1}$, and M is a prime.

This system may have the maximum period $M^N - 1$ for some G and M. To let the period of PRVG be maximum, the *characteristic polynomial* of G must be primitive over finite field F_M. Let F_M be a finite field with M elements and $GL(N, F_M)$ be a linear group with order $N^M - 1$. Given a matrix $G \in \mathbb{Z}_M^{N\times N}$, system (2.10) has the maximum period if G satisfies Theorem 1.

Theorem 1 *G is a general linear group $GL(N, F_M)$, if and only if the characteristic polynomial of G is primitive over the finite field F_M [59].*

It is almost impossible to design a *Random Number Generator* (RNG) due to the finite precision of number presentation in digital world, so a PRVG is designed for a very long period such that its period is infinite-like. In order to get a longer period than $M^N - 1$, a simple method is to add a control term which starts off once the period of system (2.10) is detected. Because (2.10) has the maximum period if G satisfies *Theorem* 1, each vector $X_n \in \mathbb{Z}_M^{N\times 1}$ is a periodic point. The dynamics of system (2.10) can be changed by altering the control term $U_j = [u_{0,j}, u_{1,j}, u_{2,j}, u_{3,j}]^T$, where j indexes how many times the period occurs.

Because the PRVG sequence generated by (2.10) is a uniform distribution vector and has the maximum length $N^M - 1$, each $X_n \in \mathbb{Z}_M^{N\times 1}$ is periodic. Herewith the period can be checked by discovering the repetition of X_0. When the period is detected, the control term $U_j = [u_{0,j}\ u_{1,j}\ u_{2,j}\ u_{3,j}]^T$ (U_0 is given by k_e) is computed below:

$$U_{j+1} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \otimes_M U_j, \tag{2.11}$$

where $\otimes_M$ is modulo-M multiplication. A period-check mechanism can dramatically increase the period. Since the modulus M is prime, the set $\{u_{i,0},\ u_{i,1},\ \ldots,\ u_{i,M-1}\}$ is a multiplicative group. With altering control term in (2.10), the period will grow into $(N^M - 1)^2$. The functions controlled by PRVG are itemized as follows:

1. State transition or substitution functions ($S_{e,i}$ and $S_{d,i}$): the distinct pair (pos_i, V_i) alters one state diagram to the others at the ith time. For a (N_c, N_p, m, M) cipher, the maximum number of state diagram is $2m$.

2. Random permutations ($R_{1,i}$ and $R_{2,i}$): the random permutation of $X_p \oplus X_{1,i}$ and of $X_c \oplus X_{2,i}$ will let the intruder hard use differential attacks to get the information about X_p, X_c, $X_{1,i}$, and $X_{2,i}$.

3. Exclusive-or pair: exclusive-or operation acts as mask. There are two types exclusive-or pairs, $\{X_p,\ X_c\}$ and $\{X_{1,i},\ X_{2,i}\}$. The former pair is constant when the private key k is given and the latter is obtained from the output of PRVG.

2.3.2 Expansion Function

The expansion function is used to yield not only k_e but also $R_{1,i} \in \mathcal{Z}_2^{N_p\times 1}$ and $R_{2,i} \in \mathcal{Z}_2^{N_c\times 1}$. If the bit length of PRVG's output is less than N_{RP} bits required to set the permutation, then $R_{1,i}$ and $R_{2,i}$ are created through expansion function with pseudorandom vectors as inputs.

Assume the input of bit length l_i and output of bit length l_o. The expansion function can be implemented by the following steps.

Step 1: Segment the input into $\nu = \lceil \frac{l_i}{8} \rceil$ blocks of which block size is 8 bits. If l_i is not an 8-multiple number, then 0s are attached to the LSB of the input, where n is an element of the set $\{0, 1, \ldots, 7\}$.

Step 2: The ν blocks, $\Lambda = \{\lambda_0, \lambda_1, \ldots, \lambda_{\nu-1}\}$, can be at most grouped into 8 subsets Λ_n=$\{\lambda_\mu \mid \mu \equiv n \bmod 8\}$.

Step 3: The another ν blocks, $\{\tilde{\lambda}_0, \tilde{\lambda}_1, \ldots, \tilde{\lambda}_{\nu-1}\}$, can be obtained by circularly left shifted by n of each entry in set Λ_n.

Step 4: Extend 8ν bits obtained in Step 3 to l_o bits by appending 0s as the LSB of the new block set $\tilde{\Lambda}$.

Step 5: Set $TEMP = \tilde{\Lambda}$. The output is obtained as below:

```
for $\mu = 1 : (l_0 - 8\nu)$
    $\tilde{\Lambda} = \tilde{\Lambda} << \mu$; circular left shift of $\tilde{\Lambda}$ by $\nu$.
    TEMP = TEMP$\oplus\tilde{\Lambda}$
end
output =TEMP
```

If l_i and l_o are smaller than 8, then the block size can be reduced as 4 bits. The illustrative representation of this algorithm is shown in Fig. 2.5.

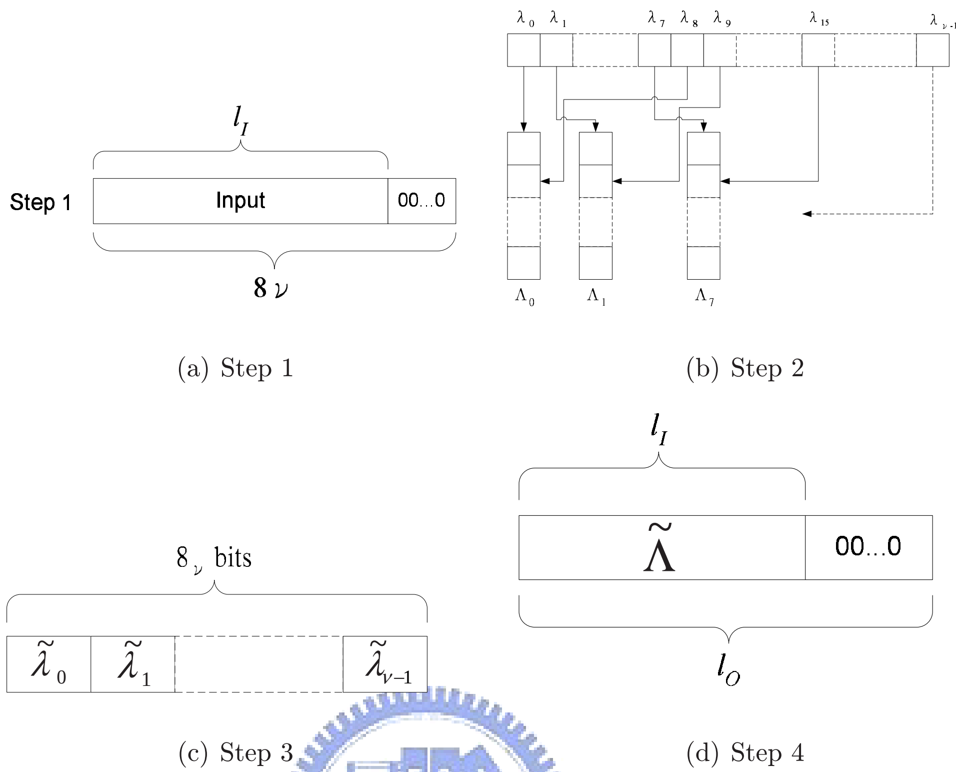

l_I
Step 1
Input
00...0
8ν
(a) Step 1
λ_0 λ_1 λ_7 λ_8 λ_9 λ_{15} $\lambda_{\nu-1}$
Λ_0 Λ_1 Λ_7
(b) Step 2
8_ν bits
$\tilde{\lambda}_0$ $\tilde{\lambda}_1$ $\tilde{\lambda}_{\nu-1}$
(c) Step 3
l_I
$\tilde{\Lambda}$
00...0
l_O
(d) Step 4

Figure 2.5: The illustrative representation of Expansion Algorithm

Key expansion (KE) in Fig. 2.2 expands the original key to meet the requirement of the initialization process. Assume that the key length of k and k_e are N_k and N_{k_e}, respectively, and $N_k + N_p + N_c = N_{k_e}$. KE is an expansion function with $l_i = N_k$, $l_o = N_{k_e}$, and the private key k as input.

2.3.3 Random Permutation

The permutation is a function which maps the input $x = [x_0 \quad x_1 \quad \ldots \quad x_{n-1}]$ into the output $y = [y_0 \quad y_1 \quad \ldots \quad y_{n-1}]$, and the mapping is determined by the $R_{1,i}$ and $R_{2,i}$ of bit length N_{RP}. For a permutation box which is obtained by given $\lceil \log_2 n \rceil$ bits as position indexes, each input bit possibly appears on each output bit. Assume ν_l is the decimal presentation of the lth index, then the output y_l is

$$y_l = x_{\nu_l \bmod n}. \tag{2.12}$$

Hence, for an n-bit random permutation function, the total bits to define the mapping are $n\lceil log_2 n \rceil$. The permutation is a nonlinear mapping, because it will result in an one-to-multiple mapping, i.e., not injective.

N_{RP} required by $R_{1,i}$ and $R_{2,i}$ is calculated by

$$N_{RP} = N_P \cdot \lfloor \log_2 N_p \rfloor + N_c \cdot \lfloor \log_2 N_c \rfloor. \tag{2.13}$$

Because N_{RP} bits needed by RP are probably larger than the bits that PRVG can provide, the expansion function described in the previous subsection can solve this problem. The output bits of PRVG are fed into a expansion function with $l_o = N_{RP}$ and the output of expansion function will be sufficient to set the mapping.

2.3.4 State-Hopping Shift Register

These shift registers are used as substitution functions. All shift registers are formed by distinct primitive polynomials over GF(2) with degree m. There is a new concept introduced into the shift-register structure. In Fig. 2.6, the state is changed by not only shifting the content in registers but also the value of V_i. Besides the plaintext $P_i = [p_{i,0}, p_{i,1}, \ldots, p_{i,N_p-1}]$, $V_i = [v_{i,0}, v_{i,1}, \ldots, v_{i,N_p-1}]$ and $pos_i \in \mathbb{Z}_m^{1\times1}$ are the inputs of SHSRs, where $p_{i,n},\ v_{i,n} \in \mathbb{Z}_2^{1\times1}$.

Given the nth SHSR with the primitive polynomial

$$p(x) = 1 + a_1 x + \cdots + a_{m-1}x^{m-1} + x^m, \tag{2.14}$$

then the output can be obtained by (2.5). The content of the SHSR, $[b_{n,1} \quad b_{n,2} \quad \ldots \quad b_{n,m}]$, is changed by $p_{i,n}$ and $v_{i,n}$ sequentially. After $b_{1,n} = p_{i,n}$, the value $v_{i,n}$ is assigned to SHSR by the following rule.

$$b_{pos_i} = v_{i,n} \tag{2.15}$$

It can be adumbrated that the transition of state diagram is determined by V_i and pos_i obtained from PRVG.

When the input source of the shift register is not one but two, the state transition will be more complex. It is obvious by comparison of Fig. 2.7 and Fig. 2.8. Fig. 2.7 is the state diagram of the shift register with $m = 3$. There is only one state diagram of the shift register. Fig. 2.8 is the possible state diagrams of an SHSR with $m = 3$. The state diagram is determined by (V_i, pos_i), hence the maximum number of state diagram is $2m$. In Fig. 2.8, the sequential corresponding 2-tuple (V_i, pos_i), are $(0/1, 1)$, $(0/1, 1)$ and $(0/1, 1)$. The transition of state diagram is dominated by the sequence of 2-tuple (V_i, pos_i), $i = 0, 1, \ldots$. The state

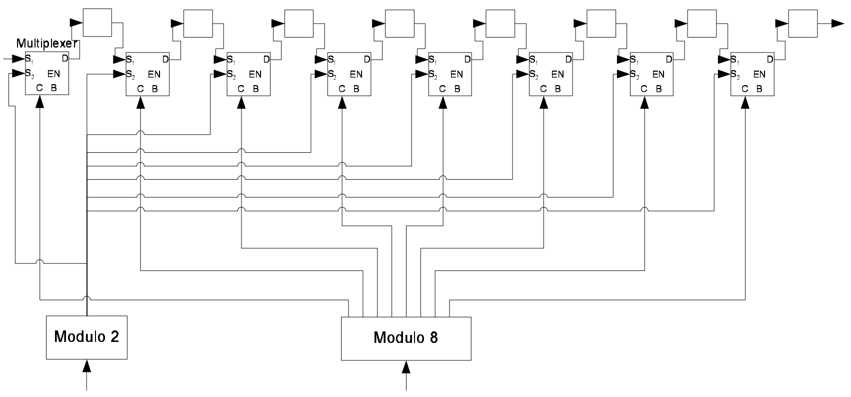


Figure 2.6: The shift register with arbitrary bit assignment

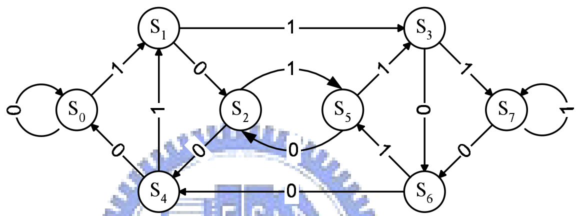


Figure 2.7: State diagram of shift register with $m = 3$

will enter into another state diagram while the (V_i, pos_i) is changed. This will let the state trajectory hard to be predicted, because the transition of state diagrams is dependent on the dynamic of PRVG. An example is shown in Table 2.1. Assume that the primitive polynomials of an SHSR are

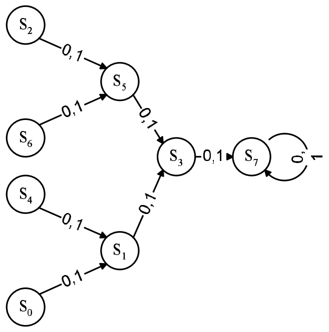
$$p_1(x) = 1 + x + x^3, \text{ and} \tag{2.16}$$

$$p_2(x) = 1 + x^2 + x^3 \tag{2.17}$$

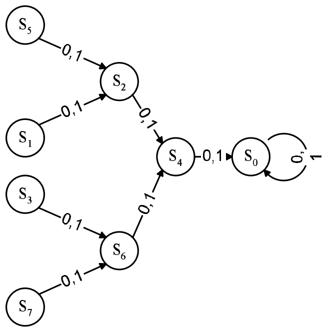
and the content of three registers can be modified by (V_i, pos_i). For plaintext $P = [0 \quad 1 \quad 0 \quad 0]$, the state transition is shown in Table 2.1.

2.4 System Design Flow

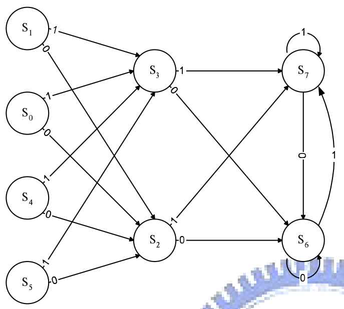
In our scheme, only the expansion algorithm and system structure in Fig. 2.2 are fixed, but the others, e.g., PRVG, SHSRs, the bit assignment on k_e and the output of PRVG, are



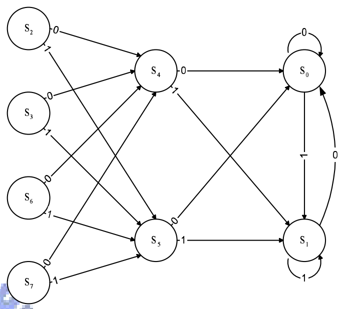
(a) $(V_i, pos_i) = (1, 1)$



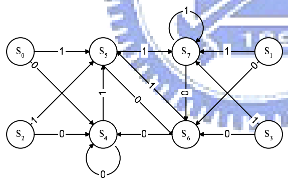
(b) $(V_i, pos_i) = (0, 1)$



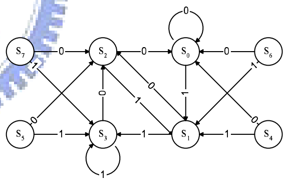
(c) $(V_i, pos_i) = (1, 2)$



(d) $(V_i, pos_i) = (0, 2)$



(e) $(V_i, pos_i) = (1, 3)$



(f) $(V_i, pos_i) = (0, 3)$

Figure 2.8: State diagram of an SHSR with $m = 3$

configurable in the design procedure. Without any consideration about security, a stream cipher $(N_c,\ N_p,\ m,\ M)$ can be arbitrarily chosen by demand under two constraints: $N_p \le N_c$ and a prime $M \ge \max\{\lceil \log_2 m \rceil, \lceil \log_2 N_c \rceil\}$. The system parameters are configured as follows:

Step1: Give a system (N_c, N_p, m, M), then calculate the length N_k of the private key k, the expanded key length N_{ke}, dimension N of PRVG, and the bit length N_{RP} needed by RP function.

Step2: Choose a matrix G such that X_n has the maximum period $M^N - 1$ with nonzero X_0

Table 2.1: State transition of an SHSR with $m = 3$, initial state is S_0.

i	V_i	pos_i	P_i	Next stata	output
0	1	2	0	S_2	00
1	1	2	0	S_6	10
2	0	1	1	S_4	10
3	0	2	0	S_0	11

by the following identity.

$$X_{n+1} = GX_n \bmod M, \text{ where } X_n \in \mathbb{Z}_M^{N\times 1}, G \in \mathbb{Z}_M^{N\times N}. \tag{2.18}$$

Step3: Select N_c m-degree primitive polynomials of which the coefficients are the entries of $A = \{A_0, A_1, \ldots, A_{N_p}\}$ in (2.5).

2.4.1 Parameters Determination

When N_c and N_p are given, N_{RP} is computed by (2.13). And the dimension N is an integer falling in the interval defined below.

$$\left[max(\frac{N_{RP}}{\lfloor \log_2 M \rfloor}, 4)\ ,\ \lceil \frac{N_{RP}}{\lfloor \log_2 M \rfloor} \rceil + 4 \right] \tag{2.19}$$

N_{RP} is the bit length needed by RP_1 and RP_2 in Fig. 2.2. N must be selected from the interval (2.19) with minimum value 4. The reason of restricting N no less than 4 is that four significant parameters, pos_i, V_i, $X_{1,i}$ and $X_{2,i}$ in Fig 2.2, must be separately given by four outputs of PRVG. The input argument, needed by RP_1 and RP_2 to set the permutation table at the ith time, can be obtained from arranging and expanding the four outputs to fit the length N_{RP}.

Before computing the key length N_k and N_{k_e} bits, dimension of PRVG, N, has to be chosen one value from the interval (2.19). For the case that N equals to the maximum integer in (2.19), six parts(RP_1, RP_2, pos_i, V_i, X_c and X_p) use the distinct output bits of PRVG; otherwise, the outputs are shared among RP_1, RP_2, and et al.. From Fig. 2.2, it's obvious that there are four parts which are determined by k_e, exclusive-or pairs(X_p and X_c), control term(U_0), initial conditions of PRVG, and the initial state $B_0 = \{B_{0,1}, B_{0,2}, \ldots, B_{0,m}\}$ of

SHSRs. Given initial conditions X_0 and U_0 of PRVG and the initial state B_0 of SHSRs, N_k is the smallest 8-multiple bits with the following inequality.

$$N_k \geq N_p \times m + 2N\lceil log_2 M\rceil \tag{2.20}$$

The right half side in (2.20) is derived from the required bit length for initializing U_0, X_0, and B_0. For initializing X_p and X_c, N_{k_e} is given by

$$N_{k_e} = N_p + N_c + N_k. \tag{2.21}$$

In the initialization process, the total bit length needed by the stream cipher is N_{k_e}. Once N_k and N_{k_e} are obtained, pass the private key k and the information of l_i and l_o to KE, and KE will expand k to k_e of bit length N_{k_e}.

2.4.2 The design with Error Correction

In general, our scheme can perform secrecy coding and channel coding simultaneously without further modification. In some cases, an SHSR has a problem that the states and some segments of ciphertexts do not change with distinct plaintexts. When the errors occur just before these segments, the decoder may make the wrong decision, that is, the correction ability decreases. The phenomenon can be explained by the example shown in Table 2.1. After stage 2, the outputs of SHSRs and the states are the same no matter what the inputs are. From Table 2.1, the transmitted signal $v = [0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1]$. Assume an additive noise is $e = [0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0]$, then the received signal r is $[0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1]$. The decoder chooses the codeword with zero hamming distance and obtains the output $[0\ \ 1\ \ 1\ \ 1]$, but this error can be corrected by (2, 1, 3) convolution code. This situation can be avoided by not changing the first memory of SHSRs and, for large m, the probability of occurrence of this situation will decrease.

2.5 The Decryption Scheme

The structure of decryption scheme is similar to the encryption scheme in Fig. 2.2, except the design of the SHSRs block. This block can be designed with or without the channel coding. Without error correction ability, i.e. $N_p = N_c$, the decoder is similar to the encoder, and it is simply using the AR structure shown in Fig. 2.4.

When the channel coding is enable, i.e., $N_p < N_c$, the decoding algorithm of convolution code must be suppose to adopt. The Viterbi algorithm and sequential decoding [40] are two frequently used techniques. Due to the intrinsic characteristic of SHSRs, the Viterbi algorithm is hard to modify for decoding. But the sequential algorithm is suitable for SHSR-based stream ciphers. The Viterbi algorithm decoding is based on *trellis diagram*. In the center portion of the trellis, 2^m states are possible for m-memory convolution code, but it is impossible for the one generated by SHSRs. Hence the Viterbi algorithm is not suitable for the proposed stream cipher. The sequential algorithm decoding is based on the code tree as that shown in Fig. 2.9, hence there will have no problems in decoding by sequential algorithm.

An SHSR is an nonlinear error coding, hence there must have a corresponding decoder for each SHSR, as the Fig. 2.10 shows.

2.5.1 Sequential Decoding with State Hopping

There are several algorithms of the sequential decoding , e.g., *stack algorithm*, *Fano algorithm*, *generalized stack algorithm*, and *multiple stack algorithm* [40]. The purpose of a sequential decoding algorithm is to find the maximum likelihood in an efficient way through the code tree. For simplicity, the stack algorithm [40] is adopted.

Step 1: Load the stack with the origin node in the tree, whose metric is taken to be zero.

Step 2: Compute the metric of the successors of the top path in the stack.

Step 3: Delete the top path from the stack.

Step 4: Insert the new paths in the stack, and rearrange the stack in the order of decreasing metric values.

Step 5: If the top path in the stack ends at a terminal node in the tree, stop. Otherwise , return to step2.

A revised state algorithm is proposed here. At Step 2, the two metrics are computed by adding previous metric stored in the stack and the current metric obtained by comparing the outputs of shift registers and received signals. In general, the outputs of shift registers are dependent on the current state, but not for the proposed system. In convolution code, there is only one state diagram, i.e., the state is changed only by the inputs of the shift registers. So the state can be obtained from the decoding path in the stack when computing the metric in *Step 2*.

For our system, the SHSRs are taken as substitution functions. For 3-degree SHSR, the transition between state diagrams Fig. 2.8 (a), (b), and (c) is random. When computing the metric in *Step 2*, the current state needed to yield the outputs of SHSRs can not be observed from the top path stored in the stack. In other words, the corresponding state for a path must be stored in the stack. For example, when a convolution code with memory size of 3 has a decoding result $(1, 0, 1, 1, 0, 0)$, we can say that the current state is $(1, 0, 0)$. But in our system, the state can be also changed by PRVG, hence the current may be not $(1, 0, 0)$.

There still lacks one information in the SHS algorithm. Besides the inputs of SHSRs, the determination of the next state must have the knowledge about the two values, pos_i and V_i in Fig. 2.2. Hence the history of pos_i and V_i have to be recorded.

2.6 Simulation

The experimental results of a (4,8,8,977) system are illustrated in this section. Following the design described earlier, we can obtain that the private key length N_k is 96 bits, the expanded key length N_{k_e} is 108 bits, the dimension N of PRVG is 4, and N_{RP} is $7 \times 2 + 7 \times 3 = 35$. The polynomial matrixes of SHSRs are selected as

$$A_0 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad (2.22)$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, A_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}, \quad (2.23)$$

where each row in A_n is the coefficient of the primitive polynomial in ascending degree. Because of $N_p = 4$ and $N_c = 8$ in the system, the secrecy-channel coding scheme is selected. From (2.22) and (2.23), each subsystem $s_{e,i}$ is an error control code with coding rate 1/2. Note that the code rate is not obtained directly from the division $\frac{N_p}{N_c}$.

Table 2.2: Probability distribution in 4 tests with zero-input, one-input, different keys and different data

Outcome	Test 1	Test 2	Test 3	Test 4
0	0.49362	0.50325	0.49363	0.50054
1	0.50638	0.49676	0.50637	0.49946

2.6.1 Security analysis

In Table 2.2, the probabilities of 0 and 1 are addressed for the four cases. The input in test 1 is a zero vector with length 10^5 bits, and is a 10^5-bit vector of 1 in test 2. There are 10^3 patterns in test 3, each pattern is a 10^3-bit vector with Hamming distance 1. In test 4, the 10^3 zero bits are encrypted in 96 distinct keys $\{k_0, k_1, \ldots, k_{95}\}$, the Hamming distance between k_0 and k_j is 1, where $1 \leq j \leq 95$.

Table 2.3: The experimental results of our system obtained by NIST's statistical test suite

Statistical Test	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value	Ratio
Frequency	6	3	5	3	4	2	5	2	6	4	0.834308	0.9750
Block-Frequency	4	4	6	4	3	4	5	3	1	5	0.834308	1.0000
Cusum	5	6	7	2	5	1	5	5	1	3	0.350485	0.9750
Runs	2	6	5	4	3	4	4	2	3	7	0.739918	1.0000
Long-Run	3	2	3	5	2	6	2	6	6	5	0.637119	1.0000
Rank	2	5	5	8	6	3	4	2	3	2	0.437274	1.0000
FFT	1	1	4	5	3	4	4	5	6	7	0.484646	1.0000
Aperiodic	5	4	4	5	3	4	4	3	4	4	0.999438	1.0000
Serial	6	3	5	3	4	2	5	2	6	4	0.834308	0.9750
Lempel-Ziv	9	3	4	4	1	3	3	2	5	6	0.242986	0.9750
L. Complexity	3	3	3	3	5	7	2	5	5	4	0.834308	1.0000

According to the numerical data shown in Table 2.2, the probability approximates to 0.5 for each case. For security issue or randomness concern, this is a good phenomenon.

It is not helpful for error control coding, since the error control ability is determine by the shortest Hamming weight of the designed code. Table 2.2 also shows that no matter what the distance is calculated by the number of 0 or 1, the Hamming distance is not long enough for constituting a good code for error-control coding.

We also use NIST's statistical test suite [48] to verify the randomness of our system. We generate 40 sequences of 10^5 bits and run the 11 tests, *frequency test, block-frequency test, cumulative sums test, runs test, long-run test, rank test, discrete fourier transform test, non-overlapping template matching test, serial test, Lempel-Ziv test, and linear complexity test*. The results are shown in Table 2.3. The table has 13 columns: column 1 is the name of test, column 12 is the P-value that arises via the application of chi-square test, column 13 is the ratio of sequences that passed the test, columns 2-11 is the distribution of P-value of the give 40 sequences, where C1 to C10 are separately correspond to 10 equal bins obtained by dividing an unit interval. Each row in Table 2.3 is a single test. The test program transform each result into an identical index named P-value. High P-value means that the sequence provides high randomness. In general, if the P-value is greater than 0.01, we can conclude that the sequence is random. As Table 2.3 shows, our system has high P-value above 0.5 in most tests and high passing ratio. From the testing results, we assure the randomness of our system.

Even the *linear congruential PRNG* is not a secure random number generator, the crypt-analyses are hard to obtain the sequence generated by PRVG. Because we add several non-linear functions, as Fig.2.2 shows, to translate the values before using them. Therefore, the security of our system will not be thinned by PRVG.

2.6.2 Performance of error correction ability

Fig. 2.11 is the error probability of this code over an AWGN channel. The system parameters are the same in both cases, the only difference is the changeability of the 1st register in SHSRs. From the simulation result, it's obvious that, under the channel-coding sense, the coding performance in case 1 which the 1st register is unchangeable, is better than the one in case 2 which the 1st register is changeable. Comparing to pure channel coding, our system has normal performance at low SNR, but lower performance at high SNR. This causes by the noise-like state transition of SHSRs.

2.7 Conclusions

The SECC and JEEC schemes presented by Rao and et al. are block coding systems, but our proposed scheme, a new secrecy-channel scheme, is a stream coding system. Our architecture can be either a pure cryptosystem or a secrecy-channel coding system. The combination of secrecy-channel coding reduces the computation time and enjoys an extra benefit that the channel error will make intruders hard to attack. The proposed scheme is also flexible for design. Given a 4-tuple (N_p, N_c, m, M) for an application, following the design flow will get an appropriate system. The plaintexts/ciphertexts can be fast encrypted/decrypted by a pure cryptosystem, since the system is designed based on two simple structures, SHSRs and PRVG. The system security is dependent on three nonlinear functions RPs, SHSRs ,and PRVG. These nonlinear functions and exclusive-or pairs are changed each time with the values indirectly given by PRVG.

Each subsystem $s_{e,i}$ may have different code rates. The error control ability depends on the polynomial of SHSRs, the output of PRVG and the decoding scheme. Comparing with the convolution code, the merging of PRVG and convolution code reduces the error control ability. The SHSRs let the code become a nonlinear code, and the Trellis diagrams are distinct for each key. The key-dependent Trellis diagram is hard to be decoded by a Trellis decoder, however it is suitable in secrecy communication.

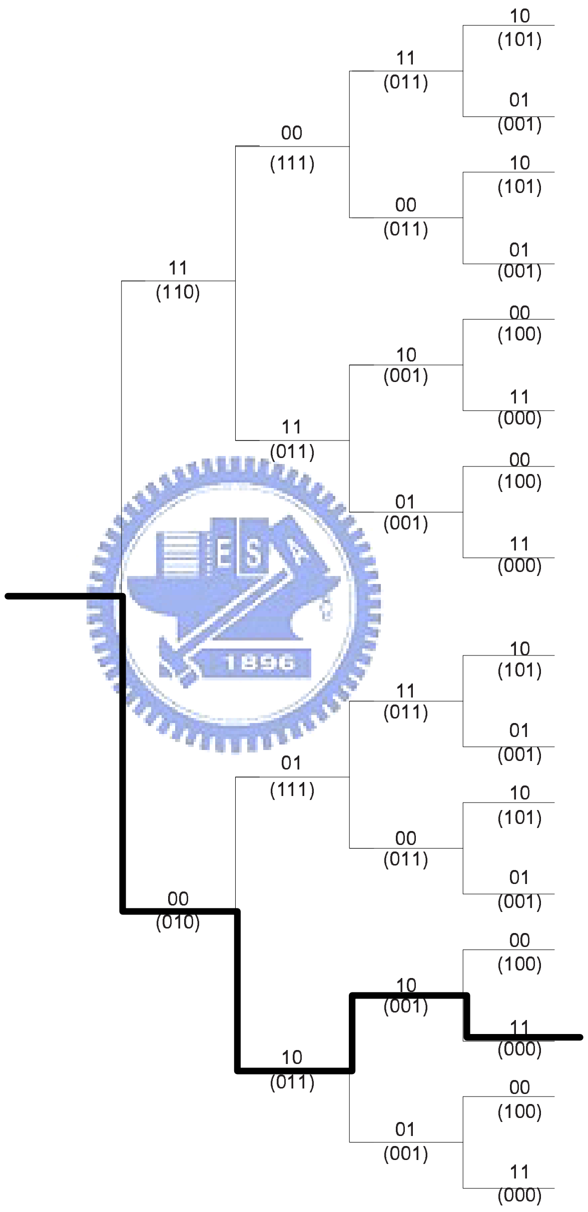


Figure 2.9: The code tree for a (2,1,3) stream cipher

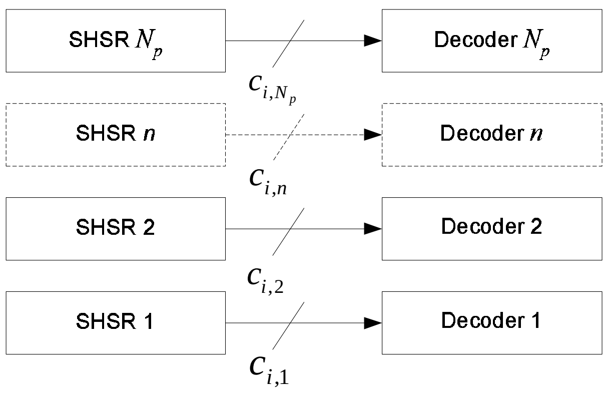
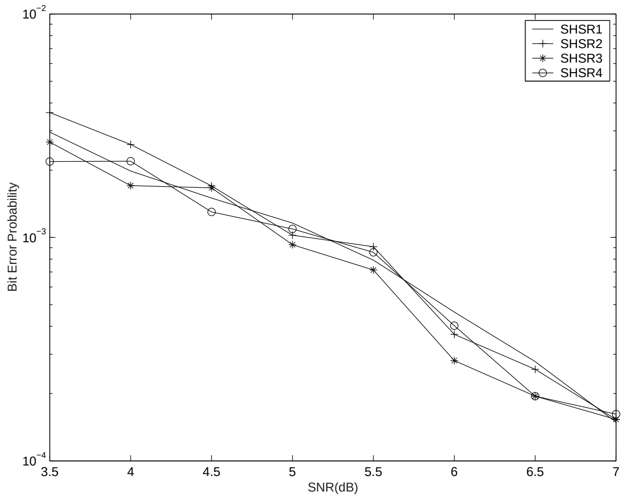
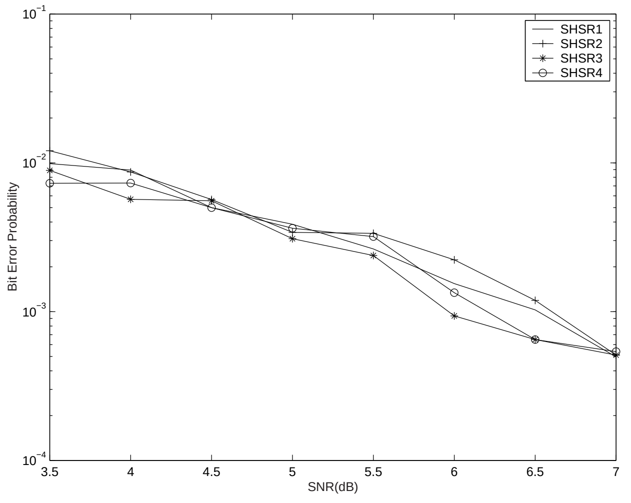


Figure 2.10: Each SHSR has a own decoder



(a) Case 1: The 1st register of SHSRs is unchangeable



(b) Case 2: The 1st register of SHSRs is changeable

Figure 2.11: The error probability on system (4,8,8,977)

Chapter 3

Secure MP3 Algorithm

Three new *partial/adaptive* encryption approaches to secure MP3 compression algorithm are presented. To adopt the proposed approaches, *a cryptosystem can be chosen to be embedded into or be concatenated behind the MP3 algorithm*, depending on the different characteristics of applications. The MP3 [27] is a popular format for audio distribution, hence securing MP3 will provide diversified applications, such as music trial services, authorized access, and multilevel encryption. To encipher a compressed file by partial encryption, the encrypted bits will be diffused after decompression, hence the partial/adaptive encryptions are suitable for multimedia security. In this work, the audio data are separately enciphered by these proposed approaches: *sign bits of frequency magnitudes, Huffman codes* and *side information*, and the results are analyzed in *Masking to Noise Ratio* sense. *The proposed secure MP3 algorithm can be easily achieved without extensive computation, major modifications for MP3, and loss of the compression ratio.* Moreover, the partial encryption can exactly provide enough security on multimedia application. And the encrypted MP3 files is compatible to MP3 standard for trial service.

Moreover, for portable devices with MP3 codec, the demands of digital right management arise recently. To provide a secure scheme to the most portable devices with MP3 codec, the approach described above is modified to be efficiently implemented it on a dual-core system with one DSP and one RISC. The secure MP3 algorithm is a combination of a MP3 algorithm and a stream cipher. The MP3 algorithm is executed on DSP and the stream cipher is on RISC. This separated design can dynamically update the type of stream ciphers in various applications. However, only the main data of a MP3 frame, rather than sign bits and side information, is encrypted in the modified approach.

This chapter is organized as follows. The overview of security of MP3 is given in Section 3.1. Section 3.2 gives brief descriptions on *Masking to Noise Ratio (MNR)*, MP3, and *Software encryption algorithm (SEAL)* and the definitions of *partial encryption* and *adaptive encryption*. The three proposed schemes and their performance are dilated on in Section 3.3. The simulation results are shown and explained in Section 3.4. In Section 3.5, the realization issues, security of three algorithms, and solutions for trial service are described here. Section 3.6 briefly describes the modified secure MP3 scheme, describes the scheme and analyzes the security thereof. The performance of the modified approach is shown in Section 3.7. Finally, Section 3.8 summarizes this work and provides directions for future work.

3.1 Overview

The digitization of media has profoundly affected copyright and intellectual property. Online MPEG Layer III (MP3) sharing seems to threaten the music industry. Accordingly, topics in the area of *Digital Rights Management* (DRM) have become increasingly important over recent years. The aim of DRM is to solve the problems of the distribution of digital content. The access to content produced using DRM is restricted in several ways, including encryption, watermarking, finger-printing, mechanism of access control and others. Currently, selling music is the most popular DRM application, as done by Apple's iTunes [1], iMUSIC [26], and others. The details of DRM systems of music industry vary from implementation to implementation. However, the encryption is a method to limit the access of the protected music. First, the content is encrypted using a media key. Then, the encrypted content, the encryption key, the copyrights and information about the content are packaged and encrypted once more using a license key. After the consumer receives the digital product and pays for the license key, the player decrypts and undo the package to yield the encrypted content and the encryption key. Then, the player can fully access the encrypted content if all of the information is correct.

Some studies of the encryption of MP3 have been published. Torrubia et al. [61] presented the perceptual cryptography of MP3 streams. They employed two primitives — scalefactor encryption and Huffman-codeword substitution. Torwirth et al. [60] presented a selective encryption algorithm, that encrypts the main data of MP3 granules. The encrypted part is determined by mapping the byte index of Huffman codeword onto the exact frequency boundaries. Both schemes can be used to encrypt the already encoded MP3 files. However, both schemes involve extra computations to determine accurately the quality of the encrypted MP3 files. This work presents a simple method for adaptively encrypting the main data in MP3 frame, and yields similar results to those of Torrubia et al. [61] and Torwirth et al. [60]. The security level can be varied from 0%(lowest security) to 100%(highest security). Any stream ciphers can be adopted. Therefore, any stream cipher can be employed to generate the random bitstreams. The advantage of the partial encryption/decryption is that it accelerates processing overall. Additionally, the encryption scheme and decryption scheme are identical, so only one security scheme is required to perform both encryption and decryption. Following encryption, the format of MP3 frame remains valid. Therefore, the MP3 algorithm could directly decompress the encrypted MP3 without decryption, but the consumers receive only the low-quality music. Content providers can use this feature to provide free music to consumers. Gang et al. [18] concluded that MP3 files can be encrypted in compression progress or after compression.

Usually, multimedia encryption is just to cascade a cryptosystem behind a source encoder. Indeed, the multimedia encryption has been especially designed for practical applications. Source coding is data-dependent and compacts the data size; however, secrecy coding is data-independent and keeps the data size. Because of the different natures between source coding and secrecy coding, the secrecy-source coding is hard to design. At general points, designs of cryptosystems do not consider the properties of data. In fact, the characteristics of the the multimedia content have to be taken more consideration to achieve more efficiency on processing and more flexibility on applications while applying a cryptosystem on a compression algorithm. The diffusion, which is an important feature for substitution-permutation ciphers, also appears on decompression, hence the encrypted bits will extensively infect the decompressed multimedia content, i.e., the effects on quality of partial encryption may be as well as of full encryption. Additionally, the partial encryption can process less data and is suitable in real-time applications.

Our schemes can be implemented into two cases: simultaneously encrypting and compressing or encrypting the already encoded MP3 files. For live applications, which the former case is fit, the media streams are encoded then are delivered immediately. Since it is a time critical issue, so it would be better to do encryption and compression simultaneously. For other applications, such as MP3 providers, because the compression is time-wasting processing, it would be fine enciphering compressed files than doing encryption in compression step. In this research, we propose several approaches to secure MP3 to meet the above cases. The first approach is the *sign-bit encryption*: the sign bits of frequency magnitudes are treated as plaintext. The second method encrypts the Huffman codes of quantized frequency magnitudes. Both above schemes are applicable to adaptive encryption. In adaptive encryption, the information must be recorded in headers for receivers, hence we devise a way of recording the extra information to be compatible the MP3 standard. The last approach enciphers the side information in MP3 headers.

However, most playback devices are dual-core systems, DSP and RISC, so the proposed approach is modified for being implemented and accelerated on such a system. In the modified approach, only *Huffman-code encryption* is chosen as the security phase to execute on RISC, and the MP3 algorithm is on DSP. The stream cipher SEAL is chosen in our schemes to perform encryption, but not limit which ciphers to be adopted for our approaches. The security phase executes parsing and encryption/decryption of the XOR operation. The MP3 phase performs as does the MP3 algorithm.

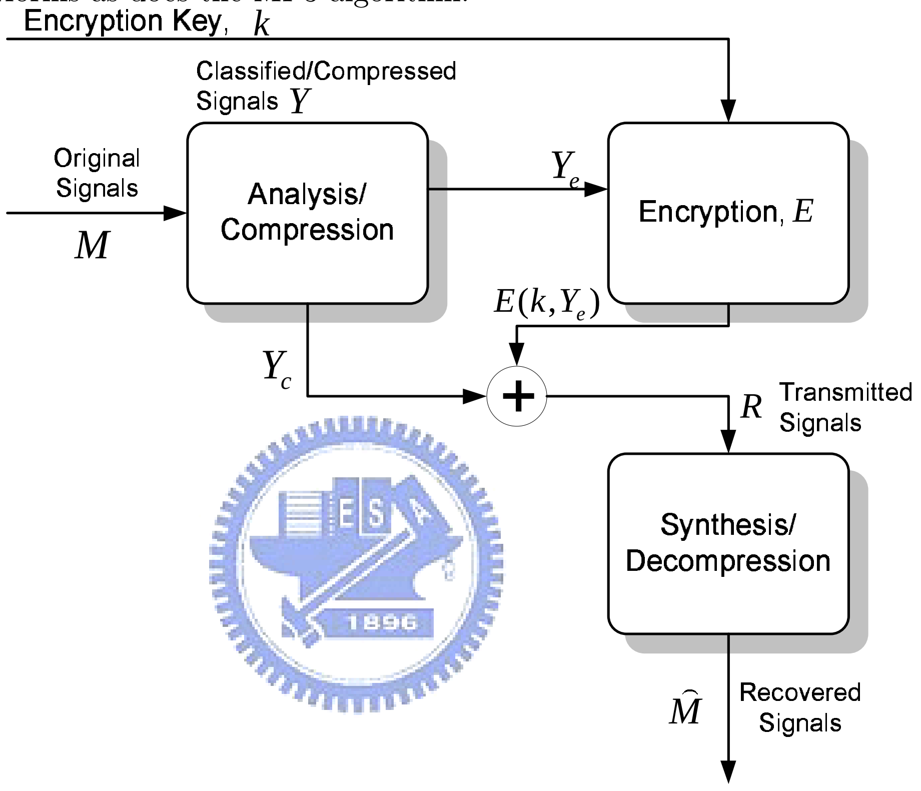


Figure 3.1: The model description of *partial/adaptive* encryption

3.2 Preliminaries

3.2.1 Partial Encryption and Adaptive Encryption

The *partial/adaptive* encryption is a particular derivative from the combination of a cryptosystem and a source coding or an error control coding. The main concept of the *partial/adaptive encryption* is to protect the entire content by only encrypting the significant part which has smaller size. Generally, the encryption is behind compression. Hence, as shown in Fig. 3.1, the original signal M is processed by filters or compression algorithm to obtain Y. The significant part can be determined among the generation of Y, and we denote it as Y_e, the part to be encrypted. Therefore, in the *partial/adaptive encryption*, Y is divided into an encrypted part Y_e and a clear part Y_c. The transmitted signal R is the combination of Y_c and $E(k, Y_e)$. Signal $\hat{M}$ is the reconstruction of R without decryption, and we can compare M and $\hat{M}$ to examine how the influence $E(k, Y_e)$ has. A scheme which makes some cumbrance on sensible presentation and causes that the polluting size of $\hat{M}$ is larger than the size of Y_e is called *partial/adaptive* encryption, as shown in Fig. 3.2.

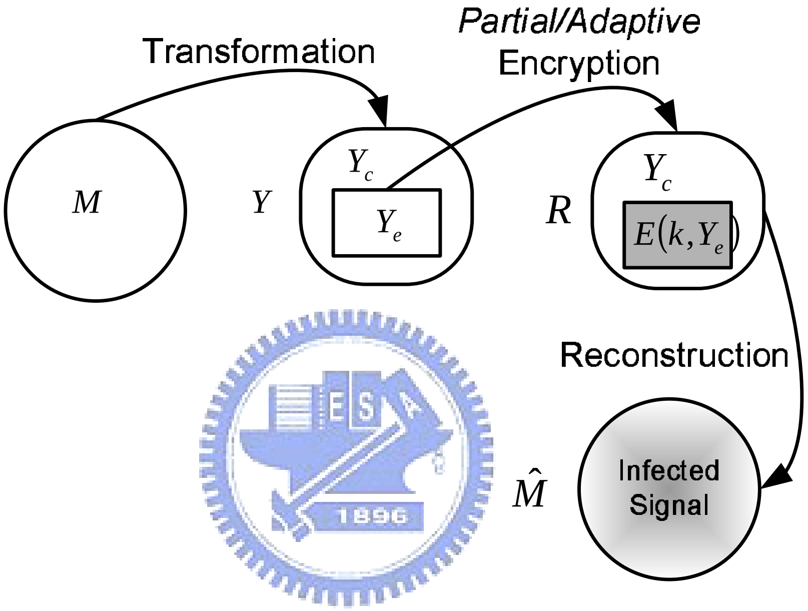


Figure 3.2: The data flow of *partial/adaptive* encryption

Both schemes are similar in the concept of encryption, but not in the applications. Hence we give definitions of *partial encryption* and *adaptive encryption* below.

Definition 2 Partial encryption *is an encryption method which brings about the larger size of infected part of $\hat{M}$ than the size of Y_e and some cumbrance while displaying $\hat{M}$,* but the quality of $\hat{M}$ is too troublesome to be determined by the size of Y_e.

Definition 3 Adaptive encryption *is an encryption method which brings about the larger size of infected part of $\hat{M}$ than the size of Y_e and some cumbrance while displaying $\hat{M}$,* but the quality of $\hat{M}$ can be determined by the size of Y_e in a systematic way.

Both schemes have a common property that the complete reconstruction can't be gained without decryption or authorized access. The *partial/adaptive encryption* relies upon how to select Y_e. The only principle for Y_e selection is to pick the M-dependent data to encrypt. There are two types of the M-dependent data.

1. Variant of M: This kind of data can be truly regarded as signal M of other formats, in other words, it is just some transformation of M, for examples, the results of *Discrete Cosine Transform* (DCT), or *Huffman coding* of M.

2. Accompaniment of M: The input-dependent and additional information, which receivers needed to decode, are belong to this type, for examples, *Huffman* table, CRC code, and frame headers.

Generally, the *partial encryption* encrypts the later type and the *adaptive encryption* takes the other.

The MP3 algorithm has both pre-described types of M-dependent data; actually, most applicable algorithms do. Hence our proposed schemes include both *partial encryption* and *adaptive encryption*.

3.2.2 Masking-to-noise ratio

We use *MNR* to analyze the audio quality after MP3 compression and encryption. *MNR* can be obtained from the masking model of human hearing, so it's better than *SNR* to represent the audio quality. The *MNR* is defined in (3.1).

$$MNR = 10\log_{10}\left(\frac{power\_of\_masking\_threshold}{power\_of\_noise}\right). \tag{3.1}$$

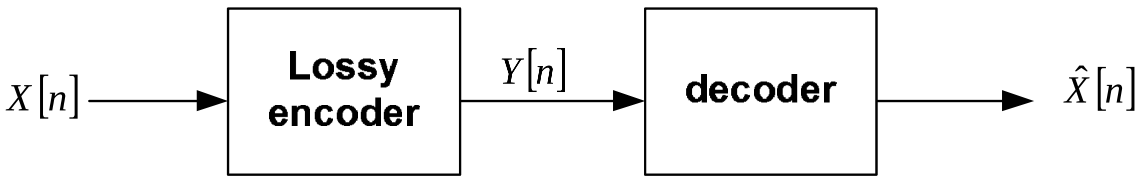


Figure 3.3: The general model of a lossy compression

Fig. 3.3 shows the general diagram of a lossy codec. An original signal $X[n]$ is compressed as $Y[n]$, then the error between $X[n]$ and $\hat{X}[n]$ is defined as

$$E[n] = X[n] - \hat{X}[n].$$

To have *MNR*, the *masking threshold* is calculated from the results of feeding *psychoacoustic model* with $X[n]$ and the noise energy is computed by $E[n]$. When *MNR* is greater than zero, human hearing is hard to detect the noises; more exactly, noises are masked. For a compressed audio, each frequency band has its own *MNR*. For convenient comparison, we use the mean value of *MNR*s to represent the audio quality.

3.2.3 MP3 algorithm

The MP3 [27] algorithm is firstly introduced. Fig. 3.4 presents a block diagrams of the general MP3 encoding process. A time-to-frequency mapping converts the audio input into spectral lines frame by frame. In the hybrid transformation block, MP3 uses a poly-phase filter bank followed by a *Modified Discrete Cosine Transform* (MDCT) to increase the spectral resolution. These spectral components are then divided into several scalefactor bands, according to the critical-band rate. The audio input simultaneously passes through the PAM-II, that determines the ratio of the signal energy to the masking threshold for each scalefactor band.

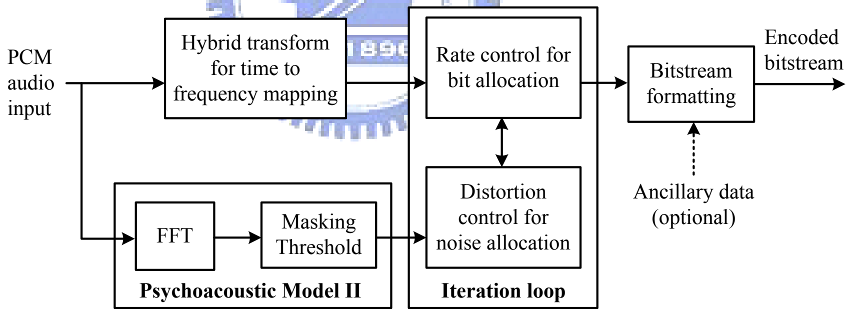

PCM audio input
Hybrid transform for time to frequency mapping
Rate control for bit allocation
Bitstream formatting
Encoded bitstream
Ancillary data (optional)
FFT
Masking Threshold
Psychoacoustic Model II
Distortion control for noise allocation
Iteration loop

Figure 3.4: MPEG/audio encoding process.

The rate controller varies the quantizer in an orderly way: quantizes the spectral values and counts the number of Huffman code bits required to code the quantized values, to satisfy the bit rate constraint. The quantizer in MP3 is non-uniform. In the quantization of the g^{th} granule in the f^{th} frame, the spectral value $x_{f,g}(i)$ is pre-emphasized and amplified by applying (3.2) and (3.3).

$$x'_{f,g}(i) = x_{f,g}(i) \times \sqrt{2}^{z_2 \times (1+z_1) \times P(b_i)}, \tag{3.2}$$

$$x''_{f,g}(i) = x'_{f,g}(i) \times \sqrt{2}^{(1+z_1) \times C(b_i)}, \tag{3.3}$$

where i is the index of the spectral line; $z_2 \in \{0,1\}$ switches on or off the pre-emphasis; $z_1 \in \{0,1\}$ determines whether the scalefactors are logarithmically quantized with a step size of 2 or $\sqrt{2}$; b_i is the scalefactor band of the i^{th} spectral line; $P(\cdot)$ is the preemphasis table as defined in [27], and $C(\cdot)$ is the scalefactor of all scalefactor bands. Then, the processed spectral value $x''_{f,g}(i)$ is quantized by

$$y_{f,g}(i) = \texttt{nint}\left(\left(\frac{|x''_{f,g}(i)|}{2^{\frac{\delta+q}{4}}}\right)^{0.75} - 0.0946\right), \tag{3.4}$$

where `nint` is the rounding function; q is the lower bound of quantization parameter, and δ is the increasing variable of quantization parameter.

Huffman coding is applied as the lossless coding tool and Huffman tables are predefined in [27]. MP3 also uses scalefactors to amplify the spectral band energy when the quantization noise exceeds the masking threshold. The distortion controller determines the scalefactors that control the quality. Finally, the information required by the decoder is packaged with compressed audio data as a valid stream of MP3 stream.

The MP3 decoding process comprises three main parts [27] — bitstream decoding, dequantization and frequency-to-time mapping, as shown in Fig. 3.5. Bitstream decoding synchronizes encoded bitstream inputs, and extracts the quantized frequency coefficients and other information about each frame. Fig. 3.6 depicts the detail functional blocks.

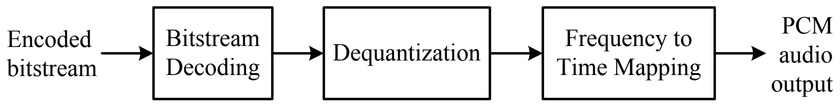


Figure 3.5: Blcok diagram of MPEG/Audio Layer 3 decoding.

Dequantization reconstructs the frequency coefficients, which are perceptually identical to those during encoding. The dequantization calculation based on the output of Huffman decoding and scalefactor information is given by (3.5) [27].

$$x_{f,g}(i) = (-1)^{\mathbf{s}(i)} \cdot y_{f,g}(i)^{\frac{4}{3}} \cdot \frac{2^{\frac{1}{4}(\Delta_{f,g}-8\Delta_s(w_i))}}{2^{(1+z_1)\cdot(C(b_i)+P(b_i))}} \tag{3.5}$$

where $\mathbf{s}(i)$ is the sign bit of $y_{f,g}(i)$; $\Delta_{f,g} = \delta+q$ is the step size of the nonuniform quantizer; w_i is the short-block window of the corresponding i^{th} spectral line, and $\Delta_s(w_i)$ is the pre-defined gain of the short-block window.

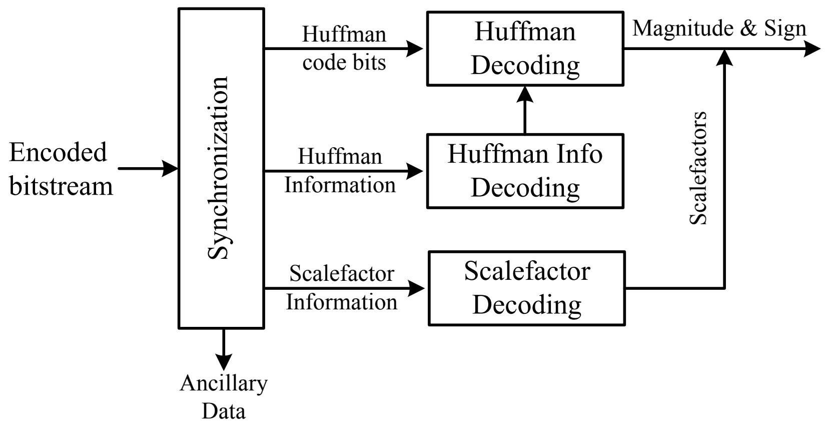


Figure 3.6: Bitstream decoding.

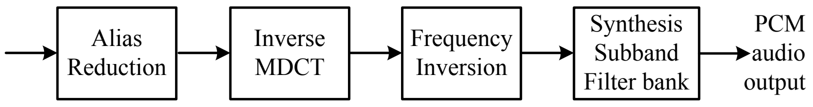


Figure 3.7: Frequency-to time mapping

The final part shown in Fig. 3.7, frequency-to-time mapping produces an audio PCM output from the dequantized coefficients. This part includes a set of reversed operations of the MDCT and analysis subband the filter bank in the encoder. The alias reduction block adds alias artifacts to dequantized coefficients, to reconstruct the data approximately as those of analysis subband filter bank in encoder. Then, the inverse MDCT reconstructs time domain subband signals from frequency lines. The frequency inversion is then applied in order to compensate the decimation used in the analysis polyphase filterbank. Thereafter, the synthesis subband filter bank is applied to the subband signals to yield the audio PCM output.

Of the above procedures, dequantization, IMDCT, and subband synthesis in particular, depend on numerous arithmetic operations, and produce quantization noise in fixed point implementation. This work describes the optimization of these three processes.

3.2.4 SEAL

The SEAL [56] is a software optimized stream cipher with key length of 160 bits. For the case of the processors with eight registers, SEAL can keep the number of involving variables less than 8. Most operations in SEAL only have two operators and the table size is less than 4K bytes, which is especially designed for the processors with small on-chip cache. SEAL uses *secure hash algorithm* (SHA) [46] to generate three tables, which are needed by initialization and key generation. According to Table 3.1, SEAL is more efficient than most stream ciphers and that's why we apply it in our work.

Table 3.1: Performance of several ciphers on Intel Pentium processor [56]

Algorithm	Mbit/s	Relative speed
SEAL	198	1
RC4	110	1.8
RC5-32/12	38.4	5.2
DES	16.9	11.7
MD5	133.1	1.5

3.3 Our Proposed Schemes

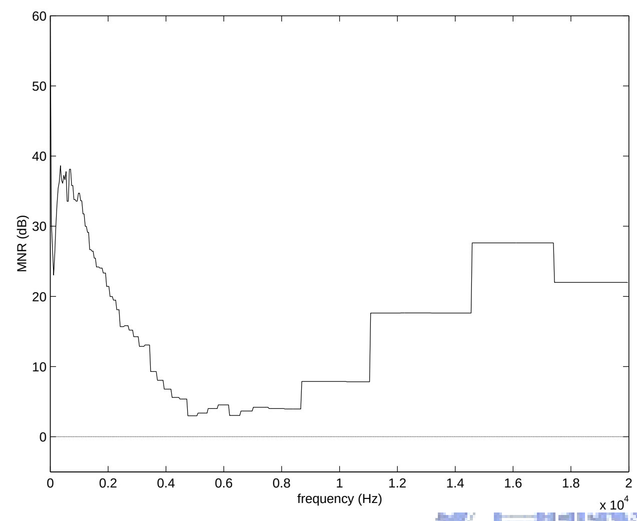
Based on the designed principle and definitions of *partial encryption* and *adaptive encryption*, we proposed three approaches, *sign-bit encryption, Huffman-code encryption,* and *side-information encryption,* to secure MP3 algorithm. All proposed schemes can be applied by embedding a cipher into MP3 encoder or using a cipher to straight encrypt the MP3 files.

3.3.1 Sign-bit encryption

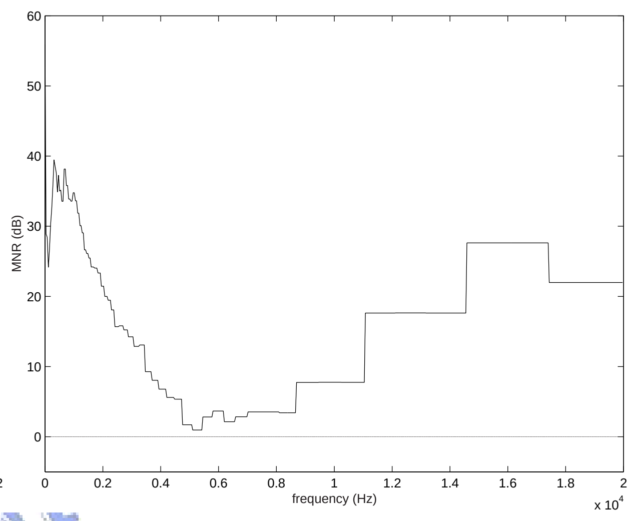
In MP3, the frequency magnitude for each sample has two parts: an absolute value and a sign bit. When the value of the sample is less than 0, the sign bit is set to 1; otherwise, it is set to 0. The human hearing is sensitive to the variation on sound stemming from changing sign bits of samples below 5 kHz [19, 20], hence the audio quality is dropped significantly while encrypting those sign bits. Encrypting the sign bits above 5 kHz is not easy to be observed by audiences, though the energy is twisted. According to the properties discussed above on sign bits, that encrypting sign bits could achieve the *adaptive encryption* is obvious.

For each granule in MP3, there are 576 samples with equal bandwidth, and we denote the set of sign bits of i^{th} granule as $S_i = \{s_{i,j}|$ the sign bit of the sample j, $0 \leq j \leq 575$, at the

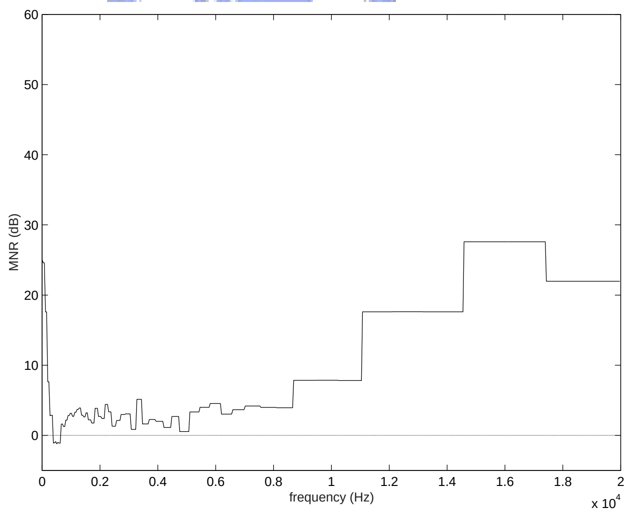
i^{th} granule}. The encrypted part of i^{th} granule, $Y_{e,i} = \{s_{i,k}|\ s_{i,k} \in S_i$ and $a \leq k \leq b$, where $[a\ b]$ is the interval of granule to be encrypted }. If the full-band encryption is selected, we encrypt the entire set S_i, i.e., the special case of $a = 0, b = 575$; otherwise, we can adjust the music quality by selecting which bands to encrypt, while the encrypted MP3 files are directly played by original MP3 player.



(a) Average *MNR* is 29.2864dB



(b) Average *MNR* is 28.9732dB



(c) Average *MNR* is 20.6121dB

Figure 3.8: The simulation of sign-bit encryption on different frequencies: (a)The original MP3 file; (b)The sign bits above 5kHZ are encrypted; (c) The sign bits below 5kHz are encrypted

Fig. 3.8 shows the *MNR* distribution in several cases. In Fig. 3.8(b), *MNR*s are greater than zero for all bands, but drop slightly at frequency about 5 kHz, as compared with

Fig. 3.8(a). But Fig. 3.8(c) reveals the poor performance in *MNR*s within the frequency range from 0 to 5k Hz, even having some *MNR*s smaller than zero. As opposed to Fig. 3.8(a), that decrements of *MNR* in Fig. 3.8(c) is more than the ones in Fig. 3.8(b) reveals the the fact that human hearing is sensitive to the variation stemming from changing sign bits of samples below 5 kHz[19, 20].

3.3.2 Huffman-code encryption

In MP3 definition, each frame has four granules, each granule has 576 MDCT coefficients. The absolute values of these 576 coefficients will be divided into three regions: *big value*, *count one region*, and *zero region*, and be sequently compressed with four Huffman tables. We denote the Huffman code $W_i = \{w_{i,0}, w_{i,1}, \ldots, w_{i,575}\}$ as the whole Huffman codeword of 576 coefficients at the i^{th} granule, where $w_{i,j}$ is the Huffman codeword of the j^{th} quantized coefficient.

The Huffman code W_i is made of concatenating each codeword of samples sorted by frequency from low to high. However, when choosing some bits of W_i, that distinctly pointing out their corresponding frequency is hard. We can only know the information that the corresponding frequency of $w_{i,j}$ is lower than the one of $w_{i,k}$, when $j < k$.

Similarly to sign-bit encryption, the quality of unauthorized accessing can be adjusted by selecting the bytes of W to encrypt, so Huffman-code encryption is an *adaptive encryption*. Different from *sign-bit encryption*, an error avalanche occurs in *Huffman-code encryption*, because Huffman coding has an error propagation problem, even encrypting the first byte will cause tremendous errors in decompression.

The W_i is a variant of 576 MDCT coefficients, hence the encrypted part $Y_{e,i}$ can be determined from W_i as the plaintext in *Huffman-code encryption*. Because the bit length of $w_{i,j}$ is variable, not multiple of byte, it is inconvenient to pick a set $Y_{e,i} = \{w_{i,j} | w_{i,j} \in W_i$, and $a \leq j \leq b$, where $[a\ b]$ is the interval to be encrypted$\}$. Therefore, we select $Y_{e,i}$ in byte.

3.3.3 Side-information encryption

Side information records the data needed by MP3 encoder. There are several fields of side information. Some are media dependent and some are not. For security concern, we have to avoid known-plaintext attacks, hence much attention have to be paid on encrypting header-like data.

In our scheme, we choose those fields which changed by input media as plaintext. In MP3, all fields are classified into three kinds: *frame side information, channel side information,* and *granule side information.* Literally, *frame side information* is for each frame, and so as *channel side information* and *granule side information.* We choose some side information, of which symbols are defined in MP3 specification [27], to keep the *side-information encryption* from the known-plaintext attacks and list them below.

1. *Frame side information*: `main_data_bigin` indicates the beginning of compressed data, hence an MP3 encoder won't correctly decode without possessing this information.

2. *Granule side information*: These fields are the decoding information, such as *quantizer step*, *region boundary*, *Huffman table*, and so on.

   (a) `part2_3_length:` this value contains the number of main data bits used for scalefactor and Huffman code data.

   (b) `big_values:` the spectral values of each granule are coded with different Huffman code.

   (c) `global_gain:` the quantizer step size information is transmitted in the side information variable `global_gain`.

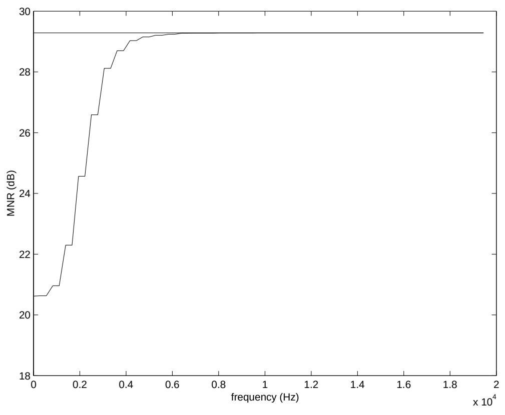
   (d) `scalefac_compress:` selects the number of bits used for the transmission of the scalefactors.

   (e) `region_address:` a further partitioning of the spectrum is used to enhance the performance of the Huffman coder.

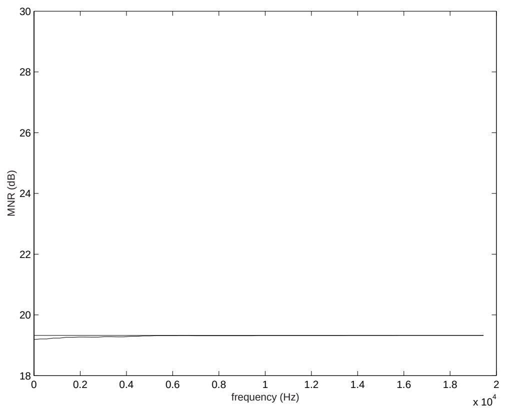
   (f) `table_select:` different Huffman code tables are used depending on the maximum quantized value and the local statistics of the signal.

Additionally, the *CRC* is encrypted necessarily to probably keep away from attacks, because *CRC* implicitly manifests some relationship among all fields in side information. All the listed fields are accompaniments of input signal, so *side-information encryption* is a *partial encryption.* It is obvious that the quality can not be systematically adjust by choosing the side information.

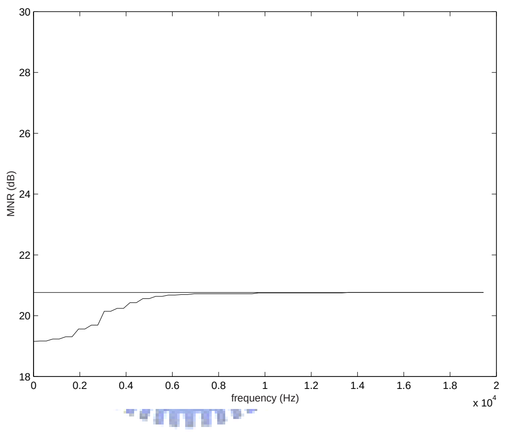
3.4 Simulation Results



(a) Piano



(b) Rock



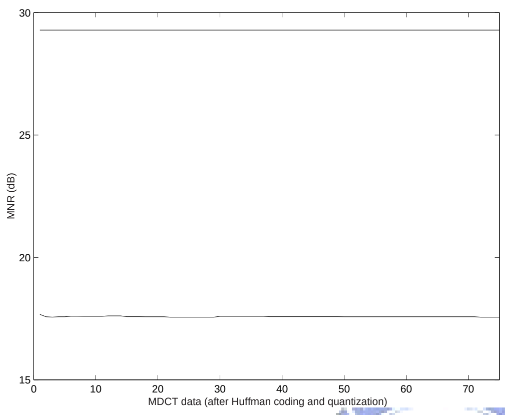
(c) Pop

Figure 3.9: The simulation of *sign-bit encryption* on different frequencies in three types of music, piano, rock, and pop. The horizontal lines are the *MNR* of original MP3 files.

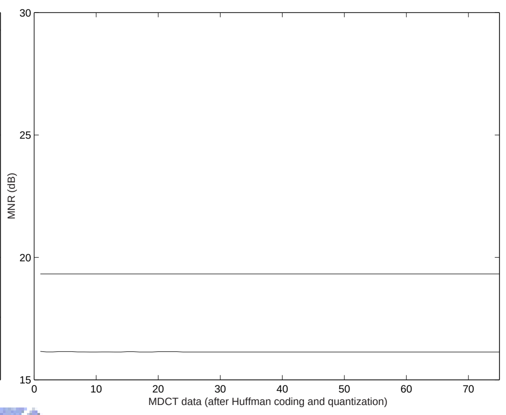
We use SEAL to encrypt and simulate the proposed schemes with three types of music: piano, rock, and pop. We discuss the results over the average *MNR*. The frequency distribution of piano, rock, and pop are $0 \sim 8$ kHz, $0 \sim 22$ kHz, and $0 \sim 16$ kHz, respectively.

In *sign-bit encryption*, we simulate with encrypting different bands. As shown in Fig. 3.9, when the encrypted frequency is increased, the audio quality is also improved. As we mentioned above, upon 5 kHz, the variation of sign bits will not vastly drop the quality. This simulation also tells one thing that *sign-bit encryption has tolerable quality with encrypting a section between* $4 \sim 6$ kHz *for music trial service.*

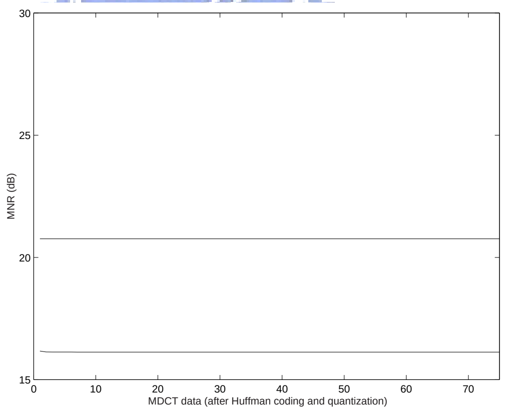
In *Huffman-code encryption*, because we can not exactly have the frequency information on Huffman codes, we use byte index instead of frequency index as x-axis. In spite of what



(a) Piano

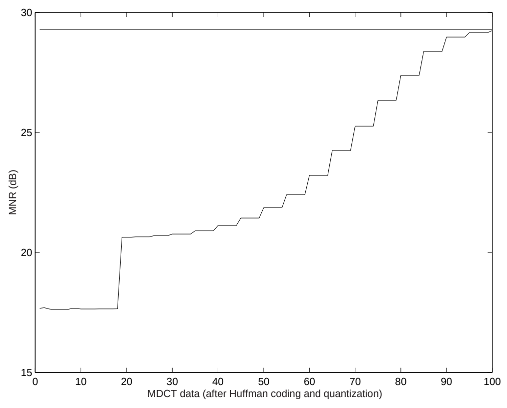


(b) Rock

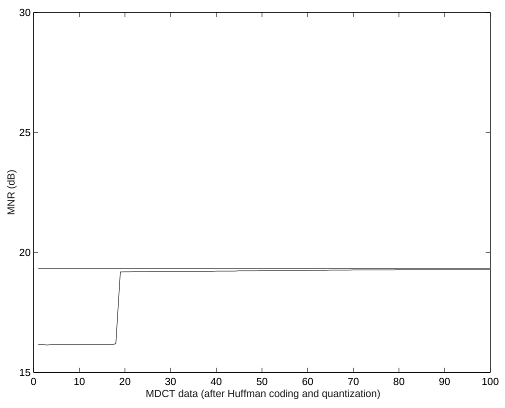


(c) Pop

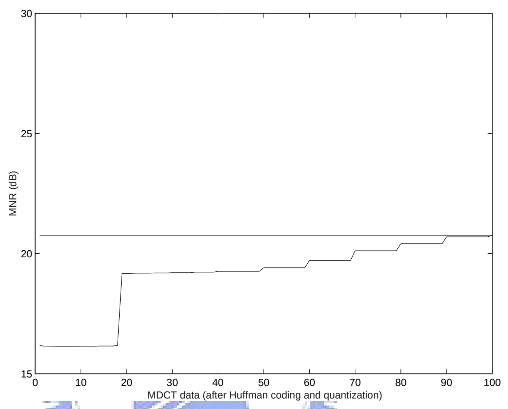
Figure 3.10: These results are obtained by encrypting Huffman codes. The x-axis is the number of encrypted bytes counted from the first byte of Huffman codes.



(a) Piano



(b) Rock



(c) Pop

Figure 3.11: These results are obtained by encrypting Huffman codes. The x-axis is the byte index of the encrypted byte.

kinds of music, from Fig. 3.10, we found no matter one byte or several bytes are encrypted, the quality of encrypted MP3 file are similar under *MNR* sense. The effects are resulted from the property of error propagation on Huffman decoding.

Different from the simulation of Fig. 3.10, instead of encrypting consecutive bytes, we separately encrypt each byte. The simulation results shown in Fig. 3.11 provide us that encrypting which byte is proper to music trial service. However, the *Huffman-code encryption* is hard to determine which frequency band is good for music trial service. The byte index only shows the relative information about frequency. For examples, even the 2^{nd} byte implies higher frequency than the 1^{st} byte in each granule, but the 1^{st} bytes of distinct granules do not represent the same frequency. The Huffman codes are the compressed data of nonzero region. For piano music, most of energy are concentrated in low bands and wide bandwidth in zero region, hence the quality of piano increases slowly while the frequency of encrypted band increases.

Table 3.2: *MNR* Comparison: Side information encryption on piano, rock, and pop.

	Non Encrypted	Frame side	Channel side	Granule side
Piano	29.3831dB	20.6328dB	23.6214dB	20.317dB
Rock	19.3537dB	19.1296dB	19.1816dB	19.1308dB
Pop	21.1033dB	19.2218dB	19.5985dB	19.1237dB

The results of side information encryption are illustrated in Table 3.2. From the *MNR* values, the MP3 files have been successfully encrypted. The important information are mostly in *frame side information* and *granule side information*, hence encryption on both of them provide good multimedia security.

For high security application, the *sign-bit encryption* encrypts the sign bits of samples below 5 kHz, the *Huffman-code encryption* takes $1^{st} \sim 70^{th}$ bytes of each granule as $Y_{e,i}$, and *side-information encryption* enciphers all side information. Table 3.3 lists the encrypted size for each case.

Table 3.3: Size of Y_e for High Security of three schemes

	Sign-bit encryption	Huffman-code encryption	Side-information encryption
$\frac{Y_e}{Y}$ (%)	15.56% (< 5 kHz)	19% (0 $\sim$ 20 bytes)	7.65% (256 bits)

3.5 Implementations and Security

Based on the simulation results, some practical issues and suggestions are addressed in following subsections.

3.5.1 Quality Level for Trial Services

According to the simulation results in Section 3.4, we suggest a quality level for music trial services for the both proposed *adaptive encryption*, *sign-bit encryption* and *Huffman-code encryption*, to original MP3 players. In *sign-bit encryption*, encryption of bands about $1,225 \sim 3,062$ Hz could obtain the music for trial service. For *Huffman-code encryption*, the quality for trial services could be provided by encrypting the $60^{th} \sim 90^{th}$ bytes of Huffman codeword in each granule. The results are shown in Table 3.4.

Table 3.4: The suggested significant part Y_e for trial service and its quality.

	Sign-bit encryption	Huffman-code encryption
Y_e	$1,225 \sim 3,062$ Hz	$60^{th} \sim 90^{th}$ bytes
MNR	~ 22 db	~ 23 db

If the audio quality is not highly concerned, the devised three schemes can fully encrypt the corresponding *encrypted parts* Y_e: sign bits, Huffman codes, and side information. If there exits the time issue of processing, the data size of Y_e can be reduced by selecting those parts Y_e from the low frequency bands to the high bands for *sign-bit encryption* or from low bytes to high bytes for *Huffman-code encryption*.

3.5.2 Extra Header Design

Taking inspection on *side-information encryption* firstly, because the quality adjustment is unsuitable, for realization, it is a good strategy that the encrypted side information is beforehand defined and known by encoders and decoders. So there doesn't need extra information to indicate how to decrypt with full quality, and standard MP3 players could decode the *side-information encryption* MP3 files as usual ones, but with low quality.

For *sign-bit* and *Huffman-code* encryptions, while the quality adjustment is available in encryption, saving the information of how many data are encrypted is necessary. In order to be compatible with the standard MP3 players, the information, needed for correct decryption, has to be added into standard MP3 header in a proper way so that the standard MP3 players can play the encrypted MP3 with low quality.

On *sign-bit* and *Huffman-code* encryptions, we can place the extra information prior to the *synchronization word*, because the MP3 decoder takes no consideration on the input before detecting the *synchronization word*, i.e., standard MP3 players can play the encrypted MP3 files as well. Without doubt, the music of full quality can be gotten under success authorization and specific MP3 decoders.

The *synchronization word* of hex is 0xFFF, hence the construction of extra information must have no probabilities to produce the pattern, 0xFFF. In *sign-bit encryption*, the encrypted part $Y_{i,e}$ can be described by two 10-bit digits ranging from 0 to 575, therefore the cases with the maximum length of consecutive 1's are {01111111111000xxx} in all possible combinations of the two digits. It is apparent that the pattern 0xFFF by no means occurs in extra information for *sign-bit encryption.* In *Huffman-code encryption*, we also use two 10-bit digits to define the bound of bytes to encrypt. Generally, the Huffman codeword size of a granule is less than 512 bytes, the most 1's case is {01111111110111111111}. For that reason, the extra information of *Huffman-code encryption* is certainly not to get the same pattern as *synchronization word.*

3.5.3 On-Line and Off-Line Encryptions

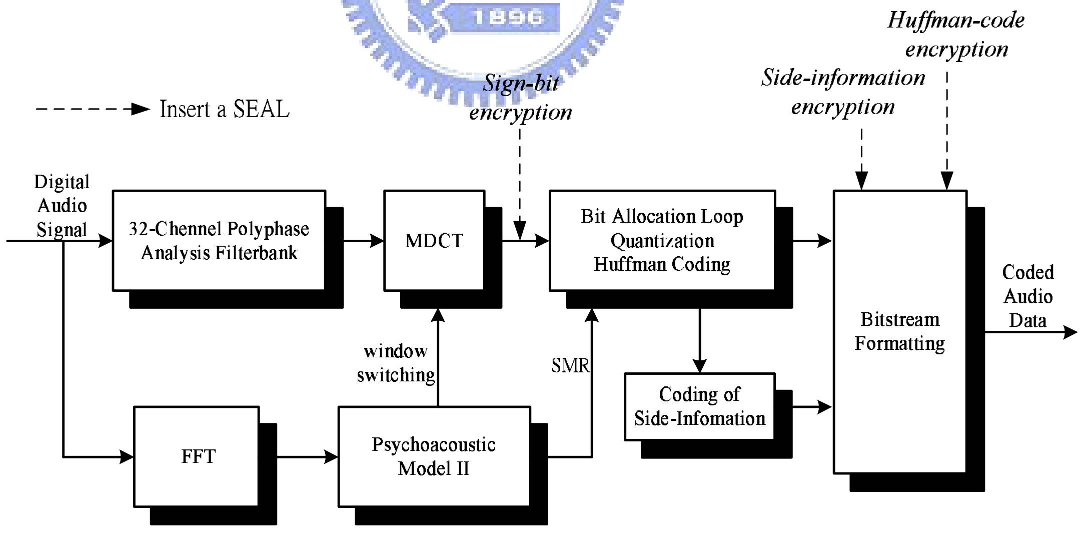


Figure 3.12: The block diagram of on-line cases. The *side-information* and *Huffman-code* encryption both are inside the *bitstream formatting* process. In terms of MP3 encoding procedure, the *side-information encryption* is ahead of *Huffman-code encryption.*

The proposed three schemes can on-line or off-line encrypt MP3 files according as the practical applications.In general, the on-line applications are time critical, such as live broad-

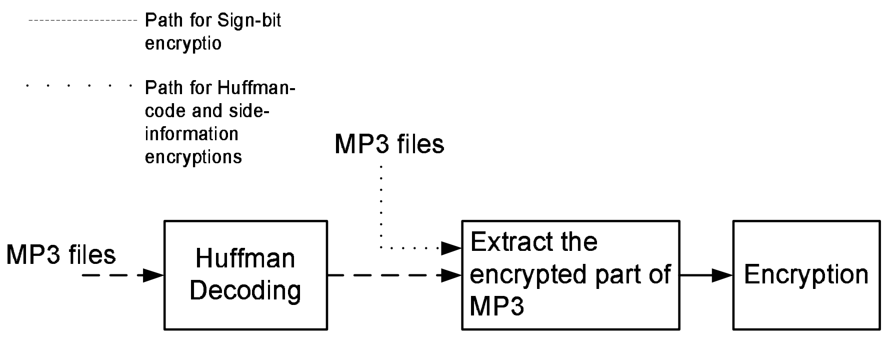

Path for Sign-bit encryptio
Path for Huffman-code and side-information encryptions
MP3 files
MP3 files
Huffman Decoding
Extract the encrypted part of MP3
Encryption

Figure 3.13: The block diagram of off-line cases.

casts of sport games, and concerts, so it's better to produce encrypted MP3 files by simultaneously encrypting and compressing the raw data. And the off-line instances, therefore, can be done by directly encrypting the ready MP3 files.

The on-line cases are composed of MP3 algorithm and SEAL, each case has different insertion position of SEAL as shown in Fig. 3.12. The off-line cases have to extract the encrypted part Y_e from the ready MP3 files, then encrypt Y_e. The encrypted part Y_e of *side-information* and *Huffman-code* encryptions are directly extracted from each frame of a MP3 file. In off-line *sign-bit encryption*, however, the sign bits to encrypt are obtained after Huffman decoding of each frame. The pictorial description is at Fig. 3.13

3.5.4 Security

Apparently, the proposed schemes are cipher independent, hence the security on our schemes are dependent on what kind of data we encrypted. For this concern, we have noted whether the encrypted data are predictable or known already by cryptanalysts. For *sign-bit encryption*, cryptanalysts are not easy to get or predict the sign bit of each sample. In *Huffman-code encryption*, the compressed data are the variant of the original signal, hence it has similarly intrinsic property as *sign-bit encryption*, that is, the compression results are a huge alphabet to guess. However, some fields of side information are still and simple to predict. For security concern, we have to avoid encrypting those fields and keep our side-information encryption from known-plaintext attacks.

3.6 The Overview of Modified Scheme

Fig. 3.14 is the overview of the proposed scheme. The proposed scheme has two phases, the secure phase and the MP3 phase.

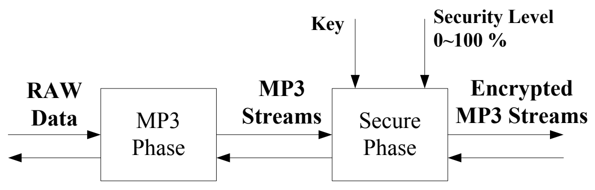


Figure 3.14: Block diagram of the coding flow.

The secure phase parses the MP3 frame to identify the security part and the normal part, encrypts/decrypts the security part, and joins normal part to processed security part as a valid MP3 frame. The security level s can be varied from 0% to 100%. The percentage is mapped onto the size of the security part. Fig. 3.14 indicates that the security phase has three inputs.

1. Key;
2. Security level;
3. A normal MP3 frame or an encrypted MP3 frame.

The MP3 phase executes a low-complexity MP3 algorithm, sends the ordinary MP3 frames to the security phase and decompress the decrypted MP3 frames from the security phase.

For on-the-fly production, the two-phase scheme increases the efficiency in dual-core platform, because we can run the two phases on two distinct CPUs for executing the low-complexity MP3 algorithm and secure coding simultaneously. Additionally, the proposed scheme can encrypt the existent MP3 files directly by security phase. The security phase reads the MP3 frames and processes the security parts. This makes the scheme easily be applied to the actual state.

Torrubia et al. [61] and Torwirth et al. [60] presented selective encryption algorithms for audio compressing. In the case of encrypting an encoded MP3 file, Torrubia et al.

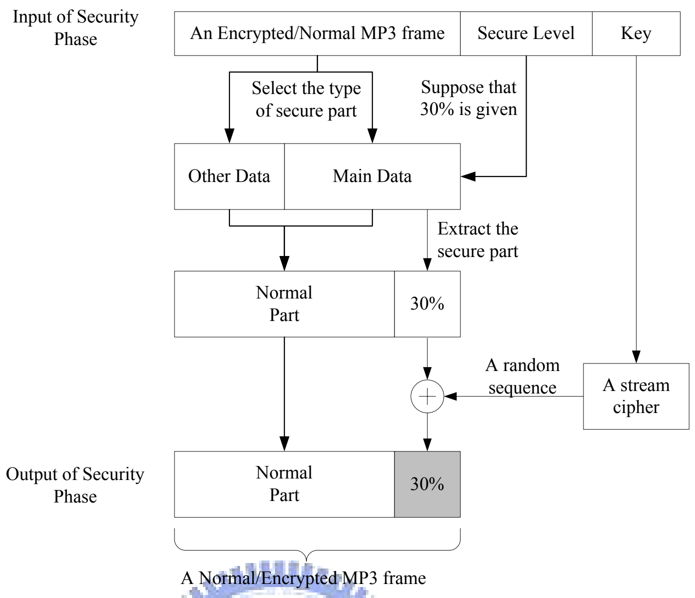


Figure 3.15: Detail of processing flow of the security phase.

[61] decompressed an MP3 file and performed Huffman encoding using a secure table to substitute codeword; Torwirth et al. [60] also decompressed an MP3 file; determined the exact frequency boundary of main data in MP3 frame, and encrypted the protected part. These tasks are all time-consuming. However, the proposed scheme uses a more simple encryption scheme than described above, yielding similar results.

Fig. 3.15 presents how the security phase works. Each MP3 frame has two or four granules for one channel or two channels, respectively. In the security phase, each granule is sequentially processed in the same way. The flow of the security phase is as follows:

1. Determine the size of the security part with given security level s, where $0 \leq s \leq 1$ and s has only two digits after decimal point. The security level for each MP3 granule is fixed. However, in each MP3 granule, the size of the security part is $S = s \times |\text{main data}|_b$, where operation $|\cdot|_b$ counts the bit length.

2. Obtain a sequence of S successive bits from the output of stream cipher. The stream cipher in proposed scheme may be any secure one. The stream cipher generates a

random sequence of appropriate length in advance, whereas the MP3 phase performs compression to accelerate the entire process.

3. XOR the security part with the sequence.

4. Join the security part to the normal part to form a valid MP3 granule.

Fig. 3.16 depicts the file format of MP3. The header field contains information about the sampling frequency, bit rates and audio modes, for example. The *CRC* is used to detect whether errors occurred in fields of header and in the side information. All parameters related to decoding information are in the side-information field. Finally, the main-data field contains the compressed audio data. The security part is backwardly extracted successive S bits from the last bit (highest frequency) of main data. However, the length of main data is not fixed, but the header of each frame includes this information. Data at lower frequency are generally more important than those at higher frequency. Therefore, when the security level s increases, the security part becomes more important. Additionally, the encrypted MP3 file has the same size as the one without encryption, because the encryption part is the final results of MP3 encoding for a granule.

Header (32 bits)	CRC (0,16 bits)	Side information (136,256 bits)	Main data

Figure 3.16: File format of MP3 [28].

Fig. 3.15 reveals that the encryption is performed by XOR-ing the pseudo-random binary sequence with the security part of main data. The decryption process is identical to the encryption process. The encrypted MP3 frame is parsed by the same function as applied in encryption, to determine the normal part and the security part, according to the specified security level. The stream cipher generates the identical pseudo-random binary sequence used in encryption. Therefore, one copy of the flow depicted in Fig. 3.15 is involved in both encryption and decryption.

3.7 Performances and Comparisons

3.7.1 Performance of partial encryption

Quality loss is addressed using the *Objective Difference Grade (ODG)* of PEAQ (Perceptual Evaluation of Audio Quality) [29]. The *software encryption algorithm* (SEAL) [56] is applied as the stream cipher in the following simulation. The three audio samples, bass, harp and spfg, are obtained from EBU SQAM [39]. Fig. 3.17 shows the ODG of the encrypted MP3 files, obtained by the proposed approach with different security levels. The quality clearly monotonically declines as the security level increases.

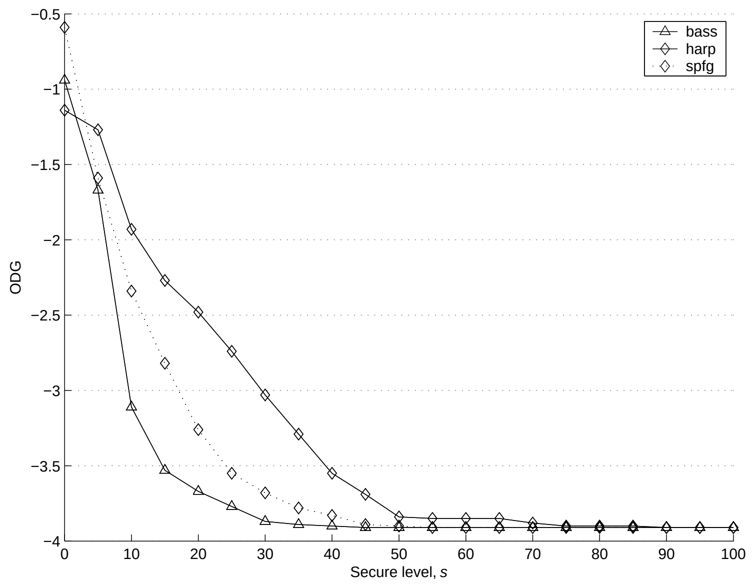


Figure 3.17: Quality loss following encryption using SEAL stream cipher

3.8 Conclusions

We propose several approaches to secure MP3 in this research. They provide an easy solution to integrate the cryptosystem and MP3 algorithm. If the *adaptive encryption* is needed, the *sign-bit encryption* and *Huffman-code encryption* can fit in with, along with good security. When the size of encrypted data are fixed, the *side-information encryption* provides good security than the others. Our schemes can be applied on simultaneously encrypting and compressing or encrypting the already encoded MP3 files. For live applications, e.g., diverse sport shows, and concerts, the former case is fit, because the media streams are encoded then are delivered immediately; otherwise, because the compression is time-wasting processing, it's better enciphering compressed files than doing encryption in compression step. The *partial/adaptive* encryption and the selection principle of Y_e addressed in Section 3.2 are applicable not only on MP3 algorithm but also other compression algorithms. Then, a modified one of *Huffman-code encryption* is proposed on a dual-core platform with RISC and DSP. It represents faster and simpler structure than the original one.

Chapter 4

Enhancement of the Application Security of AES

This chapter addresses two security issues of *Advanced Encryption Standard* (AES) [47]. First, in order to prevent AES from the *differential fault attacks* (DFA) [7, 14, 53], error detection is required to detect the errors during encryption or decryption, and then to provide the information for taking further actions, such as interrupting the AES process or redoing the process. Because errors occur within a function, it is not easy to predict the output. Therefore, general error control codes are not suited for AES operations. In this work, several error-detection schemes have been proposed. These schemes are based on the $(n+1, n)$ *cyclic redundancy check* (CRC) over $GF(2^8)$, where $n \in \{4, 8, 16\}$. Because of the good algebraic properties of AES, specifically the `MixColumns` operation, these error detection schemes are suitable for AES and efficient for the hardware implementation; they may be designed using round-level, operation-level, or algorithm-level detection. The proposed schemes have high fault coverage. In addition, the schemes proposed are scalable and symmetrical. The scalability makes these schemes suitable for an AES circuit implemented in 8-bit, 32-bit or 128-bit architecture. Symmetry also benefits the implementation of the proposed schemes to achieve that the encryption process and the decryption process can share the same error detection hardware. These schemes are also suitable for encryption-only or decryption-only cases. Error detection for key schedule in AES is also proposed and is based on the derived results in the data procedure of AES.

In addition, we also implement a parameterizable Rijndael algorithm with three changeable coefficients, including the irreducible polynomial, affine matrix, and the row vector of the matrix used in `MixColumns`. The coefficients are on-line changeable, i.e., they are also the inputs of the circuit. For increasing the speed of `SubBytes`, two techniques, basis conversion and composite field, are applied to implementations in this work. Moreover, `MixColumns` is also speeded up by pre-calculating the values of every power of `xtime` of constants in `MixColumns`. Two structures of 32-bit data bus are implemented. The normal structure provides throughput of 1.7902 Gbps and costs 83.094k gate counts on 0.18-μm CMOS cell standard library; the pipeline structure has 4.9516 Gbps and costs 125.993k gate counts. This work provides a customized Rijndael cipher to let users change coefficients; therefore, it can be utilized in the applications requiring customized security, e.g., the virtual private networks.

This chapter is organized as follows. In Section 4.1, the problems about DFAs on AES are introduced. The other weaknesses caused by algebraic properties of AES are shown in Section 4.2. Section 4.3, the AES algorithm is briefly described and the notations used throughout are defined. In Section 4.4, our proposed error detection schemes for AES are described. Derivation of error detection for each operation, including `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`, is explained, as well as the design of the key schedule. The undetectable errors of each proposed method are theoretically analyzed in Section 4.5, while in, Section 4.6, the realization issues of three levels, operation level, round level, and algorithm level, are described. In Section 4.7 advantages and comparisons between this work and other research studies are discussed and in Section 4.8 the detection capability of each scheme is simulated. In Section 4.9, the details of designing `SubBytes` and `MixColumns` are described. Section 4.10 shows the architecture and how it works. The performance and hardware complexity of each structure are discussed in Section 4.11. Finally, our conclusions are offered in Section 4.12.

4.1 The Security Issues While Implementing AES

The *Advanced Encryption Standard* (AES) [47], a successor to the *Data Encryption Standard* (DES), was finalized in October 2000 by the U.S. *National Institute of Standards and Technology* (NIST), when the Rijndael algorithm [12] was adopted. The data block size of AES is 128-bit, and the key size can be 128-bit, 192-bit, or 256-bit. In AES, although the data block is 128-bit, all operations are byte-oriented over $GF(2)$ or $GF(2^8)$. Therefore, several kinds of AES implementations have been discussed. In general, three main types of AES implementations have been discussed, 128-bit, 32-bit, or 8-bit architecture. Each architecture has its own applications. Feldhofer et al. [15] designed an 8-bit AES chip to provide security for *radio frequency identification* (RFID). Satoh et al. [57] introduced a 32-bit implementation of AES. Mangard et al. [41] proposed a scalable architecture for AES, which could process 128-bit data or 32-bit data, depending on the number of Sbox.

The hardware implementation of AES would be countered by some side-channel attacks, such as *Differential Fault Attacks* (DFA), or Differential Power Analysis (DPA). Differential fault attacks was originally proposed by Biham and Shamir [7]. Theses side-channel attacks actually threaten the security of several cryptosystems, because they are practical for a crypto module. The idea of DFA is to apply the differential attacks to a crypto module or a crypto chip. The cryptanalyst injects errors by using microwave or ionizing techniques during the encryption or decryption process. Theses errors cause the encryption results to differ from the correct results; hence, the cryptanalyst will receive the difference of outputs. Therefore, such differential attacks may be carried out in real world. Dusart et al. [14] broke the 128-bit AES under the assumption that you can physically modify a hardware AES device. This attack required 34 pairs of differential inputs and outputs, to obtain the final round key. Piret and Quisquater [53] broke AES with two erroneous ciphertext under the assumption that the errors occur between the antepenultimate and the penultimate `MixColumns`.

To avoid the possibility of suffering such attacks, error detection can be considered while implementing a cipher. In 2002, Karri et al. [31] proposed a general error detection method, called *concurrent error detection* (CED), for several symmetric block ciphers including RC6, MARS, Serpent, Twofish, and Rijndael. CED requires an inverse operation to check whether errors have occurred in calculations, or not, and has three levels: the operation level; the round level; and the algorithm level. Taking an operation-level CED in AES as an example, the `InvSubBytes` is required to detect the errors occurring in `SubBytes`, and vice versa. This method has very high fault coverage, but it is time-consuming and high hardware cost, because inverse operations are required. In 2003, Karri et al. [32] proposed a parity-based detection technique for general substitution-permutation block ciphers. However, the size of the table, required by substitution box, is enlarged. In addition, the paper did not address the error detection techniques for some specific functions, such as `MixColumns` in AES. In 2004, Wu el al. [63] applied the structure of [32] to AES and used one-bit parity for a 128-bit data block. The method of Wu et al. [63] can let the parity pass through the `MixColumns`. Bertoni et al. [3] used an error detection code of 16-bit parity for a 128-bit data block. To be precise, this approach uses one-bit parity for each byte, and thus, can detect all single errors and perhaps all odd errors. In [4], Bertoni et al. used the error detection scheme in [3] not only to detect errors bit also to locate errors. In 2004, Bertoni et al. [5] implemented the model proposed in [4]. The introduction of the mode into AES brought the performance 18% overhead of area and 26% decreasing of throughput. According the results given in [3], their approach was able to detect most cases of multiple faults. However, this approach is asymmetrical, between `MixColumns` and `InvMixColumns`, because the parity prediction of `InvMixColumns` is more complex than that of `MixColumns`. Therefore, two circuits are required to predict the parity while merging the encryption and the decryption. Besides, the detection technique for `SubBytes` doubled the table size of `SubBytes` in AES, from 256 to 512 bytes. In addition, it cannot easily be applied to an AES implementation of 8-bit architecture, because the parity prediction of `MixColumns` (`InvMixColumns`) requires information from other bytes and other parities.

This work proposes several error-detection schemes for AES. They are based on the $(n+1, n)$ *cyclic redundancy check* (CRC) over $GF(2^8)$, where $n \in \{4, 8, 16\}$ is the number of bytes contained in the message. The proposed schemes easily predict the parity of an operation's output. Because AES is byte-oriented and its constants are ingeniously designed, the parity of the output can be predicted from a linear combination of the parity of the input. In most cases, the parity is the summation of the input data; also, the proposed schemes are highly scalable and are suitable for 8-bit, 32-bit, or 128-bit architecture. This is important, because many AES designs are in an AES hardware designed either 8-bit or 32-bit architecture. Another advantage of proposed approaches is that the parity calculation between the encryption and the decryption is symmetric, because the parity generation in encryption is quite similar to the one in decryption. This will bring some benefits, while integrating encryption and decryption into one circuit.

4.2 The Algebraic Properties of AES

There are many researches on the algebraic properties of Rijndael. Fuller and Millan [17] showed that the outputs of 8×8 Rijndael SBox are all equivalent under affine transformation. Ferguson et al. [16] represented Rijndael as an algebraic formula. Murphy and Robshaw [45] defined a new block cipher, *Big Encryption System* (BES), which only operates in $GF(2^8)$ and it is a proper superset of AES. BES is mathematically simpler description than AES, and that is helpful to cryptanalysis work of AES. Barkan and Biham [2] showed that replacements of constants in AES can create several dual ciphers. Courtois and Pieprzyk [11] showed that Rijndael can be written as an overdefined system of *multivariate quadratic equations* (MQ) [8] and presented an attack, called XSL attack, based on MQ. The above researches discovered different algebraic properties from different views of Rijndael, and those properties can be utilized for cryptanalysts.

Although Rijndael has above probable leaks, but we still can build a secure connection with it. In Barkan and Bihamn's [2] research, they pointed out that random selecting a dual cipher is desired during a connection. Therefore, if all data in a connection are encrypted by several dual ciphers is possible, a more secure connection can be established by Rijndael. The coefficients of irreducible polynomial $m(x)$, `MixColumns` row vector $c(x)$, and affine transformation could be replaced by other values such that they follow some requirements, such as minimization of the largest non-trivial value for `SubBytes`, relevant diffusion power for `MixColumns` [12], et al..

A parameterizable structure is needed for on-line replacing coefficients during a connection. However, the design, providing the function of on-line changing coefficients, will incur enormous burden on complexity, area and performance, so several techniques are used to increase the performance. Different irreducible polynomials $m(x)$ result in different results of `SubBytes`, the most complex computation in Rijndael, so two techniques are adopted to increase the throughput and to reduce the hardware complexity. First, the basis conversion is used to fix the basis, i.e., the irreducible polynomial, of $GF(2^8)$ inversion. Once the basis is fixed, the implementation of $GF(2^8)$ inversion can be optimized for speed and area. In addition, the $GF(2^8)$ inversion is implemented in a composite field $GF(((2^2)^2)^2)$ for saving the gate counts. The performance of another complex computation, `MixColumns`, will also be dropped, when $m(x)$ and $c(x)$ are changeable. Therefore, based on a given irreducible polynomial $m(x)$, we calculate each power value of `xtime` of constants in $c(x)$ to be used in following encryption/decryption to speed up the implementation of `MixColumns`.

In here, we implement a parameterizable Rijndael in two ways, non-pipeline (normal) and pipeline structure. Because the coefficients are changeable; hence, the chip will operate in different dual ciphers with different given coefficients. The normal structure executes one round per clock cycle on a 128-bit data block, and the pipeline structure requires six clock cycles to perform one round on a 128-bit data block. The data bus of both structures is 32-bit, and only the 128-bit key scheduler is implemented in this work. The normal structure achieves a throughput of 1.7902 Gbps and a 153.84 MHz clock, and has 83.094k gate counts. The pipeline structure has a throughput of 4.9516 Gbps with 425.53 MHz clock and 125.993k gate counts. The details are explained in Section 4.9, 4.10, and 4.11.

This implementation of Rijndael is not only compatible to AES but also available to replace the coefficients in Rijndael, so it can be applied to applications that require customized security. Besides, the throughput of our implementations is over 1 Gbps; hence, the results of this work are suitable to network devices over Fast Ethernet or Giga Ethernet. In particular, the *virtual private network* (VPN) is an appropriate application, because this work can provide customized security for VPN users.

4.3 AES Algorithm

The AES [47] consists of two parts, the data procedure and the key schedule. The data procedure is the main body of the encryption (decryption), and consists of four operations, `(Inv)SubBytes`, `(Inv)ShiftRows`, `(Inv)MixColumns`, and `(Inv)AddRoundKey`. During encryption, these four operations are executed in a specific order– `AddRoundKey`, a number of rounds, and then the final round. The number of rounds is 10, 12, or 14, respectively, for a key size of 128 bits, 192 bits or 256 bits. Each round comprises the four operations, and the final round has `SubBytes`, `ShiftRows` and `AddRoundKey`. The decryption flow is simply the reverse of the encryption, and each operation is the inverse of the corresponding one in encryption. In the data procedure, the 16-byte (128-bit) data block is rearranged as a 4×4 matrix, called state S,

$$S = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}, \tag{4.1}$$

where s_i notes the i^{th} byte of the data block. In this context, S denotes the input of an operation, and T denotes as the output. AES is operated in two fields, $GF(2)$ and $GF(2^8)$. In $GF(2)$, addition is denoted by $\oplus$, and multiplication is denoted by $\otimes$. Similarly, the two symbols, $+$ and $\times$, denote addition and multiplication in $GF(2^8)$.

4.3.1 SubBytes

Two calculations, the $GF(2^8)$ inversion and the affine transformation, are involved in this operation. `SubBytes` substitutes each byte s_i of the data block by

$$t_i = As_i^{-1} + \texttt{63}, \tag{4.2}$$

where s_i^{-1} is the inverse of the input byte, $s_i \in GF(2^8)$, A is an 8×8 circulant matrix of a constant row vector [1 0 0 0 1 1 1 1] over $GF(2)$, and `63` (the `Courier` font number representing a hexadecimal value) belongs to $GF(2^8)$. As_i^{-1} is a matrix-vector multiplication over $GF(2)$.

4.3.2 ShiftRows

The `ShiftRows` operation only changes the byte position in the state. It rotates each row with different offesets to obtain a new state as following:

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \xrightarrow{\texttt{ShiftRows}} \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{bmatrix}. \tag{4.3}$$

The first row is unchanged, the second row is left circular shifted by one, the third row is by two, and the last row is by three.

4.3.3 `MixColumns`

The `MixColumns` operation mixes every consecutive four bytes of the state to obtain four new bytes as following.

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \xrightarrow[\texttt{MixColumns}]{} \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix} \qquad (4.4)$$

Let s_i, s_{i+1}, s_{i+2}, and s_{i+3} represent every consecutive four bytes, where $i \in \{0, 4, 8, 12\}$. Then, the four bytes are transformed by

$$\begin{bmatrix} t_i \\ t_{i+1} \\ t_{i+2} \\ t_{i+3} \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \begin{bmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ s_{i+3} \end{bmatrix}. \qquad (4.5)$$

Each entry of the constant matrix in (4.5) belongs to $GF(2^8)$, hence (4.5) is a matrix-vector multiplication over $GF(2^8)$.

4.3.4 `AddRoundKey` and Key Expansion

Each round has a 128-bit round key which is segmented into 16 bytes k_i as (4.1); the `AddRoundKey` operation is simply an addition,

$$t_i = s_i + k_i, \text{ where } 0 \leq i \leq 15. \qquad (4.6)$$

The key expansion expands a unique private key as a key stream of $(4r + 4)$ 32-bit words, where r is 10, 12, or 14. The private key is segmented into `Nk` words according to the key length, where `NK` is 4, 6 or 8 for a 128-bit, 192-bit or 256-bit cipher key, respectively. As Fig. 4.1 shows, then, it generates the i^{th} word (32 bits) by EXORing the $(i - \texttt{Nk})^{th}$ word with either the $(i-1)^{th}$ word or the conditionally transformed $(i-1)^{th}$ word, where $\texttt{NK} \leq i \leq (4r+3)$. The $(i-1)^{th}$ word is conditionally transformed by `RotWord`, `SubBytes` and

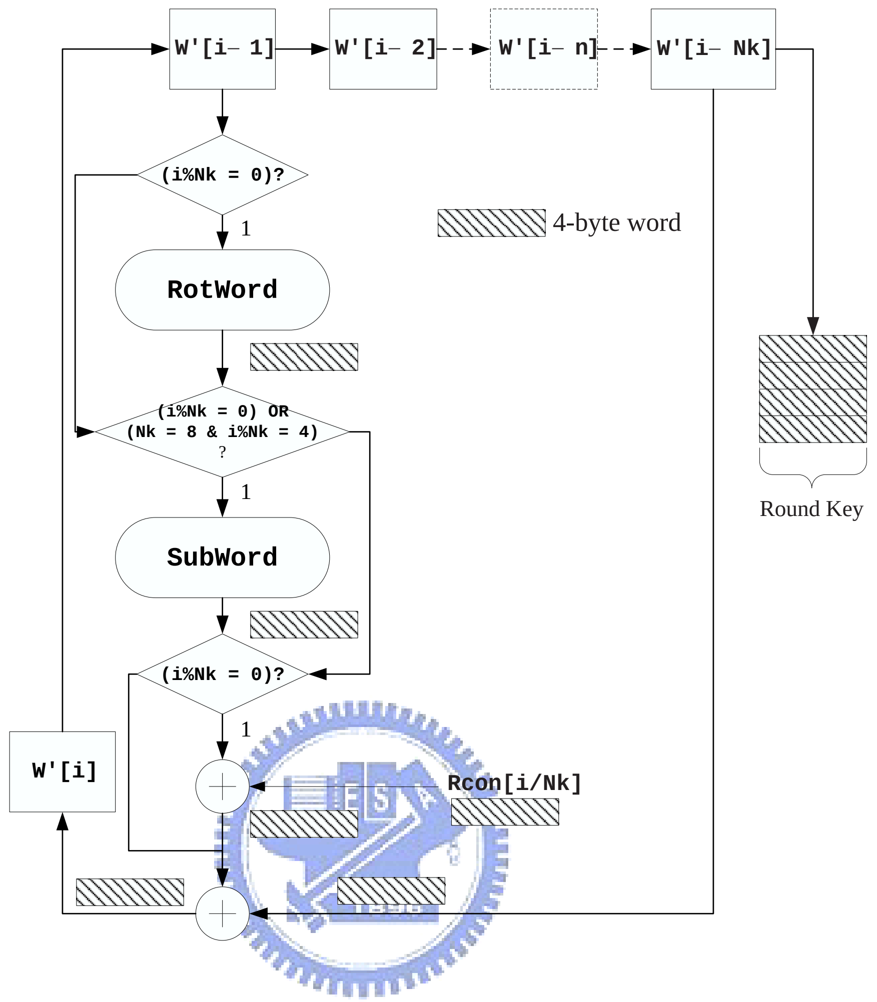


Figure 4.1: The block diagram of key expansion in AES

EXORing with `Rcon[i/Nk]` $= \{\texttt{02}^{\lfloor \texttt{i/Nk} \rfloor}, \texttt{00}, \texttt{00}, \texttt{00}\}$, where the polynomial presentation of $\texttt{02}^{\lfloor \texttt{i/Nk} \rfloor}$ is $x^{\lfloor \texttt{i/Nk} \rfloor}$ over $GF(2^8)$. Finally, the key stream is segmented into several round keys which are involved in the `AddRoundKey` operation.

4.4 Error Detection Techniques

The parts in decryption can be yielded in the similar way; hence, the following context only addressed the error detection in encryption. The differential faults attacks need differential inputs and outputs to attack a cryptosystem; hence, it is assumed that the states and round keys are polluted by additive errors, as shown in Fig. 4.2. In this work, one operation is the smallest granule for designing error detection. In Fig. 4.2, the errors are assumed to be induced between the previous operation and the current operation. If the errors occur

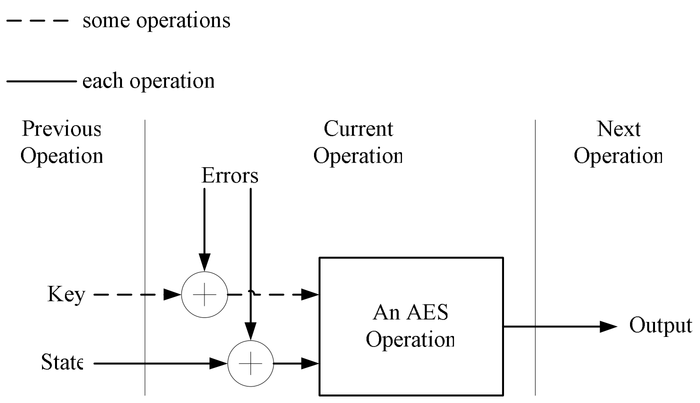


Figure 4.2: The error model assumed in this work. The solid line part appears in every operation and the dotted line part appears in some operations.

in the output of the previous operation, the erroneous input of the current operation will be treated as a different state. Actually, this situation only exists int the first round or in the first operation. The assumed error model is logical, even in the case where the errors occur during the operation. Because each operation of AES is invertible, one unique error block e would exist for an erroneous output T, such that $T = f(S + e)$, where f denotes an operation in AES.

This work adopts a systematic $(n+1, n)$ cyclic redundancy check (CRC) over $GF(2^8)$ to detect errors occurring during encryption, where $n \in \{4, 8, 16\}$ is the number of bytes contained in the message. The generator polynomial is

$$g(x) = \mathtt{1} + x, \tag{4.7}$$

where the coefficients of (4.7) are over $GF(2^8)$. Giving a message $s(x)$ of degree $n-1$, a systematic codeword, generated by $g(x)$, can be obtained from the following two steps:

1. Obtain the remainder $p(x)$ from dividing $xs(x)$ by the generator polynomial $g(x)$. The remainder $p(x)$ is a scalar p here, because the degree of $g(x)$ is one.

2. Combine $p(x)$ and $xs(x)$ to obtain the codeword polynomial,

$$p(x) + xs(x) = p + s_0x + s_1x^2 + \cdots + s_{n-1}x^n, \text{ where } p, s_i \in GF(2^8). \tag{4.8}$$

In step 1, while $g(x)$ is $1 + x$, the remainder $p(x)$ is the summation of all coefficients of the message,

$$p(x) = \sum_{i=0}^{n} s_i. \tag{4.9}$$

Therefore, the parity of a message may be obtained by calculating the summation of the input message over $GF(2^8)$.

Assuming that the received polynomial $t(x)$ is

$$t(x) = t_0 + t_1 x + t_2 x^2 + \cdots + t_n x^n, t_i \in GF(2^8). \tag{4.10}$$

The detection scheme checks whether the syndrome equals to zero or not, where syndrome u is

$$u = \sum_{i=0}^{n} t_i. \tag{4.11}$$

If the syndrome equals to zero, then it is assumed that no errors have occurred; otherwise, errors did occur.

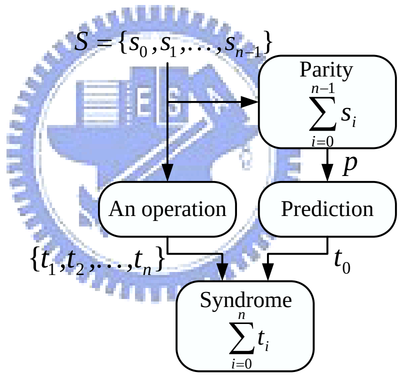


Figure 4.3: The block diagram of the error detection in this work.

In the channel coding field, it is assumed that the message $s(x)$ is transmitted over a noise channel. The channel does not change the message if no errors occur. Therefore, it is easy to predict that t_0 is identical to p, with t_0 being used to detect the errors. However, as shows in Fig. 4.3, the message, $S = \{s_0, s_1, \ldots, s_{n-1}\}$, is transformed into another message, $\{t_1, t_2, \ldots, t_n\}$, by an AES operation; hence, t_0 cannot be obtained instinctively. Therefore, this work investigates the function for each operation, such that t_0 can be predicted by p as shown in Fig. 4.3, making error detection, in each operation, is possible.

This work applies $(n + 1, n)$ CRC to AES, where $n \in \{4, 8, 16\}$. In the case where, $n = 16$, the 128-bit AES state is treated as a message; hence, only one parity is generated for a 128-bit data block. When $n = 4$, the error detection is designed to check each column of the output state. Therefore, four output messages, $\{t_{4j+1}, t_{4j+2}, t_{4j+3}, t_{4j+4}\}$, $0 \leq j \leq 3$, are checked separately. Therefore, four parities are required for a 128-bit data block when $n = 4$. For $n = 8$, two parities are required for a 128-bit data block. The following context addresses the two cases, $n = 16$ and $n = 4$, because $(9, 8)$ CRC for the AES algorithm can be constructed under the similar conditions to $(17, 16)$ or $(5, 4)$ CRC for AES.

4.4.1 In `SubBytes`

In this work, two implementation types of `SubBytes` are considered. The first type uses one table instead of the $GF(2^8)$ inversion and the affine transformation. The second type calculates separately the $GF(2^8)$ inversion and the affine transformation, and the implementation of the $GF(2^8)$ inversion is not limited to the look-up-table method or the combinational logical circuit. In this work, the first type is named as *united SubBytes*, and the second type is as *separated SubBytes*.

For *united SubBytes*, it is assumed that both the `SubBytes` table and the `InvSubBytes` table are stored in a chip, as shown in Fig. 4.4. Error detection is achieved by feeding the output of `SubBytes` into `InvSubBytes`, then comparing the input of `SubBytes` and the output of `InvSubBytes`, and vice versa, as Fig. 4.4 shows. If the both are identical, then it is concluded that no errors have occurred. Otherwise, the errors did occur. This error detection method may be time-consuming, if only the `SubBytes` operation is considered. However, in practical terms, normal encryption could be further processed, without waiting for the error detection result, because `SubBytes` is either the first operation or the second operation in each round. In other words, the operation after `SubBytes`, such as `ShiftRows`, `MixColumns` or `AddRoundKey`, may continue, when the output of the round would be intercepted if errors are detected in `SubBytes`.

If *separated SubBytes* is adopted, error detection must be applied separately to the $GF(2^8)$ inversion and the affine transformation. Considering the error detection for the $GF(2^8)$ inversion first, there are two schemes are proposed herein. Similar to Fig. 4.4, the first scheme detects errors by using the relationship of the mutual inverse. However, the computation of the $GF(2^8)$ inversion is identical for both `SubBytes` and `InvSubBytes`; hence, this scheme does not require the encryption and decryption circuits to simultaneously

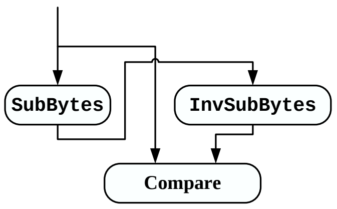


Figure 4.4: The error detection for *united SubBytes*

exist in one chip. It can be used with the encryption-only or decryption-only hardware.

The second scheme is the $(n+1,n)$ CRC, and assumed that the $GF(2^8)$ inversion is implemented in look-up-table approach. Instead of the inverse value of a giving input, the exclusive value of the giving input and its inverse is stored in the table. Therefore, giving an input $\alpha \in GF(2^8)$, the value, $\beta = \alpha + \alpha^{-1}$, is obtained from the table, and then the input α is added to β to yield α^{-1}, as the marked block in Fig. 4.4. The error is detected by the syndrome obtained by the dashed line in Fig. 4.4. In this diagram, no errors are introduced, hence the syndrome is zero.

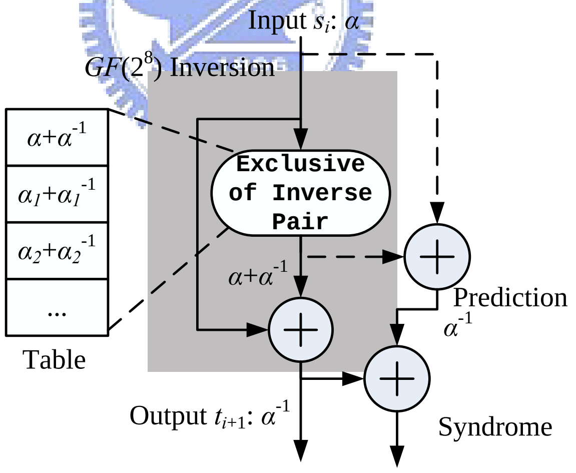


Figure 4.5: The block diagram of one $GF(2^8)$ inversion with the error detection.

For one $GF(2^8)$ inversion, according to Fig. 4.3 and the error model given in Fig. 4.2, the errors are induced a fault at the input of the $GF(2^8)$ inversion, as shown in Fig. 4.6. Supposing that the byte s_i is changed into another byte s_i' by adding the error e_0. Then the syndrome, used to detect errors, is calculated as

$$(s_i + e_1) + t_{i+1} + (t_{i+1} + t_{i+1}^{-1}) = e_0 + e_1. \tag{4.12}$$

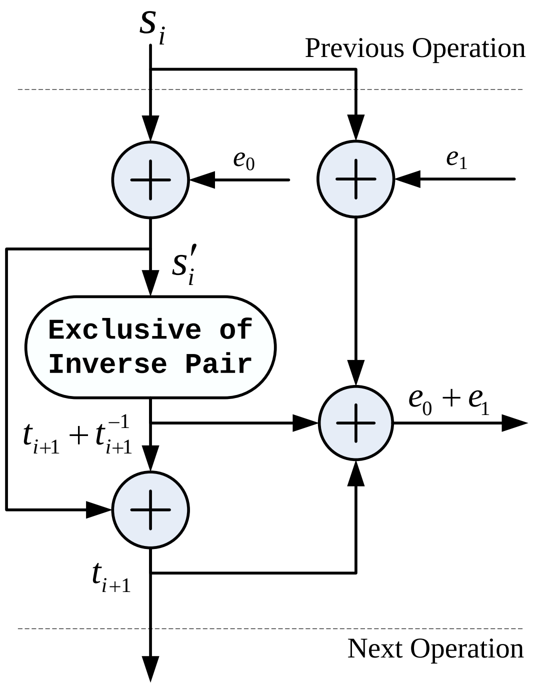


Figure 4.6: An error is injected into the input state after entering the $GF(2^8)$ inversion.

The one-byte structure in Fig. 4.5 could be extended to the 4-byte, 8-byte, or 16-byte structure. Taking the 16-byte structure in consideration, the input state is denoted as $S = \{s_0, s_1, \ldots, s_{15}\}$, and then the parity p is $\sum_{i=0}^{15} s_i$ from (4.9). According to (4.12) and Fig. 4.3, the parity of the output parity t_0 could be predicted by

$$\sum_{i=0}^{15} s_i + \sum_{i=0}^{15}(t_{i+1} + t_{i+1}^{-1}), \tag{4.13}$$

and the syndrome is

$$\begin{aligned} & t_0 + \sum_{i=0}^{15} t_{i+1}, \\ \Rightarrow & \sum_{i=0}^{15} t_{i+1} + p + \sum_{i=0}^{15}(t_{i+1} + t_{i+1}^{-1}). \end{aligned} \tag{4.14}$$

If no errors have occurred, the value t_{i+1}^{-1} will equal to s_i. Therefore, the syndrome (4.14) is zero.

In this work, all `ShiftRows`, `MixColumns`, and `AddRoundKey` are protected by error detection code. However, the detection technique of `SubBytes` is varied with its implementation. According to the error detection scheme for `SubBytes`, three proposed architectures for AES are denoted by, *united-SubBytes detection* (USBD), *hybrid-SubBytes detection* (HSBD), and *parity-based-SubBytes detection* (PbSBD), as shown in Fig. 4.7.

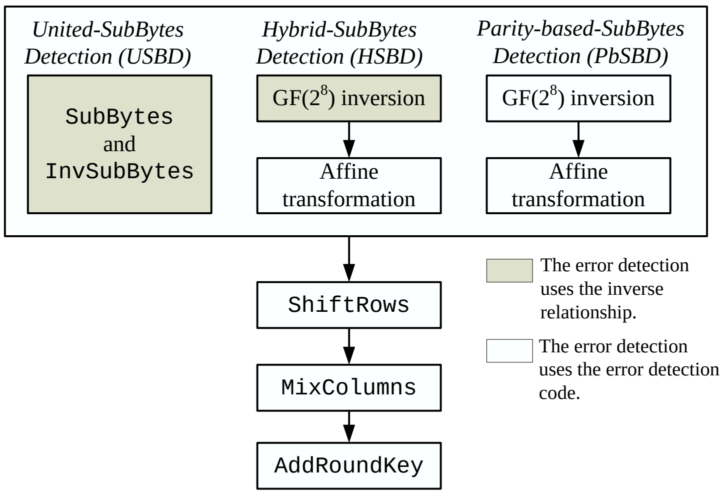

United-SubBytes Detection (USBD)
Hybrid-SubBytes Detection (HSBD)
Parity-based-SubBytes Detection (PbSBD)
SubBytes and InvSubBytes
GF(2^8) inversion
Affine transformation
GF(2^8) inversion
Affine transformation
ShiftRows
MixColumns
AddRoundKey
The error detection uses the inverse relationship.
The error detection uses the error detection code.

Figure 4.7: The three proposed architecture for AES.

For the affine transformation, error detection is achieved by the $(n+1,n)$ CRC, where $n \in \{4,8,16\}$. Considering $n = 16$ first, and according to (4.9), the parity p of an input state, $S = \{s_0, s_1, \ldots, s_{15}\}$, where $s_i \in GF(2^8)$, is generated by

$$p = \sum_{i=0}^{15} s_i. \tag{4.15}$$

The output state is denoted as $T = \{t_0, t_1, \ldots, t_{16}\}$. From (4.2) and Fig. 4.3, t_{i+1} is $As_i + \texttt{63}$, where $0 \leq i \leq 15$. The hexadecimal constant `63` will be eliminated after taking summation of the output state $T \backslash t_0$, i.e.,

$$\sum_{i=0}^{n-1} t_{i+1} = \sum_{i=0}^{n-1}(As_i + \texttt{63}) = A\sum_{i=0}^{15} s_i = Ap. \tag{4.16}$$

Therefore, t_0 can be predicted by (4.16) with input parity p. If no errors occur, the syndrome u must be a zero vector

$$u = \sum_{i=0}^{16} t_i = 0. \tag{4.17}$$

In the case of (5,4) CRC or (9,8) CRC, (4.16) also holds.

4.4.2 In `ShiftRows`

From (4.3), the `ShiftRows` operation simply rotates the input state S, but does not alter the value of s_i. Therefore, t_0 may be directly predicted by $\sum_{i=0}^{n} s_i$ in the case of $n = 16$. Similarly, the `ShiftRows` operation is error free, if the syndrome is zero

$$\sum_{i=0}^{16} t_i = 0. \tag{4.18}$$

When $n = 4$, because each column of the output state would be detected, the four parities p_j, where $0 \leq j \leq 3$, are

$$\begin{aligned} p_0 &= s_0 + s_5 + s_{10} + s_{15}, \\ p_1 &= s_4 + s_9 + s_{14} + s_3, \\ p_2 &= s_8 + s_{13} + s_2 + s_7, \\ p_3 &= s_{12} + s_1 + s_6 + s_{11}; \end{aligned}$$

hence, the $t_{j,0}$ for each output message $\{t_{4j+1}, t_{4j+2}, t_{4j+3}, t_{4j+4}\}$ is p_j. The case of $n = 8$ is analogous to the case of $n = 4$.

4.4.3 In `MixColumns`

The behavior of the `MixColumns` operation is more complex, because each byte in the input state S influences four bytes in the output state T. However, because of the ingenious design of the coefficients, it is also possible to apply the $(n+1, n)$ CRC directly, where $n \in \{4, 8, 16\}$. The `MixColumns` operation works as following

$$\underbrace{\begin{bmatrix} t_{4j+1} \\ t_{4j+2} \\ t_{4j+3} \\ t_{4j+4} \end{bmatrix}}_{T'} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \underbrace{\begin{bmatrix} s_{4j} \\ s_{4j+1} \\ s_{4j+2} \\ s_{4j+3} \end{bmatrix}}_{S'}, \text{ where } 0 \leq j \leq 3. \tag{4.19}$$

From (4.19), it is yielded that the summation of vector T' equals to that of vector S'.

$$\begin{aligned}\sum_{k=0}^{3} t_{4j+k+1} &= (\mathtt{02}+\mathtt{01}+\mathtt{01}+\mathtt{03})s_{4j}+\\ &\quad (\mathtt{03}+\mathtt{02}+\mathtt{01}+\mathtt{01})s_{4j+1}+\\ &\quad (\mathtt{01}+\mathtt{03}+\mathtt{02}+\mathtt{01})s_{4j+2}+\\ &\quad (\mathtt{01}+\mathtt{01}+\mathtt{03}+\mathtt{02})s_{4j+3},\\ &= s_{4j}+s_{4j+1}+s_{4j+2}+s_{4j+3},\\ &= \sum_{k=0}^{3} s_{4j+k}. \end{aligned} \tag{4.20}$$

Therefore, when the (5,4) CRC is applied, the output parity $t_{j,0}$ of the j^{th} column vector may be directly predicted from the j^{th} column vector of the input state by $\sum_{k=0}^{3} s_{4j+k}$. Similarly, in the case $n = 16$, t_0 is predicted by

$$\begin{aligned} t_0 &= \sum_{j=0}^{3}\sum_{k=0}^{3} t_{4j+k+1},\\ &= \sum_{j=0}^{3}\sum_{k=0}^{3} s_{4j+k},\\ &= \sum_{i=0}^{15} s_i. \end{aligned}$$

Because the summation of `02`, `01`, `01` and `03` is `01`, (4.20) can be satisfied for the (17,16), (9,8), or (5,4) CRC. The coefficients of `InvMixColumns` display an identical phenomenon. The summation of the four coefficients used in decryption, `0B`, `0D`, `09`, `0E`, is also `01`. Therefore, t_0 or $t_{j,0}$ can be predicted in the same way as that of `MixColumns`.

4.4.4 In AddRoundKey

Discussing the case $n = 16$ first, it is assumed that each round key already has a parity; hence, the round key is represented as $\{k_0, k_1, \ldots, k_{16}\}$, where $k_0 = \sum_{i=0}^{15} k_{i+1}$ is the parity and $\{k_1, \ldots, k_{16}\}$ is the normal round key. The `AddRoundKey` operation only adds the input state with a normal key $K = \{k_1, k_2, \ldots, k_{16}\}$ to yield the output state as following:

$$T = S + K. \tag{4.21}$$

Applying the summation operation to (4.21) to obtain

$$\sum_{i=0}^{15} t_{i+1} = \sum_{i=0}^{15} s_i + \sum_{i=0}^{15} k_{i+1} = p + k_0. \tag{4.22}$$

Accordingly, t_0 may be obtained from $p + k_0$. The parities for $n = 4$ or $n = 8$, p_j, are calculated in the same way; however, the round key must also have four or two parities.

4.4.5 In the Key Expansion

The $(n+1, n)$ CRC is also adopted in key expansion, where $n \in \{4, 8, 16\}$. However, the (5,4) CRC is always used in the interior of the key expansion. The key expansion and the error detection scheme are jointly depicted in Fig. 4.8, where the decision blocks are removed from Fig. 4.1 for a simple description of error detection, as the conditions only determine where the error detection is applied to, not how it is designed.

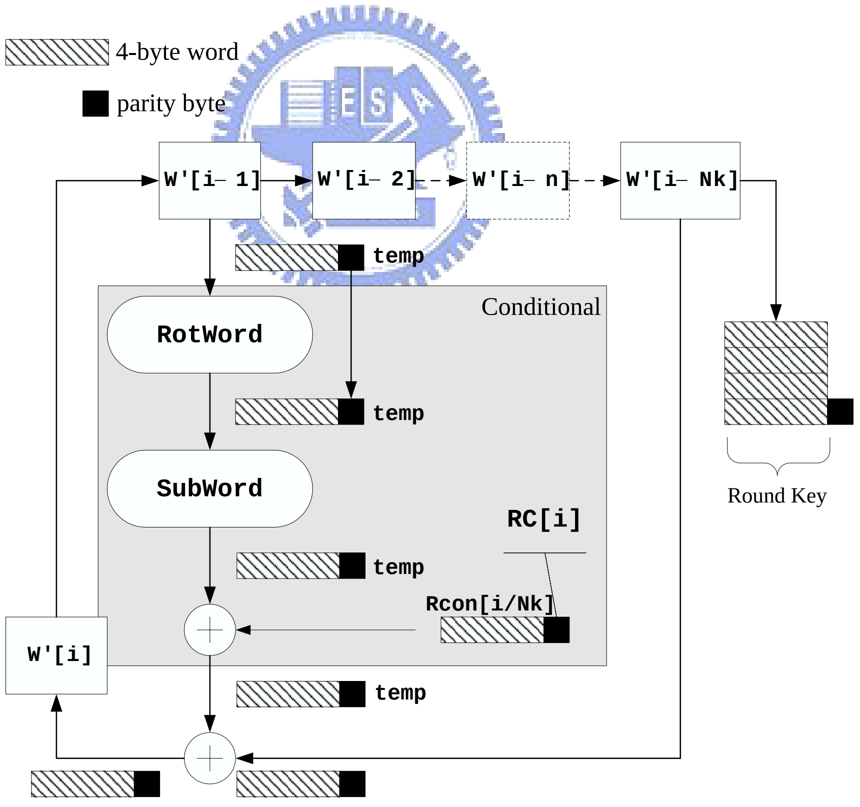


Figure 4.8: The error detection scheme for key expansion.

In this key expansion, with error detection, one word contains five bytes, and the symbol of a word is denoted by `W'[i]`=[`W[i]`‖parity], where ‖ is a catenation symbol. At first, the parities of the first `Nk` words, where `Nk` $\in \{4, 6, 8\}$, are obtained by the generator $1 + x$, i.e., the parity p_i of `W[i]` $= [w_{i,0}\ w_{i,1}\ w_{i,2}\ w_{i,3}]$ is

$$p_i = w_{i,0} + w_{i,1} + w_{i,2} + w_{i,3}. \tag{4.23}$$

Then, the `Nk`-pair parities and messages form new `Nk` words, `W'[0]`, `W'[1]`, ..., and `W'[Nk-1]`. The new words are successively put into the `Nk` shift blocks, from `W'[i-Nk]` to `W'[i-1]`, at the top of Fig. 4.8, after which, the key expansion starts. A 128-bit round key and its one-byte parities are collected after each period of four shifts. If (17,16) CRC is chosen for AES, the one-byte parity of a round key is obtained by summing the four parities of output words. If (5,4) CRC is chosen, then the four parities are kept.

In the key expansion, the `RotWord` rotates the byte order of `W[i-1]`; hence, the parity is the same as that of `W'[i-1]`. For the `SubWord` operation, because it is a function which executes `SubBytes` on each byte of input, the error detection scheme is the same as that in `SubBytes`, described in Section 4.4.1. However, in the case of *united SubBytes* being used, the parity must be calculated separately.

For the EXOR operation with `Rcon[i/Nk]`, the error detection is achieved by EXORing the parity of `temp` and that of `Rcon[i/Nk]`, where `Rcon[i/Nk]` $= \{\texttt{02}^{\lfloor \texttt{i/Nk} \rfloor}, \texttt{00}, \texttt{00}, \texttt{00}\}$. The parity of `Rcon[i/Nk]` equals to $\texttt{02}^{\lfloor \texttt{i/Nk} \rfloor}$ due to the three bytes of zero value in `Rcon[i/Nk]`. At the end of the key expansion, the parity t_0 is the EXOR of the parity of current data and the parity of `W'[Nk-1]`.

4.4.6 More details for (5,4) CRC

Although the (5,4) CRC has four parities, it is possible for only one parity to be used in realization of this scheme. AES can be implemented in a 32-bit structure, i.e., one column of a state is processed once in every round. In this structure, the position of `ShiftRows` must be shifted above the `SubBytes` operation. After `ShiftRows`, each column passes through the identical calculations, `SubBytes`, `MixColumns`, `AddRoundKey`; the parity generation, or the syndrome calculation for each column, are also identical, so only one circuit is required.

4.5 Undetectable Errors

Even though the AES algorithm propagates the errors during encryption, the error coverage can be also analyzed mathematically. Actually, only the `MixColumns` and `SubBytes` operations cause numerous erroneous bits when a single-bit error is injected, when `ShiftRows` or `AddRoundKey` do not change the bit number of the errors. Several assumptions are made, as follows:

1. The error model is considered as Fig. 4.2.
2. All nonzero error block over $GF(2^{8(n+1)})$ have the same probability, where $n \in \{4, 8, 16\}$.
3. Each operation has the same error injection probability.

4.5.1 The undetectable errors in **`SubBytes`**

Because `SubBytes` is invertible, all errors injected into input can be detected by `InvSubBytes`, and vice versa. Therefore, the *united SubBytes*, has 100% fault coverage. In *separated SubBytes*, the both operations, the $GF(2^8)$ inversion and the affine transformation, have their own error detection. The $GF(2^8)$ inversion is also invertible, so it has 100% fault coverage in *hybrid SubBytes*.

In *parity-based SubBytes*, the error detection capability of the $GF(2^8)$ inversion is analyzed. According to (4.14), the scheme only uses XOR operations, so all the codewords are the undetectable errors in *parity-based SubBytes*. Therefore, while applying the $(17, 16)$ CRC to a 128-bit data block, the number of undetectable nonzero errors is $(2^8)^{16} - 1$, and the percentage of undetectable errors is $\frac{(2^8)^{16}-1}{(2^8)^{17}}\% \cong 0.4\%$. When the $(5, 4)$ CRC is applied to a 128-bit data block, the total number of undetectable nonzero errors is $((2^8)^4 - 1)^4$, and the percentage is $(\frac{(2^8)^4-1}{(2^8)^5})^4 \times 100\% \cong 2.56 \times 10^{-8}\%$. Similarly, the percentage of undetectable errors for the $(9, 8)$ CRC is $0.16 \times 10^{-2}\%$.

The affine transformation is detected by $(n + 1, n)$ CRC. Although five erroneous bits were caused, while injecting a single-bit error, the error coverage can still be analyzed.

Theorem 2 *Giving an input state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, where parity p is $\sum_{i=0}^{n-1} s_i$, and $n \in \{4, 8, 16\}$, the output state is $T = \{t_0, t_1, \ldots, t_n\}$ where t_0 is Ap from (4.16), and t_{i+1}, $0 \le i \le n - 1$, is obtained from (4.2). Introducing an error $E = \{e_0, e_1, \ldots, e_n\}$ into the* *state* $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$*, the summation of the output* T' *will equal to zero if and only if* $\sum_{i=0}^{n} e_i = 0$.

Proof: Because n is even, the value `63` will be cancelled. Therefore, the summation of the erroneous output T' is

$$\begin{aligned}\sum_{i=0}^{n} t'_i &= Ap + e_0 + A\sum_{i=0}^{n-1}(s_i + e_{i+1}),\\ &= \underbrace{Ap + A\sum_{i=0}^{n-1} s_i}_{0} + A\sum_{i=0}^{n} e_i,\\ &= A\sum_{i=0}^{n} e_i.\end{aligned}$$

Therefore, $\sum_{i=0}^{n} t'_i$ equals to zero if and only if $A\sum_{i=0}^{n} e_i = 0$ is held. Because the matrix A is nonsingular over $GF(2)$, $A\sum_{i=0}^{n} e_i$ is zero if and only if $\sum_{i=0}^{n} e_i$ is zero. ■

In the $(n+1, n)$ CRC, the nonzero errors are undetected, when the equation $\sum_{i=0}^{n} e_i = 0$ is held, i.e., errors are also the codewords. According to *Theorem* 2, all undetectable errors are also undetected after the affine transformation. Therefore, while applying the $(n+1, n)$ CRC to a 128-bit data block, the percentages of the undetectable errors are 0.4%, 0.16×10^{-2}%, and 2.56×10^{-8}%, respectively for $n = 16$, $n = 8$, and $n = 4$.

4.5.2 The undetectable errors in `MixColumns`

`MixColumns` also has a diffusion property. It causes five or eleven erroneous bits while injecting a single-bit error in one column vector of the input state. However, the coefficients eliminate the diffusion of errors after summing the erroneous column vector of the output state. The `MixColumns` is shown again below, and it is supposed that each byte of the input vector is polluted by an error.

$$\begin{bmatrix} t_{i+1} \\ t_{i+2} \\ t_{i+3} \\ t_{i+4} \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \begin{bmatrix} s_i + e_i \\ s_{i+1} + e_{i+1} \\ s_{i+2} + e_{i+2} \\ s_{i+3} + e_{i+3} \end{bmatrix}. \tag{4.24}$$

Then, the summation of the column vector t_{i+1} is

$$\begin{aligned}\sum_{k=0}^{3} t_{i+k+1} &= (\texttt{02}+\texttt{01}+\texttt{01}+\texttt{03})(s_i+e_i)+\\ &\quad (\texttt{03}+\texttt{02}+\texttt{01}+\texttt{01})(s_{i+1}+e_{i+1})+\\ &\quad (\texttt{01}+\texttt{03}+\texttt{02}+\texttt{01})(s_{i+2}+e_{i+2})+\\ &\quad (\texttt{01}+\texttt{01}+\texttt{03}+\texttt{02})(s_{i+3}+e_{i+3}),\\ &= \sum_{k=0}^{3}(s_{i+k}+e_{i+k}). \end{aligned} \tag{4.25}$$

The equation also holds for two or four columns vectors.

Theorem 3 *Giving an input state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, where $p = \sum_{i=0}^{n-1} s_i$ is the checksum of the input state and $n \in \{4, 8, 16\}$. After* `MixColumns` *and the parity prediction (4.20), the output state is $T = \{t_0, t_1, \ldots, t_n\}$, where $t_0 = p$, and the rest is the output of* `MixColumns`*. Introducing an error $E = \{e_0, e_1, \ldots, e_n\}$ into the state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, then the errors of the $(n+1, n)$ CRC in* `MixColumns` *are undetectable if and only if the summation $\sum_{i=0}^{n} e_i$ is zero.*

Proof: The syndrome $\sum_{i=0}^{n} t_i$ is used to check whether errors occurred or not. It is assumed that no errors occurred, if and only if the syndrome is zero. The summation of the erroneous output state is

$$\sum_{i=0}^{n} t'_i = (t_0+e_0) + \sum_{i=1}^{n} t'_i$$

From (4.25), because n is the multiple of four, the above equation is represented as

$$\begin{aligned}\sum_{i=0}^{n} t'_i &= (t_0+e_0) + \sum_{i=1}^{n}(s_{i-1}+e_i),\\ &= \underbrace{t_0 + \sum_{i=0}^{n-1} s_i}_{0} + \sum_{i=0}^{n} e_i,\\ &= \sum_{i=0}^{n} e_i.\end{aligned}$$

Therefore, the error is undetectable if and only if $\sum_{i=0}^{n} e_i$ is zero. ■

From *Theorem* 3, there are $((2^8)^{16} - 1)$ nonzero errors that are undetectable, when the $(17, 16)$ CRC is applied to a 128-bit data block. This result is the same as those in the affine transformation described above. Similarly, the total number of the undetectable errors for the (9,8) or (5,4) CRC is $((2^8)^4 - 1)^4$ or $((2^8)^8 - 1)^2$, respectively.

4.5.3 The undetectable errors in `ShiftRows` or `AddRoundKey`

`ShiftRows` does not change the value of the input state, and `AddRoundKey` only EXORs the input state with a round key. Therefore, the undetectable errors are the same as those analyzed in the affine transformation or `MixColumns`.

4.6 Detection Levels

The proposed scheme may be used in operation-level, round-level or algorithm-level error detection. In operation level detection, the syndrome is checked at the end of each operation. Similarly, if the syndrome is obtained at the end of each round, it is round-level detection. The implementation of operation-level error detection is easy to figure out. The syndrome is calculated at the end of each operation according to the equations derived in Section 4.4. However, the implementation of a round-level detection needs more ingenuity, when the `SubBytes` is protected by *united SubBytes*. The parity is generated at the end of the `SubBytes` or the beginning of the `ShiftRows`. Then, the parity directly passes through `ShiftRows`, and `MixColumns`, because its value will not be changed after the two operations. Finally, the parity is EXORed with the key parity. The total path is shown in Fig. 4.9 below. Obviously, the syndrome could be then checked at the end of the round. In *hybrid SubBytes*, the structure for round-level error detection is similar to Fig. 4.9, but the parity is generated after the $GF(2^8)$ inversion. Because the parity of the state, in i^{th} round, can not pass through the inversion of $GF(2^8)$ in $i+1$ round, the parity must be re-generated in each round. Therefore, *united-SubBytes detection* or *hybrid-SubBytes detection* cannot be implemented as algorithm-level detection.

However, each operation of *parity-based SubBytes* is protected by $(n+1,n)$ CRC, hence the parity could pass through a round. Therefore, *parity-based SubBytes* could be applied as an operation-level, round-level, or algorithm-level error detection.

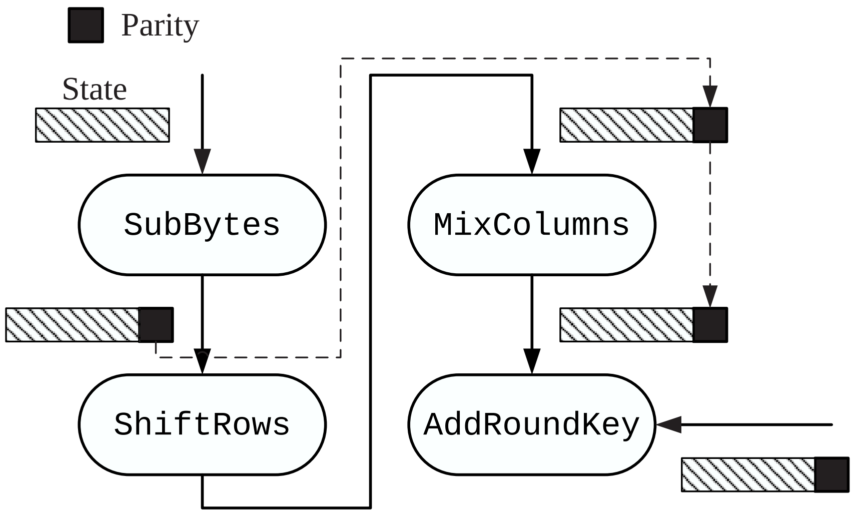


Figure 4.9: The proposed scheme under round-level error detection.

4.7 Features and Costs

4.7.1 Scalability

In Section 4.4, it was found that the three error detections, $(n+1, n)$ CRC, where $n \in \{4, 8, 16\}$, had similar structures. The calculations of parities or syndromes were all based on Byte-EXOR (B-EXOR) operation and the length of the message was a multiple of four bytes. Therefore, the proposed approach is scalable with practical hardware design; in other words, the three CRCs can be applied to an AES implementation of an 8-bit, 32-bit, or 128-bit structure. In general, the portable devices are more probable to encounter DFA than non-portable device. Therefore, the scalability of error scheme is good for practical purposes, because 8-bit and 32-bit architectures are most commonly used in portable applications, such as cell phones, SmartCard, or RFID tag.

The approach proposed by Bertoni et al. [3] cannot easily be scaled down into the 8-bit architecture, because the parity of s_i requires the information from s_{i+1} and s_{i+2}. However, this work can easily be applied to an 8-bit, 32-bit or 128-bit AES architecture. The syndrome generation is similar to parity generation. Fig. 4.10 shows a block diagram of (4.17) and (4.16) for 8-bit AES architecture. While sixteen bytes t_i, are obtained, the syndrome u is obtained immediately, where the initial value of parity registers as a zero byte. The

`ShiftRows`, `MixColumns`, or `AddRoundKey` have similar structures to Fig. 4.10, but the matrix transformation, A, is not required. The 32-bit or 128-bit AES can also be implemented, based on the concept in Fig. 4.10.

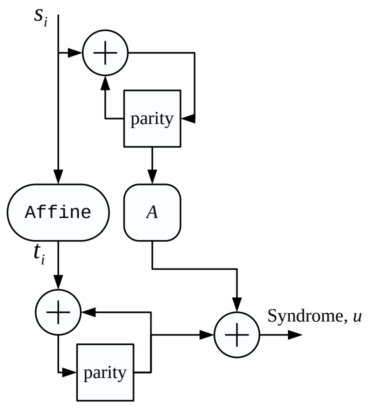


Figure 4.10: The block diagram of error detection for 8-bit AES architecture.

The 32-bit architecture is the most flexible structure from the point of error detection, because it could use (17,16), (9,8) or (5,4) CRC to achieve the error detection objective. No matter which one is selected, it is possible that only a one-byte register is required to store the parities. However, the input must be a one-column vector, defined in AES; thus, (4.20) may be used to detect faults for a one-column calculation.

4.7.2 Symmetry

From Fig. 4.10, it can be seen that the proposed scheme is symmetric in both encryption and decryption. This has the advantage of the encryption and decryption being integrated into one chip. However, the scheme proposed by Bertoni et al. [3] is asymmetrical in `MixColumns` and `InvMixColumns`. As shown in Table 4.1, the output parity prediction of `InvMixColumns` is more complex than that of `MixColumns`.

4.7.3 Costs

While introducing proposed error detection schemes into AES, hardware cost, required by those schemes, is evaluated through their computational complexity. Error detection consists of two parts– the parity and syndrome generation. Discussing the cost in parity generation

Table 4.1: The cost of syndrome generation in each AES operation. (B-EXOR = 8 b-EXORs, b-EXOR=bit EXOR operation, EN=encryption, DE=decryption, AM = affine multiplication)

		Ours (n =16, 8 or 4)	Bertoni [3]	Karri [31]
Bit number of parity		8/16/32 bits	16 bits	0 bit
`SubBytes`	USB	`InvSubBytes`	$m \times 256$ bytes memory, and comparison circuits.	InVSubBytes
	HSB	the $GF(2^8)$ inversion, 16×8 b-EXORs, and 1/2/4 AMs	—	
	PbSB	32×8 b-EXORs, 16×8 b-EXORs, and 1/2/4 AMs	—	
`ShiftRows`		16×8 b-EXORs	16×8 b-EXORs	`InvShiftRows`
`MixColumns`	Cost in EN	16×8 b-EXORs	$16 \times 8 + 16 \times 4$ b-EXORs	`InvMixColumns`
	Cost in DE	16×8 b-EXORs	More complicated than in EN	`MixColumns`
`AddRoundKey`		$(16 + 1/2/4) \times 8$ b-EXORs	$16 \times 8 + 16$ b-EXORs	`AddRoundKey`

first, in our proposed schemes, the parity requires only the EXOR operation. A total of $(n-1)\times\frac{16}{n}$ Byte-XORs (B-EXOR) is required to calculate the parity of the input for the proposed approach. Taking the $(5,4)$ CRC for a 128-bit data block as an example, one checksum of an input message is generated by three B-EXORs, and a total of 12 B-EXORs for four parities. However, *united SubBytes* uses `InvSubBytes` to check error, so no parity generation is required. In *hybrid SubBytes*, the $(n+1,n)$ CRC is applied to the affine transformation; fifteen, fourteen or twelve B-XORs are required to produce the parities for n being 16, 8 or 4, respectively. In the method proposed by Bertoni et al. [3], 16×7 bit-EXORs (b-EXOR) were required to obtain sixteen one-bit parities for an AES state. In [31], they used the inversion operation to detect the errors; hence, no parities were paid for. However, the hardware of parity generation is minor, because the parity generation is required to perform at the beginning of the parity-based detection is applied. In PbSBD, because the parity can pass through each operation along with predicting the parity, the parity generation only performs once. In USBD and HSBD, the parity must be regenerated in `SubBytes` of each round; nevertheless, only one circuit of parity generation is required when one round is implemented to achieve AES computing. In the approach of Bertoni et al. [3], the parity also can pass through the round; hence, one circuit of parity generation is required.

As regards the cost of the syndrome generation, it varies from operation to operation. *United SubBytes* uses the *InvSubBytes* to detect errors. In *hybrid SubBytes*, the $GF(2^8)$ inversion is used to check errors; the $(n+1,n)$ CRC is used to detect errors. According to (4.17), 16 B-EXORs are required to obtain the syndrome for every $(n+1,n)$ CRCs. However, the execution number of affine multiplication, (4.16), depends on n; the number is one, two or four, when n is 16, 8 or 4, respectively. For *parity-based SubBytes*, the cost in affine transformation is the same as that in *hybrid SubBytes*. However, the $GF(2^8)$ inversion also uses $(n+1,n)$ CRC; according to (4.14), 32 B-EXORs are required (note that the $(t_{i+1}+t_{i+1}^{-1})$ in (4.14) is obtained from table, not requiring EXOR calculation). In `ShiftRows` and `MixColumns`, no prediction functions are necessary, and the syndrome is obtained by summing all output byte and the parity. Therefore, in the two operations, 16 B-EXORs are required. In `AddRoundKey`, the one, two or four one-byte parities of a round key are involved in the calculation, requiring extra B-EXORs to be paid for.

The costs of Bertoni et al. [3]'s approach are varied in each operation. The `SubBytes` requires extra m 256-byte memory spaces, where m is dependent on the implementation of the AES. Taking an AES implemented in a 32-bit structure as an example, four bytes are calculated in parallel, thus four tables are required. The size of a table with error detection, in [3], is a double of that in AES, so a total of 512 bytes is for one table, i.e., 256 extra bytes are caused for one table. The 256 extra bytes are constants with odd parity, e.g., 00000000 1; therefore, one comparison circuit or syndrome generation circuit are required to detect the error. The error detection of one byte, appended with one-bit parity, requires eight b-EXORs (bit EXOR operation), or a total of 16×8 b-EXORs for a 128-bit data block. However, Bertoni et al.'s scheme must predict the output parity in `MixColumns`, therefore the extra calculations of 16×4 b-EXORs are required in the encryption process. In decryption, the error-detection hardware for `InvMixColumns` is more complicated than in encryption. Because the prediction of `InvMixColumn` is not derived in [3], the cost is not specified in Table 4.1. The costs of Karri et al.'s scheme required the inversion of each operation and it was also time-consuming. The operations in the key expansion are similar to the four major operations of AES; thus, the detailed comparisons of the key expansion are not discussed. Although most operations require 16 B-EXORs to compute the syndrome, it is possible to achieve the computation with less B-EXORs.

4.8 Performances

Table 4.2: The possible combinations of our proposed schemes.

USBD	HSBD	PbSBD
(17,16)	(17,16)	(17,16)
(9,8)	(9,8)	(9,8)
(5,4)	(5,4)	(5,4)

All simulations and statements, addressed here, are also under the three assumptions given in Section 4.5. Three architectures, USBD, HSBD and PbSBD, were proposed herein; each architecture has three types of CRC, (17,16), (9,8) and (5,4) CRCs, as shown in Table 4.2. Thus nine methods were simulated. In PbBSD, the data procedure is thoroughly protected by the $(n+1, n)$ CRC; thus, each operation has undetectable errors. However, in USBD, the fault coverage in `SubBytes` is 100%, so the amount of overall undetectable errors is 80% of that in USBD. Similarly, in HSBD, the amount is reduced 75% of that in USBD.

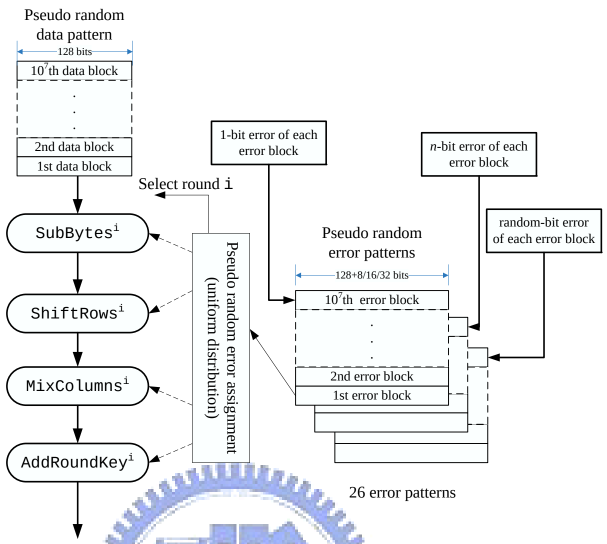

Pseudo random data pattern
128 bits
10^7th data block
2nd data block
1st data block
Select round i
SubBytes^i
ShiftRows^i
MixColumns^i
AddRoundKey^i
Pseudo random error assignment (uniform distribution)
1-bit error of each error block
n-bit error of each error block
random-bit error of each error block
Pseudo random error patterns
128+8/16/32 bits
10^7th error block
2nd error block
1st error block
26 error patterns

Figure 4.11: The simulation model. Each data block has 64 ones, and the position of ones uniformly distributed in a data blcok. The error bits uniformly distribute in an error block. The assignment of error blocks uniform distributes in both rounds and operations.

The simulation model is shown in Fig. 4.11. Each method is simulated by twenty six tests distinguished by the bit number of the injected errors. The last test in Fig 4.12, Fig. 4.13 and Fig. 4.14, labeled as random, used error patterns with random erroneous bit number. Each error pattern has 10^7 blocks, and the bit length of every block is $136(128+8)$, $144(2 \times (64+8))$, or $160(4 \times (32+8))$, respectively for the (17,16), (9,8), or (5,4) CRC. The all-one error block was considered as a totally different state; hence, the maximum number of erroneous bits was 135, 143, or 159 in random test. Each test used one data pattern of 10^7 data blocks, and every block has 64-bit ones of normal distribution. The erroneous rounds and erroneous operation were also randomly chosen.

As seen in Fig. 4.12, all the simulated odd-bit errors were detected. The percentage of the undetectable errors dropped dramatically, as the erroneous bit number increased. When the number of erroneous bits was greater than eight, the percentage was below 1%, and stable. The test using random erroneous bits is about 0.3%, and it was close to the theoretic value obtained in Section 4.5, 0.4%. Obviously, all the experimental results followed the curves of ideal cases.

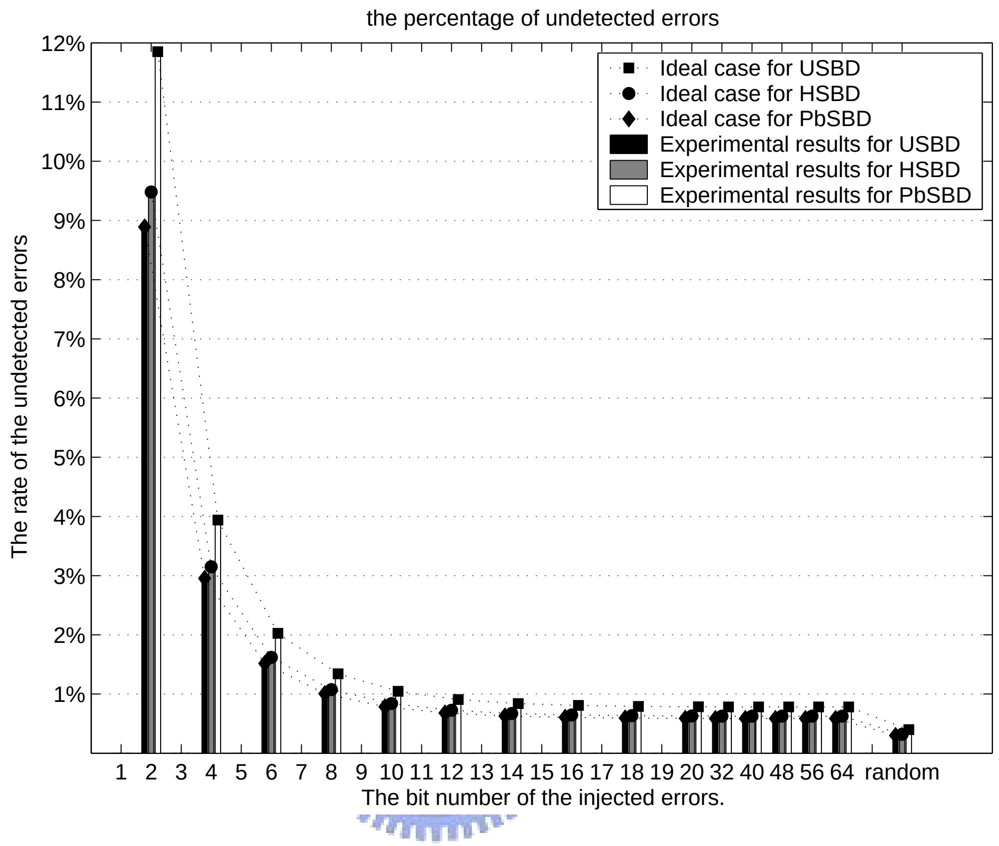


Figure 4.12: Percentage of undetectable errors of the (17,16) CRC over $GF(2^8)$.

The same data patterns used in the above tests were also used for the (9,8) CRC and the (5,4) CRC; all test conditions, except for the error patterns, were identical to those used to test the (17,16) CRC. The (9,8) CRC generated two parities for a 128-bit data block. Because the values in the two tests, 2-bit and 4-bit erroneous bits, are too large, they were dependently shown in Fig. 4.13. All odd-bit errors were also detected. The percentage also dropped dramatically when the erroneous bits increased, as shown in Fig. 4.13. For the random test, the percentage is about $0.14 \times 10^{-2}\%$, very close to the theoretical value of $0.16 \times 10^{-2}\%$.

In Fig. 4.14, the results of the (5,4) CRC and Bertoni et al. [3] are shown. Obviously, this percentage is very small in contrast to the (17,16) CRC or the (9,8) CRC. When the number of erroneous bits was larger than 16, the percentages of undetectable errors dropped to zero. The percentage in the random test was 0%, very close to the theoretic value of $2.56 \times 10^{-8}\%$. Of course, all odd-bit errors could be detected.

Fig. 4.14 also shows the results in Bertoni et al. [3]. The test models of Bertoni et al. [3] are different from ours. They have injected multiple bit errors (between 2 to 16) at the beginning of the round. From Fig. 4.14, their scheme has better error detection than ours, when the errors are between 2 to 6, and the cases of 8-bit errors are close. When the number of erroneous bits is above 10, the performance of the proposed scheme is better than that of Bertoni et al. [3].

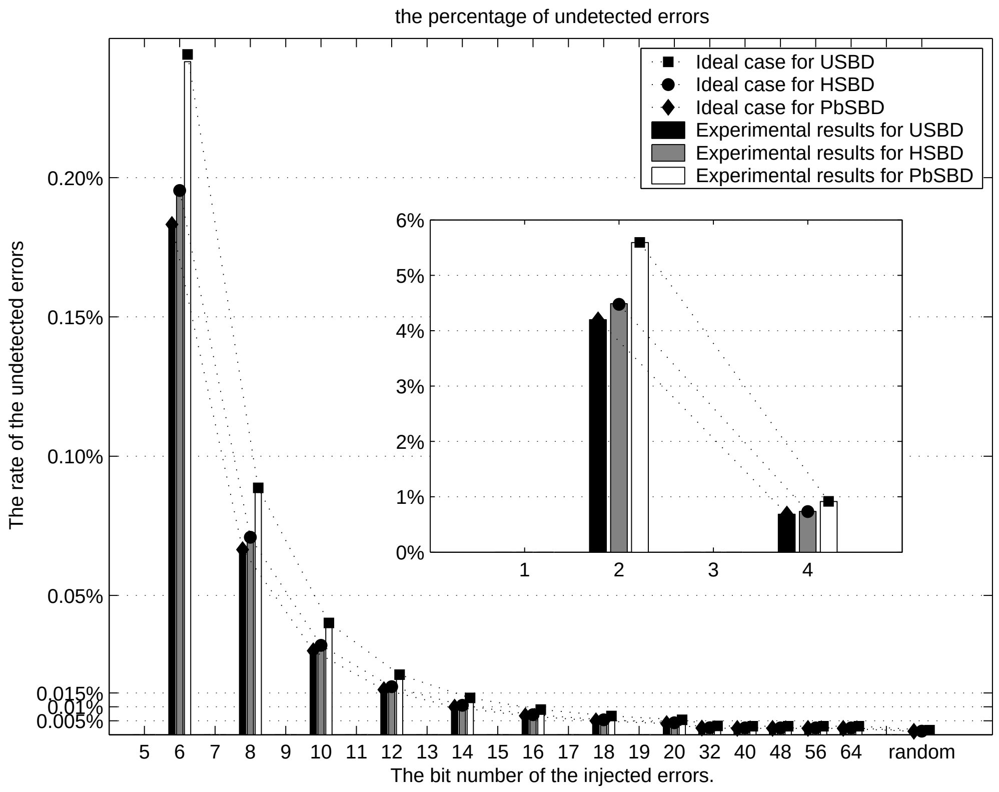


Figure 4.13: Percentage of undetectable errors of the (9,8) CRC over $GF(2^8)$. The percentage is 4.14% for 2-bit errors and 0.67% for 4-bit errors.

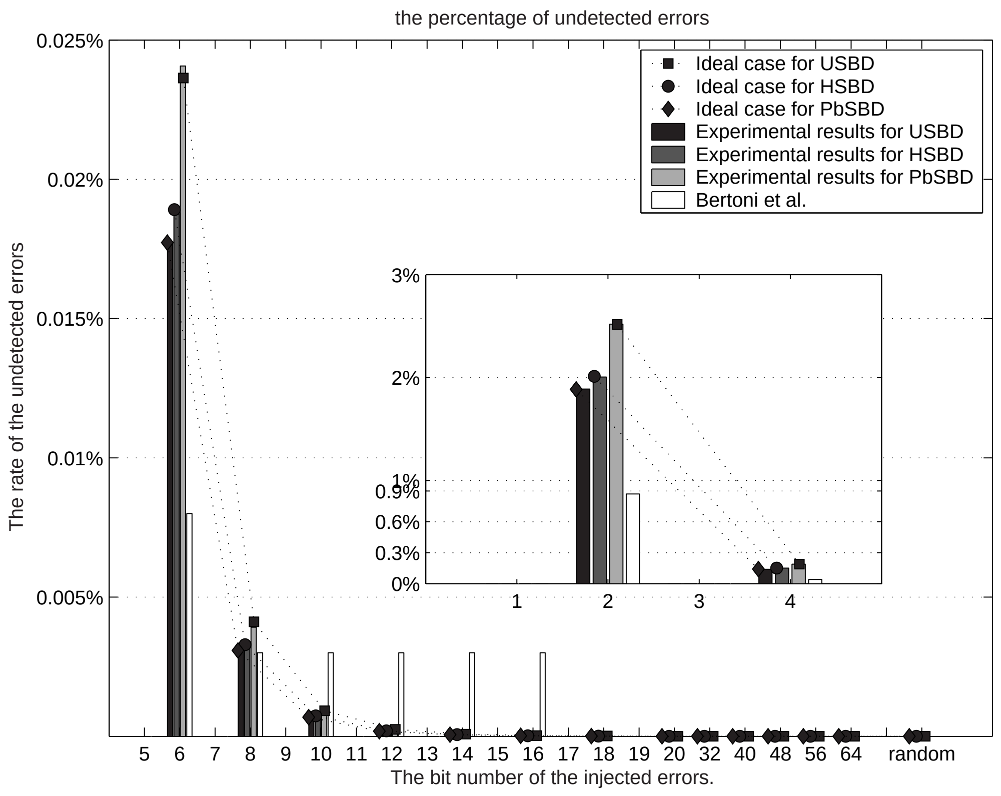


Figure 4.14: Percentage of undetectable errors of the (5,4) CRC over $GF(2^8)$. The percentage is 1.8% for 2-bit errors and 0.13% for 4-bit errors.

4.9 Parameterized Rijndael Algorithm

According to the algebraic properties described in Section 4.2. The model of Rijndael is very mathematical, and the operations are easy to represent in equations over $GF(2^8)$ and $GF(2)$. The constants, chosen in the design of Rijndael, are security-concerned and implementation-oriented. However, it is possible to replace the coefficients in Rijndael algorithm without changing the strength. In this work, the irreducible polynomial, the matrix of `MixColumns`, the matrix of affine transformation is changeable.

4.9.1 Change irreducible polynomial

The irreducible polynomial affects the implementation of multiplication over $GF(2^8)$. In Rijndael, the `SubBytes` and `MixColumns` obtain different values with different irreducible polynomials. The multiplication and inversion over Galois Field are time-consuming; hence, changing irreducible polynomial lets the hardware design of `SubBytes` and `MixColumns` more complex and more inefficient than that of original Rijndael. For enhancing the performance, the technique of basis conversion is adopted to speed up the `SubBytes`, and $\texttt{xtime}^n(c_i)$ is calculation in advance, where $n \in \{0, 1, \ldots, 7\}, i \in \{0, 1, 2, 3\}$, and c_i is the entry of the `MixColumns` matrix.

The `SubBytes` includes two calculations, the $GF(2^8)$ inversion and the affine transformation. In the proposed design, the implementation of the $GF(2^8)$ inversion uses the techniques including basis conversion and the composite-field inverter. The basis conversion [25] changes the representation of an element, over $GF(2^m)$, from a basis to another basis with an $m \times m$ matrix over $GF(2)$. If any input is transformed into the representation in a fixed basis, then the $GF(2^8)$ inversion in parameterized Rijndael can be implemented in a fixed-basis architecture.

Suppose that an element $\alpha_{B_1} \in GF(2^8)$ is given, where α_{B_1} denotes the representation of α by means of basis B_1, and the element α_{B_1} is transformed into the fixed basis B_0 before performing the $GF(2^8)$ inversion. The $GF(2^8)$ inversion calculates in B_0, and then the outcome of inversion is transformed back to the basis B_1. The block diagram is shown in Fig. 4.15.

We choose the irreducible polynomial defined in Rijndael as B_0, so the hardware design

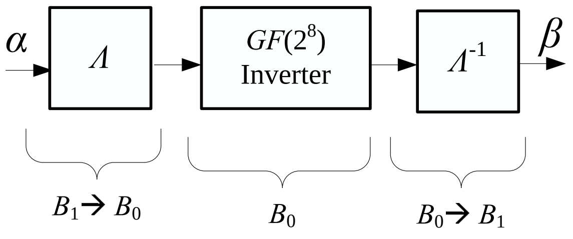


Figure 4.15: Basis conversion in `SubBytes`

of the $GF(2^8)$ inverter could be adopted an existing one. In this work, the implementation of $GF(2^8)$ inverter implemented in the $GF(((2^2)^2)^2)$ field, proposed in [57]. Two isomorphism functions are required between $GF(2^8)$ and $GF(((2^2)^2)^2)$.

$$\text{isomorphism } \Theta : \text{GF}(2^8) \rightarrow \text{GF}(((2^2)^2)^2) \tag{4.26}$$

$$\text{isomorphism } \Theta^{-1} : \text{GF}(((2^2)^2)^2) \rightarrow \text{GF}(2^8) \tag{4.27}$$

The two isomorphism matrices are

$$\Theta = \begin{bmatrix} 1&1&0&0&0&0&1&0 \\ 0&1&0&0&1&0&1&0 \\ 0&1&1&1&1&0&0&1 \\ 0&1&1&0&0&0&1&1 \\ 0&1&1&1&0&1&0&1 \\ 0&0&1&1&0&1&0&1 \\ 0&1&1&1&1&0&1&1 \\ 0&0&0&0&0&1&0&1 \end{bmatrix}, \Theta^{-1} = \begin{bmatrix} 1&0&1&0&1&1&1&0 \\ 0&0&0&0&1&1&0&0 \\ 0&1&1&1&1&0&0&1 \\ 0&1&1&1&1&1&0&0 \\ 0&1&1&0&1&1&1&0 \\ 0&1&0&0&0&1&1&0 \\ 0&0&1&0&0&0&1&0 \\ 0&1&0&0&0&1&1&1 \end{bmatrix}. \tag{4.28}$$

The $GF(((2^2)^2)^2)$ inverter replaces the $GF(2^8)$ inverter in Fig. 4.15, as shown in Fig. 4.16, and the four matrices, $\Lambda, \Lambda^{-1}, \Theta$ and Θ^{-1} ,can be merged into two matrices, $\Gamma = \Lambda \times \Theta$ and $\Gamma = \Theta^{-1} \times \Lambda^{-1}$, to reduce the computing times. There are thirty 8-degree irreducible polynomials, i.e., thirty bases, $\{B_0, B_1, ..., B_{29}\}$. Therefore, the 58 matrices, $\{\Lambda_{B_i \rightarrow B_0} \times \Theta, \Theta^{-1} \times \Lambda^{-1}_{B_i \rightarrow B_0}\}$, can be calculated beforehand by using the algorithm described in [25], where $i \in \{1, 2, \ldots, 29\}$. The 58 matrices only require memory space of 464 (58 × 64 bits) bytes. In this work, $\Gamma_{B_i \rightarrow B_0}$ and $\Gamma^{-1}_{B_0 \rightarrow B_i}$ are given from the outside, not being stored in the chip.

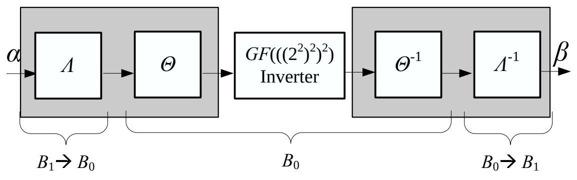


Figure 4.16: The computation of $GF(2^8)$ inversion with arbitrary irreducible polynomial

4.9.2 Change affine matrix

The affine transformation in Rijndael composes of an 8×8 matrix multiplication and an 8×1 vector addition,

$$\beta = A\alpha + B,$$

where α and $\beta \in GF(2^8)$, A is an 8×8 matrix over $GF(2)$, and B is an 8×1 vector over $GF(2)$. In Rijndael, the affine transformation is obtained from the modular polynomial multiplication followed by an addition,

$$\begin{aligned} \beta(x) &= \alpha(x)(x^7 + x^6 + x^5 + x^4 + 1) +, \\ & (x^7 + x^6 + x^2 + x) \bmod x^8 + 1. \end{aligned}$$

This polynomial calculation results in a circulant matrix A in original Rijndael. However, the affine transformation is bijective, if and only if the matrix A is unique; hence, any unique 8×8 matrix over $GF(2)$ could be applied to affine transformation in this work. The matrix multiplication is implemented by using only AND and XOR gates in this work.

4.9.3 Change `MixColumns` matrix

Because the irreducible polynomial is changeable, the implementation of `xtime()` function, described in [12], should be modified. In addition, the `MixColumns` matrix is also changeable in this work. If each multiplication of two elements of $GF(2^8)$ is achieved by repeating `xtime()`, the performance of `MixColumns` will drop dramatically. Therefore, after the irreducible polynomial, $m(x)$, and the `MixColumns` matrix, C, are given,

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \\ c_3 & c_0 & c_1 & c_2 \\ c_2 & c_3 & c_0 & c_1 \\ c_1 & c_2 & c_3 & c_0 \end{bmatrix} \tag{4.29}$$

the value of $\texttt{xtime}^n(c_i)$ is calculated beforehand and stored in 4×8 8-bit registers, where c_i is the entry of `MixColumns` matrix, and $n \in \{0, 1, \ldots, 7\}$. Suppose $s_0 = \texttt{0xCA}$ in (4.5), the calculation of $c_0 \times \texttt{0xCA}$ is achieved by

$$\begin{aligned} c_0 \times \texttt{0xCA} &= 1 \cdot \texttt{xtime}^7(c_0) + 1 \cdot \texttt{xtime}^6(c_0)+, \\ &\quad 0 \cdot \texttt{xtime}^5(c_0) + 0 \cdot \texttt{xtime}^4(c_0)+, \\ &\quad 1 \cdot \texttt{xtime}^3(c_0) + 0 \cdot \texttt{xtime}^2(c_0)+, \\ &\quad 1 \cdot \texttt{xtime}(c_0) + 0 \cdot c_0. \end{aligned}$$

In this approach, there are 32 8-bit registers to save $\texttt{xtime}^n(c_i)$ of `MixColumns` or `InvMixColumns`

4.10 The Hardware Structure

In this work, a 128-bit half-duplex parameterized Rijndael is proposed, and its architecture is depicted in Fig. 4.17. The key schedulers for encryption and decryption are also implemented. The solid line is the encryption path, and the dash lined is the decryption path. In order to make the decryption have the identical sequence of operations as encryption has, the positions of `AddRoundKey` and `InvMixColumns` are exchanged. Therefore, `InvMixColumns` also presents in key scheduler in decryption. Considering a 128-bit state, x, is processed with a normal sequence, `AddRoundKey` and then `InvMixColumns`,

$$x' = C^{-1} \times (x + k),$$

where k is a round key. If the sequence is reversed, i.e., `InvMixColumns` and then `AddRoundKey`, then x' is

$$x' = C^{-1}x + C^{-1}k.$$

Therefore, the `InvMixColumns` has to be incorporated into the key scheduler in decryption to obtain $C^{-1}k$.

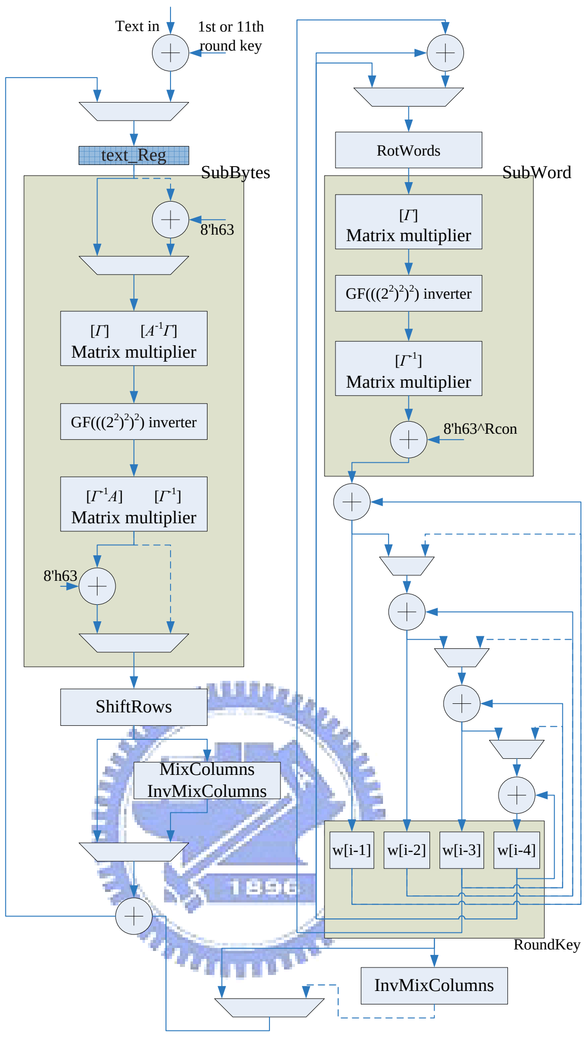


Figure 4.17: Architecture of parameterized Rijndael designed in this work. The dash line is the decryption path.

In this architecture, the generation of round keys in decryption is on-the-fly to save the memory requirement for storing every round key. In decryption, the final round key is generated first, and then is used to produce other round keys. The data procedure is a 128-bit architecture, i.e., 16 bytes are processed simultaneously. In key schedule, the structure is 32-bit; hence, the `SubWords` calculates one 32-bit word per clock cycle. However, the `InvMixColumns` in key schedule processes one 128-bit state per clock cycle.

For enhancing the throughput, a six-stage pipeline designed based on the architecture shown in Fig. 4.17 is proposed. The critical path delay of `SubBytes` is largest; hence, it is divided into three stages, and one stage for each remaining operation.

4.10.1 Input Coefficients

In order to utilize this module easily, the data bandwidth of the input and output interface is 32-bit. In the proposed architecture, besides the key and text, the coefficients of Rijndael also need to be given. The bit number of each coefficient is given in Table 4.3.

Table 4.3: The bit number of each changeable coefficient

Parameters	Bit number (bits)
basis conversion matrix (Γ)	64
inverse basis conversion matrix (Γ^{-1})	64
affine matrix	64
inverse affine matrix	64
affine constant	8
row vector of C	32

As that described in subsection 4.9.1, instead of the irreducible polynomial $m(x)$, the 8×8 matrix of basis conversion and its inverse are as inputs. The affine matrix can be any unique matrix, so 64-bit input is required. Because the matrix over $GF(2^8)$ in `MixColumns` is circulant, only a 32-bit row vector is necessary.

4.10.2 Initialization

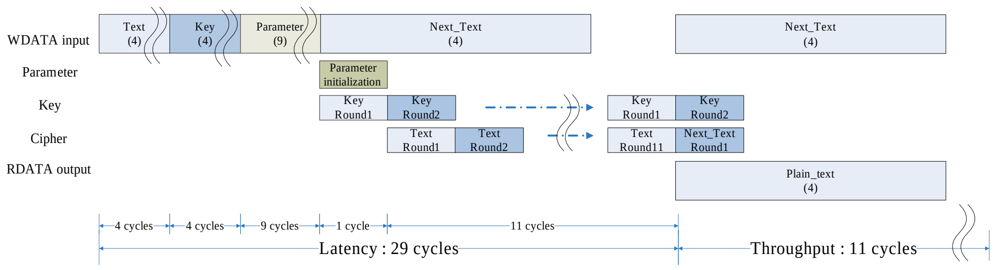


Figure 4.18: Clock distribution of the normal structure.

Once coefficients listed in Table 4.3 are given, an initial process is required. When encryption function is set, the initial process calculates $\Gamma^{-1}A$ in Fig. 4.17, and 32 8-bit registers, storing $\texttt{xtime}^n(c_i)$, for `MixColumns`. In decryption, $A^{-1}\Gamma$ and $\texttt{xtime}^n(c_i)$ is calculated in initial process. Besides, the key scheduler has to compute the final round key according to the given cipher key. Therefore, the latency of decryption is larger than that of encryption by 66 clock cycles in pipeline structure or 11 clock cycles in normal structure.

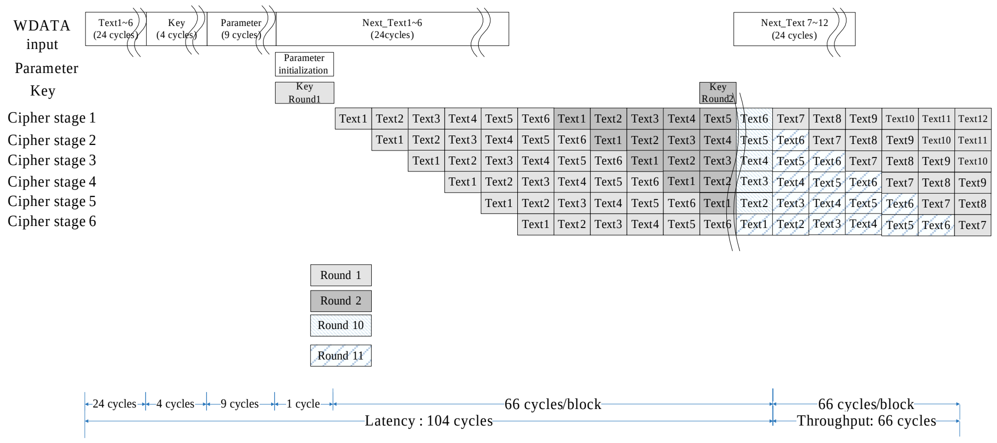


Figure 4.19: Clock distribution of the pipeline structure.

The clock cycles of the normal structure and the pipeline structure are depicted in Fig. 4.18 and Fig. 4.19. The two figures are illustrated under the assumption that the coefficients are given at the beginning of a transmission and not changed during a transmission. Obviously, in normal structure, the latency of the first block requires 29 clock cycles, and each following block requires 11 clock cycles to process. In the pipeline structure, the latency of the first block needs 104 cycles, and each following block requires 66 clock cycles.

4.11 Results

The gate counts and performance of the normal structure and the pipeline structure are described in Table 4.4. The throughput is estimated under the assumption that the coefficients are given at the beginning of a transmission, and not altered during the transmission, as shown in Fig. 4.18 and Fig. 4.19. We achieve 1.7902 Gbps and a 153 MHz on a 0.18-μm CMOS standard cell library for a speed-optimized normal structure, and 4.9516 Gbps and 425 MHz for a speed-optimized pipeline structure.

Table 4.4: Performance of the two proposed structures. The gate counts are obtained from 0.18-μm CMOS standard cell technology. C/B = cycles per block, GC = gate counts, MF = max frequency, Thr. = throughput, OF = optimization factor.

Structure	C/B	GC (k)	MF (MHz)	Thr. (Gbps)	OF
Normal	11	70.855	76.51	0.8903	area
	11	83.094	153.84	1.7902	speed
Pipeline	66	124.518	212.77	2.4758	area
	66	125.993	425.53	4.9516	speed

Table 4.5 shows the detail gate counts of each submodule in the normal and pipeline structures. The cipher core of the normal structure has gate counts of 63.392k and 75.482k, respectively, for area optimized and speed optimized configuration. The cipher core of the pipeline structure has gate counts of 98.719k and 96.198k, respectively, for area optimized and speed optimized configuration. However, the speed-optimized cipher core has less gate counts than area-optimized cipher core has; event hough, the entire gate counts of the speed-optimized design is larger than that of the area-optimized design.

Table 4.6 shows the performance comparisons. The function of changing coefficients needs massive hardware costs to achieve, although our results are also compared with other implementations of original AES. From Table 4.6, our results provide throughput over gigabit per second, and the gate counts do not hugely increase.

4.12 Conclusions

This work has proposed a simple, symmetric, and high-fault-coverage error detection schemes for AES. Although error bits are diffused in AES, this work used the linear behavior of each operation in AES to design CRC. This scheme only uses a $(n+1, n)$ CRC to detect the errors, where $n \in \{4, 8, 16\}$, and the parity of the output of each operation is predicted in a simple fashion. Even though the number of parities is two or four, respectively for $n = 8$ or $n = 4$, it is possible to use only one 8-bit register for storing the parities during hardware implementation. This error detection may also be used in encryption-only or decryption-only designs. Because of the symmetry, the encryption and decryption circuit can share the same

Table 4.5: The detail hardware cost of each submodule in the normal and pipeline structures. GC = gate counts; P. = percentage.

	Normal structure				Pipeline structure			
Submodules	Area optimized		Speed optimized		Area optimized		Speed optimized	
	GC (k)	P. (%)	GC (k)	P. (%)	GC (k)	P. (%)	GC (k)	P. (%)
P-Rijndael core	63.392	89.34	75.483	90.84	98.719	79.28	96.198	76.35
Control	**0.142**	**0.2**	**0.181**	**0.22**	**0.212**	**0.17**	**0.258**	**0.2**
register tables	**3.696**	**5.21**	**3.708**	**4.46**	**3.710**	**2.98**	**3.668**	**2.91**
Cipher	**34.774**	**49.06**	**46.797**	**56.32**	**59.213**	**47.55**	**58.087**	**46.10**
– Data register	0.981	1.38	0.981	1.18	5.546	4.45	5.546	4.40
– (Inv)MixColumns	16.447	23.2	20.423	24.58	20.908	16.79	21.358	16.95
– ShiftRows	0.324	0.46	0.861	1.03	0.324	0.26	0.555	0.44
– SubBytes	13.999	19.75	20.259	24.38	25.520	20.50	23.148	18.37
Key schedule	**24.717**	**34.87**	**24.797**	**29.84**	**35.584**	**25.58**	**34.176**	**27.13**
– Key registers	0.981	1.38	0.981	1.18	3.776	3.03	3.776	3.00
– InvMixColumns	16.329	23.04	16.329	24.58	20.743	16.66	19.215	15.25
– SubBytes	3.360	4.74	3.289	3.96	7.990	6.42	7.517	5.97
Input interface	4.799	6.77	4.853	5.84	10.896	8.73	11.878	9.43
Output interface	2.758	3.89	2.758	3.32	14.930	11.99	17.909	14.22
Total	70.885	100	83.094	100	124.518	100	125.993	100

error detection hardware. In addition, the proposed scheme can easily be implemented in a variety of structures, such as 8-bit, 32-bit or 128-bit structures.

A Rijndael algorithm with changeable coefficients is also designed in this work. Two architectures are proposed – the normal architecture and the pipeline architecture. The former provides 1.7902 Gbps and costs 83.094k gate counts on 0.18-μm CMOS cell standard library; the later provides 4.9516 Gbps and costs 125.993k gate counts. The goal of this design is providing customized security for *virtual private network* (VPN) application. In VPN, sessions do not need to compatible with standard traffics; hence, the enterprize can configure their own coefficients to protect their network. In addition, our designs provides

Table 4.6: Performance comparison. Ichikawa [24], Kuo [33], Satoh [57], Lin [38].

	[24]	[33]	[57]	[38]	Ours: normal case	Ours: normal pipeline
Technology (μm)	0.35	0.18	0.11	0.35	0.18	0.18
Clock rate (MHz)	N/A	N/A	224.22	200	153.84	425.53
Throughput (Gbps)	1.95	2.609	1.328	2.008	1.7902	4.9516
Gate counts (k)	612	173	21.337	58.430	83.094	125.993
Change coefficients	No	No	No	No	Yes	Yes

throughput over gigabit per seconds, so they are suitable for Fast Ethernet or Giga Ethernet.

Chapter 5

Future Works

5.1 Futures of Secrecy-Channel Coding

The structure of secrecy-channel coding proposed in Chapter 2 is too scalable to analysis its security. From the results listed in Section 2.6, only the randomness is examined. However, the structural security against attacks for stream ciphers, e.g., correlation attacks, linear consistency test, linear syndrome algorithm and linear cryptanalysis, is not investigated herein. First, the analysis works can be focused on the structure of one SHSR with a fixed random vector, and then expand the results to the entire system. Additionally, the PRVG can be replaced by a stream cipher. Because the PRVG is used to generate a fixed-length binary sequence, a stream cipher also can accomplish this kind of work. In general, the security of a well-designed stream cipher is stronger than that of PRVG. Therefore, it is possible to make the proposed secrecy-channel coding more secure than the original one.

5.2 Futures of Secure MP3

For a DRM solution on MP3 music, proposing only a multimedia cipher is insufficient. Several mechanisms, including key exchange, authentication, digital signature or building of a secure channel, are also required. For example, in this dissertation, the secure MP3 is finally deployed on a dual-core system; however, the channel between RISC and DSP is exposed to crackers for intercepting the unprotected traffics. Therefore, for a ideal secure environment, the secure channel has to be built between RISC and DSP. Besides, the management of content key and license distribution are another interesting research to complete this topic. In addition to complete the research, the proposed concept also can be put in use on other audio, image or video compression algorithm.

5.3 Futures of Two Results about AES

The results of error detection built in AES is fruitful, but a realization in hardware will certainly make the research more solid. How much the hardware will be brought in? How much decrement of throughput will be while performing encryption/decryption and error detection in the meantime. As for parameterizable Rijndael, its applications to customized security are interesting topics, e.g., the basis-invariant `MixColumns` matrix, the orthogonal `MixColumns` matrix. The basis-invariant matrix and its inverse are always mutual inverse without changing the representation of each entry, when the basis is changed. Therefore, we can use any irreducible polynomial $m(x)$ in a session, and the ciphertext always can be correctly decrypted. In other words, only $m(x)$ is modified, rather than every coefficients, during a session. This will save the time for initializing `MixColumns` matrix.

Bibliography

[1] Apple iTune, http://www.apple.com/itunes/.

[2] E. Barkan, and E. Biham, “In How Many Ways Can You Write Rijndael?”, *Proceedings of ASIACRYPT, Dec. 1-5, 2002*, pp. 160-175, Springer-Verlag, 2002.

[3] G. Bertoni, L. Brevegelieri, I. Koren, P. Maistri, and V. Piuri, “Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard,” *IEEE Trans. on Computers*, Vol. 52, No. 4, pp. 492-505, Apr. 2003.

[4] G. Bertoni, L. Brevegelieri, I. Koren, P. Maistri, and V. Piuri, “Detecting and Locating Faults in VLSI Implementations of the Advanced Encryption Standard,” *Proceedings of 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 3-5 Nov., 2003*, pp.105-113, 2003.

[5] G. Bertoni, L. Brevegelieri, I. Koren, and P. Maistri, “An Efficient Hardware-based Fault Diagnosis Scheme for AES: Performances and Cost,” *Proceedings of 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 10-13 Oct., 2004*, pp. 130-138, 2004.

[6] E. Biham, and A. Shamir, “Differential Cryptanalysis of the Data Encryption Standard,” *New York: Springer-Verlag*, 1993.

[7] E. Biham, and A. Shamir, “Differential Fault Analysis of Secret Key Cryptosystems,” *Advances in Cryptology - CRYPTO ’97*, LNCS 1294, pp.513-525, Springer-Verlag, 1997.

[8] A. Biryukov, and C. D. Cannière, “Block Ciphers and Systems of Quadratic Equations”, *Proceedings of Fast Software Encryption, Feb. 24-26, 2003*, pp. 274 - 289 , Springer-Verlag, 2003.

[9] M. Bystrom, and J.W. Modestino, "Combined Source-channel Coding Schemes for Video Transmission over an Additive White Gaussian Noise Channel," *IEEE Journal on Selected Areas in Communcations*, Vol 18, Issue 6, pp. 880-890, June 2000.

[10] H. Cheng, and X. Li, "Partial Encryption of Compressed Images and Videos," *IEEE Trans. on Signal Processing*, Vol. 48, Issue:8, pp. 2439-2451 August 2000.

[11] N. Courtois, and J. Pieprzyk, "Cryptanalysis of Block Ciphers with Overdefined Systems of Equation", *Proceedings of ASIACRYPT, Dec. 1-5, 2002*, pp.267-287, Springer-Verlag, 2002.

[12] J. Daemen, and V. Rijmen, "AES Proposal: Rijndael," *AES Algorithm Submission*, Sep. 3, 1999.

[13] A. Denis, and W. Kinsner, "Secure and Resilient Data Printed on Paper," *IEEE Canadian Conference on Electrical and Computer Engineering*, Vol. 1, pp. 245-248, 1999.

[14] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on A.E.S,", *Lecture Notes in Computer Science, Applied Cryptography and Network Security*, Vol. 2846, pp. 293-306, 2003.

[15] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm," *Proceedings of Cryptographic Hardware and Embedded Systems (CHES '04)*, pp. 357-370, 2004.

[16] N. Ferguson, R. Schroeppel, and D. Whiting, "A Simple Algebraic Representation of Rijndael", *Proceedings of Selected Areas in Cryptography, Aug. 16-17, 2001*, pp. 103-111, Springer-Verlag, 2001.

[17] J. Fuller, and W. Millan, "On Linear Redundancy in the AES S-Box", preprint available at http://eprint.iacr.org, August, 2002.

[18] L. Gang, A.N. Akansu, M. Ramkumar, X. Xie, "On-line Music Protection and MP3 Compression", *Proceedings of International Symposium on IEEE ,Intelligent Multimedia, Video and Speech Processing*, pp.13-16, 2001.

[19] L. Gang, A.N. Akansu, M. Ramkumar, “MP3 Resistant Oblivious Steganography”, *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 3, pp. 1365-1368, 2001.

[20] M. Goodwin. “Adaptive Signal Models : Theory , Algorithms, and Audio Applications”, Ph.D. thesis, University of California, Berkeley, 1997.

[21] K.P. Ho and J.M. Kahn, “Transmission of Analog Signals Using Multicarrier Modulation: a Combined Source-channel Coding Approach,” *IEEE Trans. on Communications*, Vol. 44, Issue 11, pp. 1432-1443, November 1996.

[22] T. Hauser, and C. Wenz, ”DRM under Attack”, *Lecture note in Compute Science*, Vol. 2770, pp. 206–223, Springer-Verlag Heidelberg, 2003.

[23] T. Hwang, and T.R.N. Rao, “Secret Error-correcting Codes (SECC),” *Advances in Cryptology-Crypto* ’88, pp. 540-563, Springer-Verlag.

[24] T. Ichikawa, T. Kasuya, and M. Matsui, “Hardware Evaluation of the AES Finalists,” *Proceedings of 3rd AES Candidate Conference*, NIST, pp.279V285, Apr. 2000.

[25] IEEE P1363/D13, “Standard Specifications for Public Key Cryptography,” Institute of Electrical and Electronics Engineers, Nov. 1999.

[26] iMUSIC, http://www.imusic.com/

[27] ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 11172-3, “*Coding of moving pictures and associated audio for digital storage media at up to about 1.5M bit/s, Part 3: Audio*,” 1993.

[28] ITU-R Recommendation BS.1116, “Methods for the Subjective Assessment of Small Impairment in Audio Systems Including Multichannel Sound Systems”, International Telecommunication Union, Geneva, Switzerland, 1994.

[29] ITU-R Recommendation BS.1387-1, “Method for Objective Measurements of Perceived Audio Quality”, Dec. 1998.

[30] M. S. Jeong, S. Kim, J. Sohn, and J. Y. Kang, “Finite Wordlength Effects Evaluation of the MPEG-2 Audio Decoder”, International Conference on Signal Processing Applications & Technology, pp.351–355, Jan. 1996

[31] R. Karri, K. Wu, P. Mishra, and Y. Kim, “Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 21, No. 12, pp. 1509-1517, Dec. 2002.

[32] R. Karri, G. Kuznetsov, and M. Goessel, “Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers,” *Proceedings of Cryptographic Hardware and Embedded Systems (CHES ’03)*, pp. 113-124, 2003.

[33] H. Kua, and I. Verbauwhede, “Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm,” *Proceedings of Cryptographic Hardware and Embedded Systems, May 14-16, 2001*, pp.53-67, Springer-Verlag, 2001.

[34] Lame Aint an MP3 Encoder (LAME), http://sourceforge.net/projects/lame/

[35] B. G. Lee, “A New Algorithm to Compute the Discrete Cosine Transform”, *IEEE Trans. On Acoustic, Speech and Signal Processing*, Vol. ASSP-32, NO.6, pp.1243–1245, 1984.

[36] K. H. Lee, K. S. Lee, T. H. Hwang, Y. C. Park, and D. H. Youn, “An Architecture and Implementation of MPEG Audio Layer III Decoder Using Dual-core DSP”, *IEEE Trans. on Consumer Electronics*, Vol. 47, No. 4, pp.928–933, Nov. 2001.

[37] K. S. Lee, H. O Oh, Y. C. Park, and D. H. Youn,“High Quality MPEG-audio Layer III algorithm for a 16-bit DSP”, *Proceedings of IEEE International Symposiums Circuit and Systems, May 6-9, 2001*, vol. II, pp.205-208, 2001.

[38] T.F. Lin, C.P. Su, C.T. Huang, and C.W. Wu, “A High-throughput Low-cost AES Cipher Chip,” *Proceedings of 3rd IEEE Asia-Pacific Conf. ASIC, Aug., 2003*, pp. 85-88, 2003.

[39] European Broadcasting Union, EBU SQAM, http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/

[40] S. Lin, "Error Control Coding: Fundamentals and Applications", Prentice-Hall, 1983, pp. 350-359.

[41] S. Mangard, M. Aigner, and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," *I*EEE Trans. on Computers, Vol. 52, No. 4, pp. 483-491, Apr. 2003.

[42] R.J. McEliece, "A Public Key Cryptosystem Based on Algebraic Coding Theory," *DSN Progress Report* 42-44, pp. 114-116, Jet Propulsion Laboratory, CA, January and February,1978.

[43] W. Meier and O. Staffebach, "Fast Correlation Attacks on Certain Stream Ciphers," *Journal of Cryptology*, Vol. 1, No.3, pp. 159-167, 1989.

[44] J.W. Modestino, D.G. Daut, and A.L. Vickers, "Combined Source-Channel Coding of Images Using the Block Cosine Transform," *IEEE*, Vol. IT-33, pp. 827-837. March 1981.

[45] S. Murphy, and M.J.B. Robshaw, "Essential Algebraic Structure Within the AES", *Proceedings of CRYPTO, Aug. 18-22, 2002*, pp.17-38, Springer-Verlag, 2002.

[46] National Institute of Standards and Technology, *FIPS 180-1: Secure Hash Standards*, U.S. Department of Commerce, April 17,1995(supersedes FIPS PUB 180).

[47] National Institute of Standards and Technology, *FIPS 197: Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, Federal Information Processing Standard, FIPS-197, 2001, available at URL, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[48] National Institute of Standards and Technology, *FIPS 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, Federal Information Processing Standard, FIPS 800-22, Dec., 2000.

[49] H. Niederreiter, "Random Number Generation and Quasi-Monte Carlo Methods," Society of Industrial and Applied Mathematics, 1992, pp. 206-207.

[50] P. Noll,"MPEG Digital Audio Processing", *IEEE Signal Processing Magazine*, pp.59–81, Sep. 1997.

[51] H. O Oh, J. S. Kim, C. J. Song, Y. C. Park and D. H. Youn, "Low Power MPEG/Audio Encoders Using Simplified Psychoacoustics Model and Fast Bit Allocation", *IEEE Transaction on Consumer Electronics*, Vol.47, No.3, pp.613–621 Aug. 2001.

[52] D. Pan, "A Tutorial on MPEG/Audio Compression", *IEEE Trans. on Multimedia*, Vol.2, No.2, pp.60–74, 1995.

[53] G. Piret, and J.J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD," *Cryptographic Hardware and Embedded Systems - CHES '03*, LNCS 2779, pp. 77-88, 2003, Springer-Verlag.

[54] T.R.N. Rao and K. Nam, "Private-key Algebraic Cryptosystems," *Advances in Cryptology-Crypto* '86, pp. 35-48, Springer-Verlag, 1986.

[55] T.R.N. Rao and K. Nam, "Private-key Algebraic-code Encryptions," *IEEE Transactions on Information Theory*, Vol. IT-35, No. 4, pp. 829-833, July 1989.

[56] P. Rogaway and D. Coppersmith, "A Software-Optimized Encryption Algorithm", *Proceedings of Fast Software Encryption, Cambridge Security Workshop* , Springer-Verlag, Vol. 809, pp. 56-63, 1994.

[57] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proceedings of ASIACRYPT, Dec. 9-13, 2001,* pp. 239-254, Springer-Verlag, 2001.

[58] T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext Only," *IEEE Transactions on Computing*, Vol. C-341, pp. 81-85, 1985.

[59] D. R. Stinson, "Cryptography: Theory and Practice," CRC press, 1995, pp. 20-21.

[60] N.J. Thorwirth, P. Horvatic, R. Weis, and Jian Zha, "Security Methods for MP3 Music Delivery", *Proceedings of the* 23th *Asilomar Conference on IEEE Signals, Systems and Computers*, Vol. 2, pp.1831-1835, 2000.

[61] A. Torrubia, F. Mora, "Perceptual Cryptography on MPEG-1 Layer III Bit-streams", *Proceedings of International Conference on IEEE Consumer Electronics*, pp.324-325, 2002.

[62] X. Wang, W. DOU and Z. HOU, “An Improved Audio Encoding Architecture Based on 16-bit Fixed-point DSP”, IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions, Vol. 2, pp.918-921, Jul. 2002.

[63] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, “Low Cost Concurrent Error Detection for the Advanced Encryption Standard,” *International Test Conference (ITC ’04)*, pp. 1242-1248, 2003.

[64] K. Zeng, C.H. Yang, and T.R.N. Rao, “Cryptanalysis of the Hwang-Rao Secret Error-Correcting Code Schemes”, Lecture Notes in Computer Science, Vol. 2229, pp. 419-428, Springer-Verlag, 2001.

[65] M. Zhang, “Maximum Correlation Analysis of Nonlinear Combining Functions in Stream Ciphers,” *Journal of Cryptology*, Vol. 13, pp. 301-313, 2000.