



國立台灣科技大學
電子工程系

碩士學位論文（初稿）

設計與實現一個應用於 IoT 之 AES-GCM 加密認證
演算法硬體架構

Design and Implementation of a Hardware Architecture
of the AES-GCM Algorithm for IoT Applications

林品妤

M11002160

指導教授：林銘波 博士

中華民國 112 年 月 日

指導教授推薦書

論文口試委員會審定書

致謝

摘要

物聯網的快速發展使得資料交換和通信需求大量提升，同時資料安全和隱私保護的需求也提升。具有高效和安全特性的 AES-GCM 被廣泛應用於資料加密和認證中，然而在物聯網中的邊緣計算或是嵌入式系統裝置之硬體資源是有限的。因此，在本論文中，我們設計與實現一個符合資源有限的物聯網環境的 AES-GCM 加密認證演算法之低面積的硬體架構。

為了達到低面積的需求，在 AES 演算法中的位元組替代轉換模組，根據硬體的資源特性，提供各自最佳化的設計方法。在 FPGA 中，使用直接邏輯映射的方法實現，可以節省約 32.9% 的 LUTs 數量。而在 ASIC 中，使用複合場運算的方法實現，可以節省約 55.6% 的等效邏輯閘數量。在 GHASH 模組中，為了降低硬體資源，使用 Karatsuba 乘法演算法實現的有限場乘法器，可以節省約 50.7% 的硬體資源。

完成的 AES-GCM 加解密設計分別使用 FPGA 與 ASIC 實現與驗證。在 FPGA 實現上，使用 Xilinx 公司的 Virtex 7 系列的 XC7VX330T，其合成結果之工作頻率為 181.917 MHz，最高吞吐量為 1293.632 Mbps，使用 2767 個 Registers、8801 個 LUTs 及 2815 個 slices。在 ASIC 實作上，使用 tsmc 0.18 μm 製程元件庫，其合成結果之工作頻率為 83.682 MHz，最高吞吐量為 595.072 Mbps，晶片核心面積為 $1047.345 \mu m \times 1051.26 \mu m$ ，等效邏輯閘數量約為 71677 個，核心功率消耗為 19.1656 mW，I/O Pad 功率消耗為 1.4866 mW。

關鍵詞：AES-GCM、進階加密標準、Karatsuba 演算法、認證加密、FPGA、ASIC。

ABSTRACT

The rapid development of the Internet of Things (IoT) has led to a significant increase in data exchange and communication demands, along with the need for data security and privacy protection. The AES-GCM algorithm, known for its efficiency and security features, is widely used for such requirements of data encryption and authentication. Nevertheless, the hardware resources are limited in the edge-computing devices or embedded systems used in IoT. As a consequence, in this thesis we design and implement a low-area hardware architecture based on the AES-GCM algorithm in compliance with the resource-limited IoT environment.

In order to achieve the low area requirement, an optimized design method is applied for the byte substitution module in the AES algorithm, in accordance with the hardware resource characteristics. In FPGA implementation, the method of direct logic mapping reduces about 32.9% of the number of LUTs. In ASIC implementation, the method of composite field decreases about 55.6% of the equivalent gate count. In the GHASH module, a finite field multiplier implemented using the Karatsuba multiplication algorithm saves about 50.7% of the hardware resources.

The resulting architecture of the AES-GCM encryption and decryption algorithm is implemented and verified by both FPGA and ASIC technologies. In FPGA implementation, a device (XC7VX330T) of Xilinx's Virtex 7 series is used and can operate at a frequency of 181.917 MHz, achieving a maximum throughput of 1293.632 Mbps, with 2767 registers, 8801 LUTs, and 2815 slices. In ASIC implementation, the cell library of the tsmc 0.18 μm process is employed and the resulting chip can operate at a frequency of 83.682 MHz in simulation, achieving a maximum throughput of 595.072 Mbps, with a core area of $1047.345 \mu m \times 1051.26 \mu m$, equivalent to 71677 gates. The core power consumption is 19.1656 mW while the I/O pad power consumption is 1.4866 mW.

Keywords: AES-GCM, advanced encryption standard (AES), Karatsuba algorithm, authenticated encryption, FPGA, ASIC.

目錄

致謝	I
摘要	II
ABSTRACT	III
目錄	V
圖目錄	VIII
表目錄	XI
第 1 章 緒論	1
1.1 研究動機	1
1.2 研究方向	1
1.3 章節介紹	2
第 2 章 AES-GCM 加、解密演算法介紹	3
2.1 密碼學	3
2.1.1 對稱式密碼學	3
2.1.2 非對稱式密碼學	4
2.2 區塊密碼工作模式	6
2.2.1 電子密碼本模式	6
2.2.2 密碼區塊連結模式	7
2.2.3 密文反饋模式	8
2.2.4 輸出反饋模式	9
2.2.5 計數器模式	10
2.3 訊息認證碼	11
2.4 伽羅瓦有限場	13
2.4.1 加法運算	14

2.4.2 乘法運算	15
2.5 AES 演算法	17
2.5.1 金鑰擴展	20
2.5.2 AES 加密演算法	23
2.5.3 AES 解密演算法	28
2.6 AES-GCM 演算法	32
2.6.1 AES-GCM 加密演算法	33
2.6.2 AES-GCM 解密演算法	35
2.6.3 GCTR 加密	37
2.6.4 GHASH 雜湊	38
第 3 章 設計與分析	39
3.1 AES 回合運算單元	39
3.1.1 位元組替代轉換模組	39
3.1.2 列位移轉換模組	44
3.1.3 混合行轉換模組	45
3.1.4 加入回合金鑰模組	46
3.2 金鑰擴展單元	47
3.2.1 金鑰擴展單元設計方法分析	47
3.2.2 金鑰擴展單元設計	49
3.3 GHASH 運算單元	49
3.3.1 Karatsuba 演算法	50
3.3.2 Karatsuba GF(2^{128})乘法器	51
第 4 章 AES-GCM 硬體架構	57
4.1 AES-GCM 矽智財架構	57
4.2 AES 運算模組	59
4.3 GHASH 運算模組	60

4.4 AES-GCM 資料路徑模組	62
4.5 AES-GCM 控制單元	63
第 5 章 FPGA 與 ASIC 實現	65
5.1 FPGA 模擬與實現	65
5.1.1 行為模擬結果	66
5.1.2 佈局結果	68
5.1.3 FPGA 設計結果	68
5.2 標準元件庫設計與實現	69
5.2.1 Design Compiler 模擬與實現結果	70
5.2.2 IC Compiler 模擬與實現結果	71
5.2.3 LVS 與 DRC 結果	73
5.3 效能分析與比較	74
5.3.1 效能分析方式	74
5.3.2 FPGA 效能比較	75
5.3.3 ASIC 效能比較	76
第 6 章 結論與未來展望	77
參考文獻	78

圖目錄

圖 2.1 對稱式密碼學加、解密流程	4
圖 2.2 非對稱式密碼學加、解密流程	5
圖 2.3 數位簽章流程	5
圖 2.4 ECB 模式加密流程[7].....	6
圖 2.5 ECB 模式解密流程[7].....	7
圖 2.6 CBC 模式加密流程[7]	8
圖 2.7 CBC 模式解密流程[7]	8
圖 2.8 CFB 模式加密流程[7].....	9
圖 2.9 CFB 模式解密流程[7].....	9
圖 2.10 OFB 模式加密流程[7].....	10
圖 2.11 OFB 模式解密流程[7].....	10
圖 2.12 CTR 模式加密流程[7].....	11
圖 2.13 CTR 模式解密流程[7].....	11
圖 2.14 訊息認證流程[7].....	12
圖 2.15 認證綁定明文流程[7]	13
圖 2.16 認證綁定密文流程[7]	13
圖 2.17 AES 狀態矩陣.....	18
圖 2.18 AES 金鑰狀態矩陣， $Nk = (a)4$ 、 $(b)6$ 、 $(c)8$	18
圖 2.19 AES 不同金鑰所對應的 Nb 、 Nk 、 Nr	18
圖 2.20 AES 加、解密流程.....	19
圖 2.21 金鑰擴展虛擬碼.....	21
圖 2.22 AES-128 金鑰擴展架構	22
圖 2.23 AES 加密演算法虛擬碼.....	23
圖 2.24 位元組替代轉換操作	24

圖 2.25 列位元轉換操作	26
圖 2.26 混合行轉換操作	27
圖 2.27 加入回合金鑰操作	28
圖 2.28 AES 解密演算法虛擬碼	29
圖 2.29 反列位元轉換操作	31
圖 2.30 AES-GCM 加密流程圖	35
圖 2.31 AES-GCM 解密流程圖	36
圖 2.32 GCTR 加密操作	37
圖 2.33 GHASH 雜湊操作	38
圖 3.1 乘法反元素架構圖[13]	40
圖 3.2 乘法反元素內部子電路， x^2	42
圖 3.3 乘法反元素內部子電路， $\times \lambda$	42
圖 3.4 乘法反元素內部子電路， $GF(2^4)$ 乘法器	42
圖 3.5 乘法反元素內部子電路， $GF(2^8)$ 乘法器	42
圖 3.6 乘法反元素內部子電路， $\times \varphi$	43
圖 3.7 位元組替代轉換電路架構[13]	43
圖 3.8 列位移轉換電路架構	44
圖 3.9 $xtime$ 電路架構	46
圖 3.10 混合行轉換電路架構	46
圖 3.11 加入回合金鑰電路架構	47
圖 3.12 傳統有限場乘法器	50
圖 3.13 Karatsuba 演算法虛擬碼	51
圖 3.14 Karatsuba 有限場乘法器	52
圖 3.15 Karatsuba $GF(2^{128})$ 乘法器	55
圖 4.1 AES-GCM 矽智財方塊圖	57
圖 4.2 AES-GCM 設計之內部架構圖	59

圖 4.3 AES 運算模組電路架構圖	60
圖 4.4 GHASH 運算模組電路架構	61
圖 4.5 Karatsuba GF(2^{128})乘法器電路架構[22].....	62
圖 4.6 AES-GCM 資料路徑模組	63
圖 4.7 AES-GCM 控制單元 ASM 圖	64
圖 5.1 加密模式_前置輸入	67
圖 5.2 加密模式_輸出	67
圖 5.3 解密模式_前置輸入	67
圖 5.4 解密模式_輸出	67
圖 5.5 佈局後加密模式模擬_前置輸入	68
圖 5.6 佈局後加密模式模擬_輸出	68
圖 5.7 佈局後解密模式模擬_前置輸入	68
圖 5.8 佈局後解密模式模擬_輸出	68
圖 5.9 FPGA 硬體資源使用報告	69
圖 5.10 FPGA 時序分析報告	69
圖 5.11 Cell-based IC 設計流程	70
圖 5.12 合成後模擬波形圖_加密	71
圖 5.13 合成後模擬波形圖_解密	71
圖 5.14 佈局後模擬波形圖_加密	72
圖 5.15 佈局後模擬波形圖_解密	72
圖 5.16 晶片電路圖	73
圖 5.17 LVS 驗證	74
圖 5.18 DRC 驗證	74

表目錄

表 2.1 AES 演算法專有名詞解釋.....	17
表 2.2 AES-128 加密與解密每所需的回合金鑰	20
表 2.3 AES 加密替換盒.....	25
表 2.4 AES 解密替換盒.....	30
表 2.5 AES-GCM 演算法專有名詞解釋	33
表 3.1 位元組替代轉換模組之 FPGA 合成結果比較.....	44
表 3.2 位元組替代轉換模組之 ASIC 合成結果比較.....	44
表 3.3 有限場乘法器之空間複雜度比較	53
表 3.4 Karatsuba 乘法器資源消耗比較.....	54
表 3.5 有限場乘法器之資源消耗比較[22]	54
表 4.1 AES-GCM 矽智財腳位功能定義	58
表 4.2 AES 運算模組腳位功能定義	59
表 4.3 GHASH 運算模組腳位功能定義	60
表 5.1 AES-GCM 測試範例	66
表 5.2 晶片規格	72
表 5.3 FPGA 相關文獻比較.....	75
表 5.4 ASIC 相關文獻比較	76

第1章 緒論

本章節將探討論文的研究動機和研究方向，並對論文中各個章節的內容進行簡要介紹。

1.1 研究動機

隨著物聯網[1] (Internet of Things, 簡稱 IoT) 技術的迅速發展，越來越多的裝置和系統進入到網絡中，形成了龐大的物聯網生態系統。然而，這種大規模的連接性也帶來了許多安全風險和挑戰。在這個日益互聯的環境中，保護敏感資料和確保通信的安全性變得至關重要。資料的安全性是一個關鍵問題。不僅包括資料的機密性，還包括資料的完整性和真實性。

為了解決上述問題，必須採取有效的加密和認證機制來保護物聯網中的數據。其中，高級加密標準—加密認證模式 (Advanced Encryption Standard-Galois/Counter Mode, 簡稱 AES-GCM) [2][3] 是一種廣泛使用的加密演算法，它結合了對稱式加密和消息驗證碼 (MAC) 技術，同時提供了數據的機密性和完整性。

儘管 AES-GCM 已被廣泛應用於網絡和通信安全領域，但在物聯網中的應用仍然存在一些挑戰。由於物聯網設備的資源受限，例如計算能力和能源供應，需要設計出適合於物聯網環境的低面積 AES-GCM 實現。因此，本研究的動機在於設計和實現一個適用於物聯網的 AES-GCM 加密認證演算法之硬體架構，以解決現有實現在物聯網環境中所面臨的問題。

1.2 研究方向

本論文的研究方向是設計一個低面積的 AES-GCM 加密認證演算法電路，並將其應用於物聯網 (IoT) 領域。具體而言，將研究如何設計 AES-CTR 運算模組，以及研究如何簡化複雜度高的 GHASH 運算模組。探討不同的設計方法和技術，以減少電路

的面積，同時確保運算模組的性能和安全性。透過分析和優化，我們將尋找最佳的實現方式，以降低電路面積並試圖維持一定程度的運算效率。

1.3 章節介紹

本論文一共分為六個章節，其內容如下：

- 第 1 章：緒論，包含研究動機、研究方向與章節介紹。
- 第 2 章：AES-GCM 加、解密演算法介紹，包含區塊密碼工作模式、訊息認證碼以及伽羅瓦有限場的基本概念。
- 第 3 章：分析不同硬體電路架構的優缺點，包含資料管線化、回合運算單元、金鑰擴展單元以及 GHASH 運算單元。
- 第 4 章：說明本論文所設計的 AES-GCM 硬體架構。
- 第 5 章：FPGA 與 ASIC 實現的模擬驗證，總結模擬結果以及效能分析。
- 第 6 章：總結整篇論文。

第2章 AES-GCM 加、解密演算法介紹

本章闡述了 AES-GCM 加、解密演算法相關的概念。首先會先介紹密碼學、常見的區塊密碼模式和訊息認證碼的基本原理，再說明伽羅瓦有限場的數學運算，最後詳細的說明 AES-GCM 與 AES 演算法的結構和運作原理，為後續章節的設計和實現奠定了基礎。

2.1 密碼學

密碼學[4]是保護信息安全和通信保密的學科領域。它探討並開發加、解密演算法、密碼系統和安全協議，以確保只有有權的人可以訪問敏感信息，並確保數據在傳輸過程中不會被非正常讀取或修改。其目標是確保訊息在傳輸和存儲過程中的機密性、完整性、身分認證和不可否認性。機密性確保只有授權的接收方能夠解讀和存取訊息，防止第三方獲取敏感資訊；完整性確保訊息在傳輸過程中沒有被篡改或修改，確保訊息的完整性和可靠性；身分認證用於驗證通訊雙方的身份，確保通訊對象的真實性和可信性；不可否認性則提供了對通訊雙方交易的證明，防止任何一方否認其行為或責任。

密碼學包括兩種基本類型的加密技術：對稱式密碼學和非對稱式密碼學。

2.1.1 對稱式密碼學

對稱式密碼學 (Symmetric Cryptography) [5]也稱為對稱金鑰演算法 (Symmetric-key Algorithm)、私鑰加密、共享金鑰加密。所謂對稱式，就是加密和解密時使用相同的金鑰。

對稱式加、解密流程如圖 2.1 所示，尚未經過加密的原始訊息或經過解密後的訊息稱為明文 (Plaintext)，明文經過加密演算法處理變為不可讀取或理解的訊息稱為密文 (Ciphertext)；加密 (Encryption) 是指為了保護訊息，發送方使用金鑰 (Key) 將

明文加密，使得訊息變為無法輕易解讀的密文；而解密（Decryption）則是正確的轉換訊息，接收方收到密文後，會使用與發送方相同的金鑰將密文解密，使得密文變為可解讀的明文訊息。

對稱式密碼學具有更快的運算速度，可應用於吞吐量需求較高的場合，在計算機系統中被廣泛用於保護信息，但由於加、解密都是使用相同的金鑰，因此如何在安全通信的前提下管理和共享金鑰是對稱式密碼學所需面對的難題。

常見的對稱加密演算法包括 DES(Data Encryption Standard)、3DES(Triple Data Encryption Standard)、AES(Advanced Encryption Standard)等。其中進階加密標準(AES) [8]是最熱門的，此加密技術支援 128、192 或 256 位元金鑰，是全球執行加密產業的標竿。AES 通常與 Galois/Counter Mode(GCM)結合使用，即 AES-GCM，是可產生經過身分驗證的加密演算法。

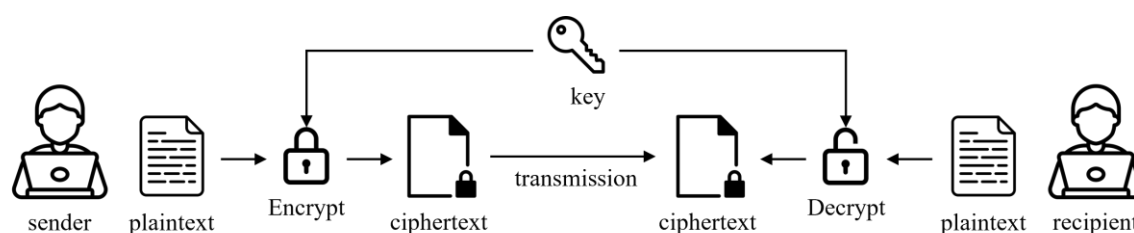


圖 2.1 對稱式密碼學加、解密流程

2.1.2 非對稱式密碼學

非對稱式密碼學（Asymmetric cryptography）[5]也稱為公開金鑰密碼學（Public-key cryptography）。所謂非對稱式，就是加密和解密時使用不同的金鑰。

非對稱式加、解密流程如圖 2.2 所示，使用者有一對金鑰，供傳送方使用。其中可公開並且可任意發布的加密金鑰稱為公開金鑰（Public key），供接收方使用；不可公開並且需要妥善保存的解密金鑰稱為私密金鑰（Public key）。加密時，發送方使用公鑰將明文加密，而解密時，接收方使用相對應的私鑰將密文解密。

非對稱式密碼學因運算速度較低，適用於吞吐量需求較低的場合或小量的資料加密。且由於加密所需的公鑰是可公開，即便攻擊者取得也無法解密，所以金鑰傳輸相對於對稱式密碼學容易。

基於公開金鑰加密的特性，還能提供數位簽章（Digital Signature）的功能，使電子檔案可以得到如同在紙本檔案上親筆簽署的效果，可確保資料的完整性、真實性和不可否認性，大部份國家已經立法承認數位簽章擁有等同傳統親筆簽章的法律效力。數位簽章流程如圖 2.3 所示，傳送方使用自己的私鑰對該封密文簽名，當接收方收到時，使用傳送方的公鑰來對簽章作二次驗證，確認真的是由正確的傳送者傳來的。

常見的對稱加密演算法包括 RSA(Rivest-Shamir-Adleman)、ECC(Elliptic Curve Cryptography)、DSA(Digital Signature Algorithm)等。

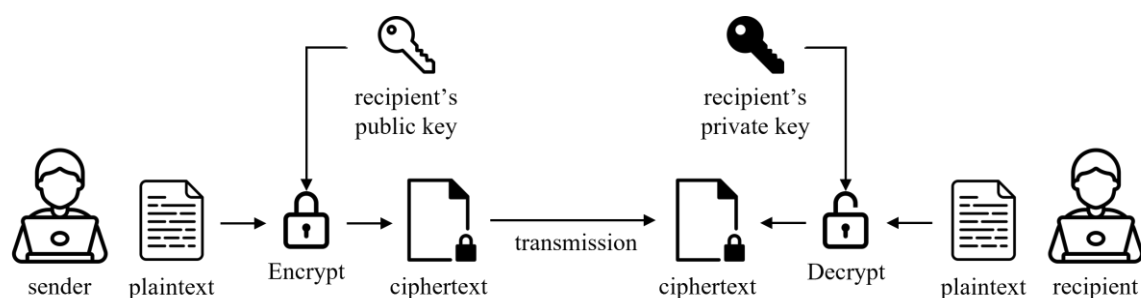


圖 2.2 非對稱式密碼學加、解密流程

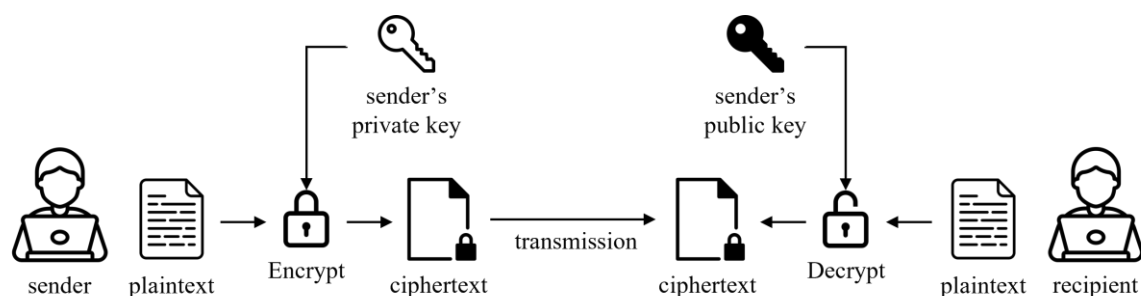


圖 2.3 數位簽章流程

2.2 區塊密碼工作模式

區塊密碼工作模式（Block Cipher Modes of Operation）[6]是對稱式密碼學中用於對長訊息進行加密的標準方法。這些工作模式將長度不固定的訊息分割成多個固定大小的區塊，並使用適當的填充方法來處理最後一個區塊的長度不足的情況。常見的區塊密碼工作模式：電子密碼本模式、密碼區塊連結模式、密文反饋模式、輸出反饋模式和計數器模式。

2.2.1 電子密碼本模式

電子密碼本模式（Electronic Codebook, ECB）是最簡單的加密模式。ECB 模式加密流程如圖 2.4 所示，明文按照區塊密碼的區塊大小被分為數個區塊，用同樣的金鑰對每個區塊進行獨立加密，再將其串接變為最終的密文。ECB 模式解密流程與加密相反，如圖 2.5 所示。

明文區塊與密文區塊類似一對一的關係，兩個相同的明文區塊，就會產出一樣的密文區塊，且因為區塊間互不相關，不會互相影響，每個區塊可以同時加密，所以速度很快，適用於加密小區塊數據或需要對特定區塊進行獨立處理的場景，但也容易遭受攻擊，不適合用於資料長度較長的訊息。

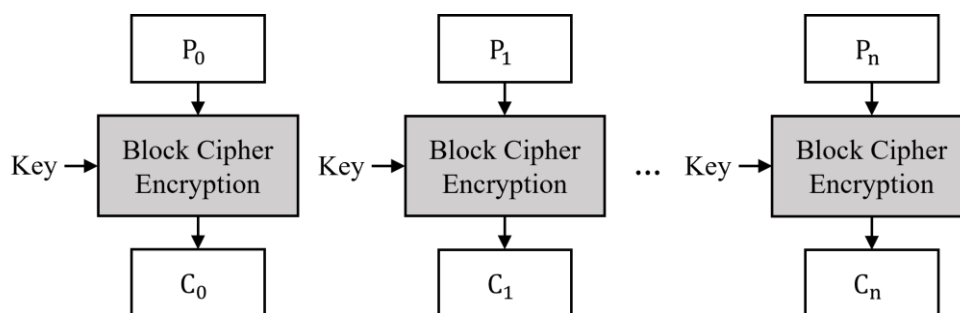


圖 2.4 ECB 模式加密流程[7]

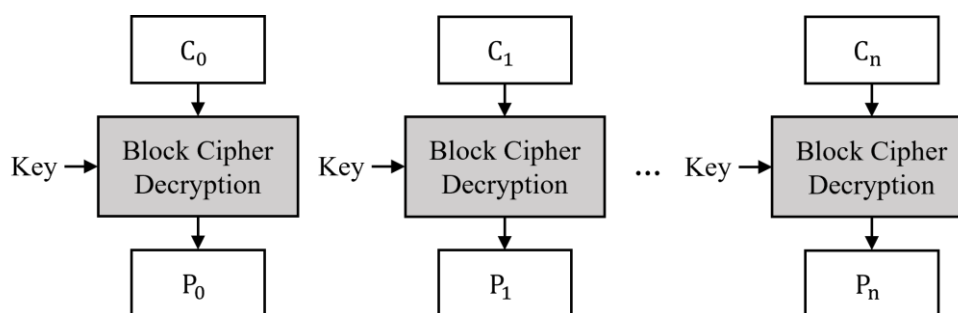


圖 2.5 ECB 模式解密流程[7]

2.2.2 密碼區塊連結模式

密碼區塊連結模式 (Cipher Block Chaining, CBC) 加入了初始化向量 (Initialization Vector, IV) 的概念。IV 是隨機的，長度為分組大小，通常同一金鑰不應多次使用同一組 IV。

CBC 模式加密流程如圖 2.6 所示，明文分割成固定大小的區塊，第一個明文區塊與 IV 進行互斥或運算後，將運算結果進行加密，生成第一個密文區塊，再將第一個密文區塊與第二個明文區塊進行互斥或運算並加密生成第二個密文區塊，以此類推直到所有明文區塊都變為對應的密文區塊。CBC 模式解密流程與加密相反，如圖 2.7 所示。

每個密文區塊的生成都依賴前一個密文區塊，使得攻擊者無法直接對單個區塊進行修改或插入，因為會導致後續密文區塊的改變。同時，也使用 IV 作為第一個區塊的加密參數，確保每次加密的結果都不相同，提供機密性和完整性，並且相對於 ECB 模式更具安全性。但需要保護好 IV，並確保每次加密使用的 IV 都是唯一的，以避免攻擊者利用重複 IV 進行攻擊。

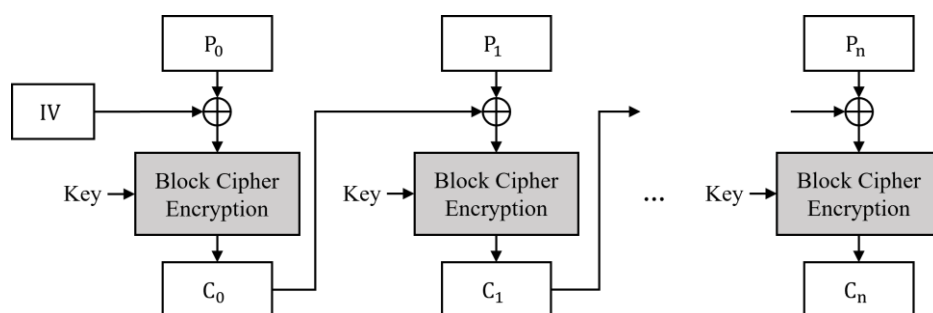


圖 2.6 CBC 模式加密流程[7]

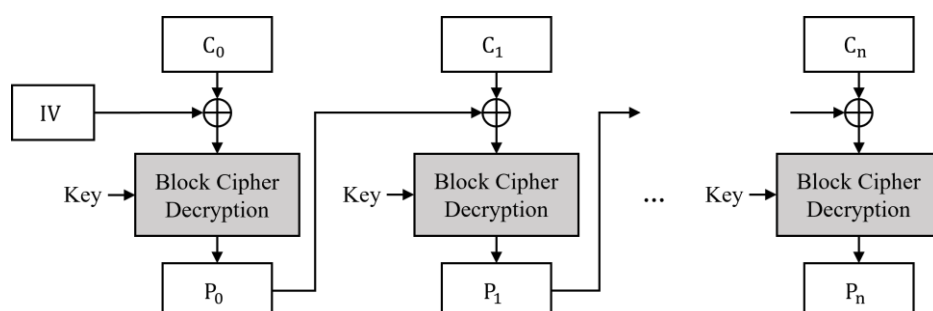


圖 2.7 CBC 模式解密流程[7]

2.2.3 密文反饋模式

密文反饋模式（Cipher Feedback, CFB）可以將區塊密碼轉換為串流密碼（Stream Cipher）。串流密碼無需將每筆資料的長度補齊到區塊長度，即可進行即時加密處理。

CFB 模式加密流程如圖 2.8 所示，初始向量作為加密演算法的輸入，將結果與第一個明文區塊進行互斥或運算，得到第一個密文，再將第一個密文加密與第二個明文區塊進行互斥或運算，得到第二個密文，以此類推直到所有資料傳輸完畢。CBC 模式解密流程與加密類似，且加、解密都是使用同個加密演算法，如圖 2.9 所示。

CFB 模式具有自同步性，無需外部同步信號，每個密文的生成只依賴於前一個密文，不會影響後續的解密操作，使得在傳輸過程中丟失或錯誤接收某些密文時，不會影響解密的正確性。使用串流密碼，使得長度彈性減少資料的填充操作，且適合於需要即時處理的應用場景。但需要注意傳輸錯誤擴散和密文依賴順序的問題，在實際應

用中，需仔細評估安全需求和加密算法的選擇，並使用適當的初始向量和區塊大小以確保安全性和效能。

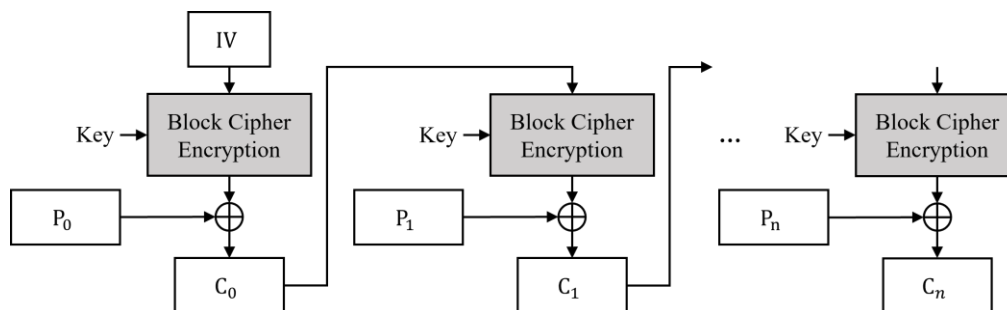


圖 2.8 CFB 模式加密流程[7]

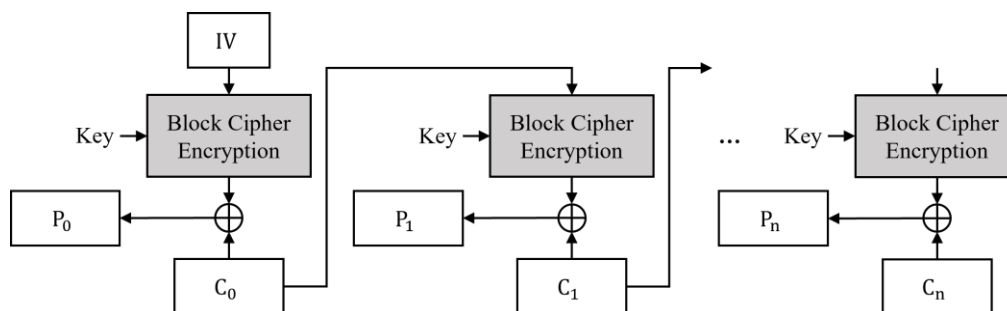


圖 2.9 CFB 模式解密流程[7]

2.2.4 輸出反饋模式

輸出反饋模式（Output Feedback, OFB），與密文反饋模式結構相似，也可以將區塊密碼轉換為串流密碼，但在運算的過程中有一些差異。當開始處理第二筆資料時，將前一個加密演算法的輸出作為當前加密演算法的輸入，如圖 2.10 所示。OFB 模式解密流程與加密類似，且加、解密都是使用同個加密演算法，如圖 2.11 所示。

OFB 模式的優點是具有良好的錯誤傳播特性，並能夠提供較高的安全性。如果明文中的某個位元發生錯誤，該錯誤僅會影響當前的密文輸出，不會將錯誤擴散到後續的所有密文中，適用於串流加密場景。但由於每個區塊的加密都是獨立的，因此無法

同時進行平行處理，導致效能較差，且密碼流的生成是基於加密演算法，若演算法存在弱點或受到攻擊，則可能導致整個加密系統的安全性受到威脅。

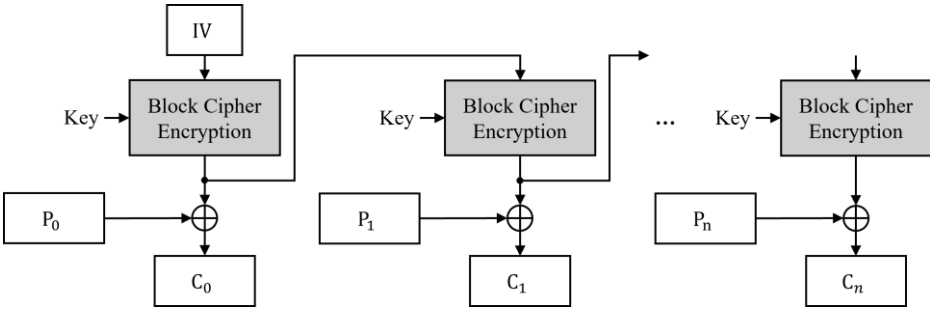


圖 2.10 OFB 模式加密流程[7]

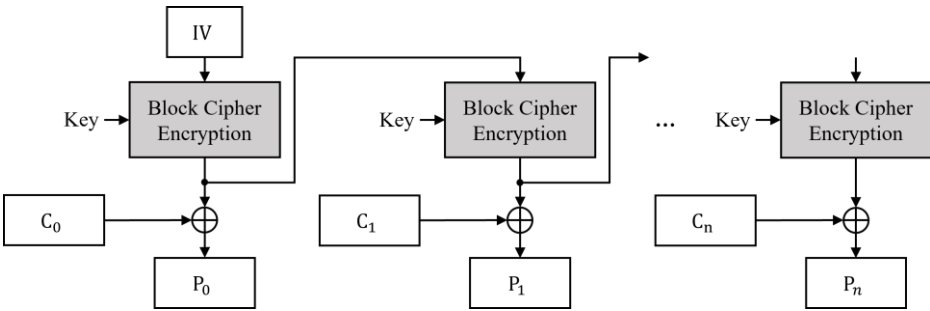


圖 2.11 OFB 模式解密流程[7]

2.2.5 計數器模式

計數器模式 (Counter Mode, CTR) 可以將區塊加密算法應用於串流加密，且引入一個計數器，以保證任意長時間均不會產生重複輸出。計數器的初始值通常被稱為計數器的 Nonce (一次性使用的數字)，每次加密時，計數器的值會被遞增。

CTR 模式加密流程如圖 2.12 所示，使用一個不斷遞增的計數器作為加密演算法的輸入，將結果與每一個明文區塊進行互斥或運算，得到相對應的密文。CTR 模式解密流程與加密相反，且加、解密都是使用同個加密演算法，如圖 2.13 所示。

CTR 模式的優點是它能夠提供高效的並行加密處理以提升吞吐量。由於每個區

塊的加密獨立於其他區塊，因此可以同時進行多個區塊的加密操作，從而加快整體的加密速度。此外，CTR 模式還具有良好的錯誤傳播特性，因為明文區塊的錯誤只會影響相應的密文區塊，而不會擴散到其他部分。

AES-GCM 演算法使用了計數器模式，使得在硬體設計上可以實現平行處理和加、解密共用的架構，提高加、解密的吞吐量並降低硬體資源的浪費。

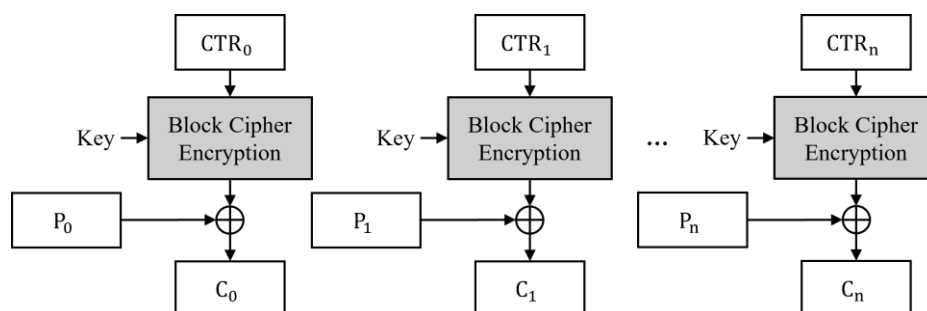


圖 2.12 CTR 模式加密流程[7]

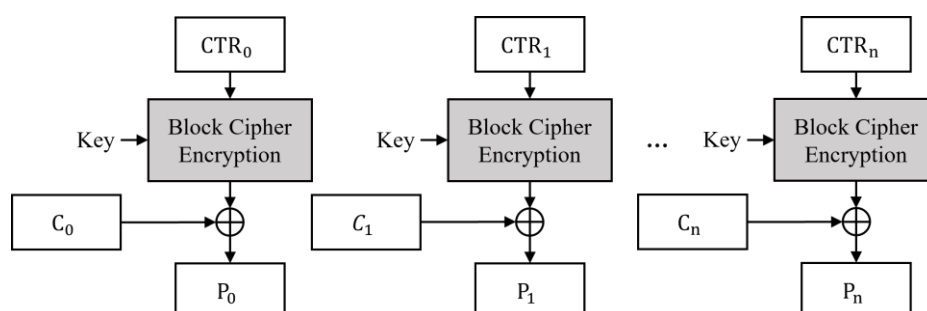


圖 2.13 CTR 模式解密流程[7]

2.3 訊息認證碼

在密碼學中，訊息認證碼（Message Authentication Code, MAC）[7][8]也稱為訊息鑑別碼、檔案訊息鑑別碼、消息認證碼、資訊認證碼，是數位簽章的對稱版本。只用於訊息的認證，不會對訊息進行加密。經過特定演算法後產生的一小段資訊，不僅用於驗證訊息的完整性，檢查在訊息傳遞過程中，其內容是否被竄改，還提供不可否認

性，即發送方無法否認發送該訊息，以及作身分驗證，確認訊息的來源。

訊息認證碼的流程如圖 2.14 所示，傳送方傳送訊息時，將訊息與一個金鑰通過訊息認證函數生成一個固定長度的 MAC 並附加在訊息後，傳送給接收方；接收方收到訊息時，將不包含 MAC 的訊息與一個金鑰通過同樣的訊息認證函數生成一個相對應的 MAC，並將其與傳送方傳來的 MAC 進行比較，如果兩者相等，則可以確定這段訊息是來自合法的傳送方且未遭到竄改。

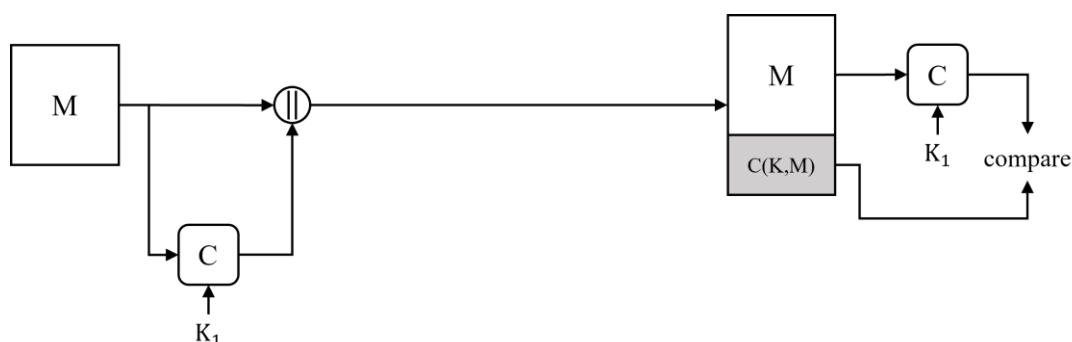


圖 2.14 訊息認證流程[7]

若要同時實現訊息加密和認證，就需要與加密演算法搭配，在進行加密的同時也對訊息進行認證，以確保訊息的機密性和完整性。加密認證的方式分為兩種：認證綁定明文與認證綁定密文。

認證綁定明文流程如圖 2.15 所示，傳送方傳送訊息時，將訊息與一個金鑰通過訊息認證函數生成一個固定長度的 MAC 並附加在訊息後，再進行加密與傳送；接收方收到訊息時，需要先進行解密，才能做明文的認證。

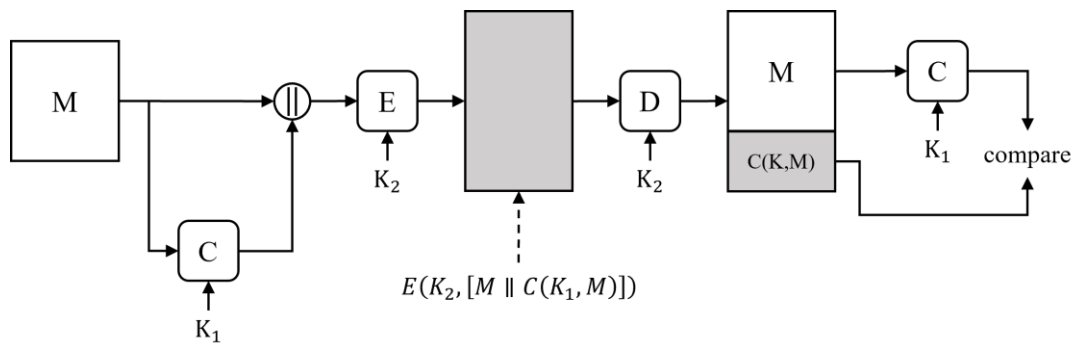


圖 2.15 認證綁定明文流程[7]

認證綁定密文流程如圖 2.16 所示，傳送方傳送訊息時，將訊息先加密，再與一個金鑰通過訊息認證函數生成一個固定長度的 MAC 並附加在密文後，傳送給接收方；接收方收到訊息時，密文會同時進行解密與認證。

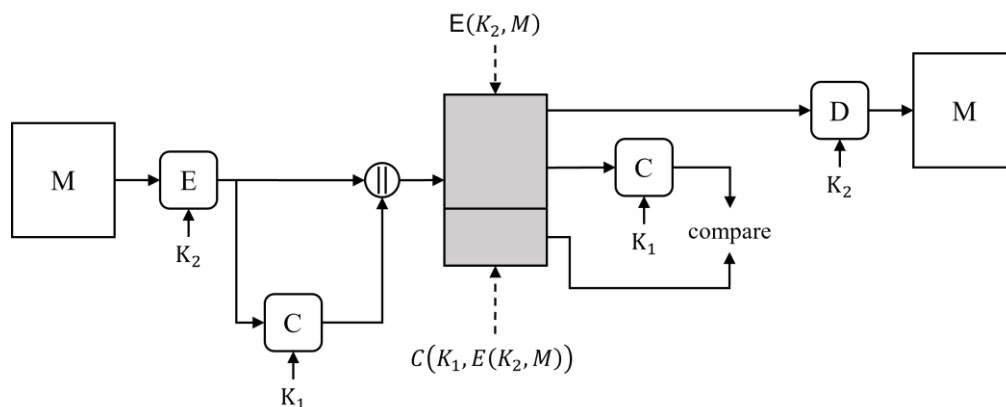


圖 2.16 認證綁定密文流程[7]

AES-GCM 演算法，採用的是認證綁定密文的方式。傳送方使用 AES-CTR 模式將訊息進行加密，再將密文輸入到 GHASH 函數中生成 MAC；接收方在接收到密文後，同時進行認證和解密操作，並確認解密後的訊息的完整性。

2.4 伽羅瓦有限場

伽羅瓦有限場 (Galois Field, GF) 也稱為有限場 (Finite Field)，是密碼學中常用

的數學運算，與一般數學運算的定義不同，可以有效處理運算溢位（Overflow）的問題[9]。

伽羅瓦有限場在 AES-GCM 中扮演了重要的角色，用於實現加密和身份驗證的運算，例如多項式的乘法和除法操作。它有助於提供高效率的加密和認證機制，允許在硬體和軟體實現中進行並行處理，從而提高效能和效率，同時也確保了 AES-GCM 的安全性和可靠性，使其成為一種廣泛應用於資訊安全領域的加密模式。

伽羅瓦有限場是一種有限的數字集合，集合中的元素被稱為場元素（Field Elements）或簡稱為元素，可以是 0 到（有限域大小 - 1）之間的整數。例如 $GF(2^8)$ 表示一個大小為 256 的有限域，其中的元素可以是 0 到 255 之間的整數。

在 AES-GCM 中，有限域中的運算是基於二元的，即所有數值都只能是 0 或 1。AES 加、解密演算法中使用了 $GF(2^8)$ 的加法和乘法，每次處理資料的單元為 8 個位元相當於一個位元組（Byte），每個位元組代表 $GF(2^8)$ 中的一個元素，每個元素都可以對應到一個一元七次多項式。例如：位元組 A 為 $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$ ，則 A 可以表示為以下多項式，如式(2-1)所示。

$$A(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \quad (2-1)$$

其中 $a_7 \sim a_0$ 只能為 0 或 1。例如： $(57)_{10} = (01010111)_2$ ，則可以表示為一元七次多項式 $x^6 + x^4 + x^2 + x + 1$ 。

而 MAC 生成演算法 GHASH 則是使用 $GF(2^{128})$ 的加法和乘法運算，每次處理資料的單元為 128 個位元；伽羅瓦有限場的加法和乘法運算遵循特定的規則，以下將介紹兩個的基本數學運算。

2.4.1 加法運算

在伽羅瓦有限場中，加法運算符號以「+」表示。定義為兩個元素中相同次方的係數相加並做模 2 運算，即對應元素之間進行位元的互斥或（以符號 \oplus 表示）運算。

例如： $(57)_{16} + (83)_{16} = (d4)_{16}$ ，如下列所示。

$$(57)_{16} = (01010111)_2 = x^6 + x^4 + x^2 + x + 1$$

$$(83)_{16} = (10000011)_2 = x^7 + x + 1$$

$$\text{則 } (57)_{16} + (83)_{16} = (01010111)_2 \oplus (10000011)_2 = (11010100)_2 = (d4)_{16}$$

$$\text{以多項式表示為 } (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

2.4.2 乘法運算

在伽羅瓦有限場中，乘法運算符號以「 \cdot 」表示。定義為在 $GF(2^8)$ 中的相乘結果必須在 $GF(2^8)$ 的集合內，但相乘可能會產生溢位的問題，為了解決溢位問題，則須將相乘結果再做模運算 (Modulo) 一個不可分解的多項式 $m(x)$ 。在 AES 演算法中，定義 $m(x)$ 此多項式如式(2-2)。例如： $(57)_{16} \cdot (83)_{16} = (c1)_{16}$ ，如下列所示。

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (2-2)$$

$$(57)_{16} = (01010111)_2 = x^6 + x^4 + x^2 + x + 1$$

$$(83)_{16} = (10000011)_2 = x^7 + x + 1$$

$$\text{則 } (x^6 + x^4 + x^2 + x + 1) \cdot (x^7 + x + 1)$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

因相乘結果大於七次方，超出 $GF(2^8)$ 的集合，發生溢位問題，故須做模運算。

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod (x^8 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + 1 = (11000001)_2 = (c1)_{16}$$

為了利於硬體架構的實現，還有另一種乘法運算稱為 *xtime*[10]。定義為使用分配律將被乘數的項次拆開並搭配 *xtime* 運算後，再將各自運算的結果做互斥或運算得到結果。例如： $(57)_{16} \cdot (2a)_{16} = (27)_{16}$ ，如下列所示。

$$(63)_{16} \cdot (2a)_{16} = (27)_{16}$$

$$(63)_{16} \cdot (02)_{16} = \textit{xtime}(63) = (c6)_{16}$$

$$(63)_{16} \cdot (04)_{16} = \textit{xtime}(c6) = (97)_{16}$$

$$(63)_{16} \cdot (08)_{16} = \textit{xtime}(97) = (35)_{16}$$

$$(63)_{16} \cdot (10)_{16} = \textit{xtime}(35) = (6a)_{16}$$

$$(63)_{16} \cdot (20)_{16} = \textit{xtime}(6a) = (d4)_{16}$$

$$(63)_{16} \cdot (2a)_{16} = (63)_{16} \cdot [(02)_{16} \oplus (08)_{16} \oplus (20)_{16}]$$

$$= (c6)_{16} \oplus (35)_{16} \oplus (d4)_{16}$$

$$= (27)_{16}$$

2.5 AES 演算法

高級加密標準 (Advanced Encryption Standard, AES) [11] 也稱為 Rijndael 演算法，為比利時密碼學家 Joan Daemen 和 Vincent Rijmen 所設計，由美國國家標準與技術研究院 (NIST) 於 2001 年發佈於 FIPS PUB 197，並在 2002 年成為有效的標準，取代了原先被證實耗時且不安全的資料加密標準 (Data Encryption Standard, DES)。

AES 演算法是對稱式密碼學，故只會有一個金鑰，且採用區塊加密，一個區塊為 128 位元。為了說明 AES 演算法，以下將會解釋此演算法中常見的專有名詞，如表 2.1 所示。

表 2.1 AES 演算法專有名詞解釋

專有名詞	解釋
狀態 (State)	AES 演算法中的內部狀態，將輸入的明文或密文轉為矩陣。如圖 2.17 所示，矩陣大小為 $4 \times Nb$ ，以位元組為單位所構成。
回合金鑰 (Round key)	根據主金鑰生成的子金鑰，為每一回合所使用的金鑰矩陣。如圖 2.18 所示，矩陣大小為 $4 \times Nk$ ，以位元組為單位所構成。
加解密區塊數目 (Nb)	狀態矩陣中的行數（一行表示 32 位元）。如圖 2.19 所示，AES 演算法中 Nb 固定為 4。
金鑰區塊數目 (Nk)	金鑰矩陣中的行數（一行表示 32 位元）。如圖 2.19 所示，根據 AES 的金鑰長度，對應不同的 Nk。
運算回合次數 (Nr)	加、解密所需運算的回合次數。如圖 2.19 所示，根據 AES 的金鑰長度，對應不同的 Nr。
金鑰 (Key)	對稱式密碼學中，加密和解密使用的機密值。根據金鑰長度可分為 AES-128、AES-196 和 AES-256。
明文 (Plaintext)	未經過加密的訊息或資料，可以直接得知原始內容。在 AES 演算法中，明文被分成固定大小的區塊進行加密。
密文 (Ciphertext)	經過加密後的訊息或資料，無法直接從中得知原始內容。在 AES 演算法，對明文進行加密後會得到對應的密文。

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

圖 2.17 AES 狀態矩陣

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

(a)

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$	$K_{0,4}$	$K_{0,5}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$

(b)

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$	$K_{0,4}$	$K_{0,5}$	$K_{0,6}$	$K_{0,7}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$	$K_{1,6}$	$K_{1,7}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$	$K_{2,6}$	$K_{2,7}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$	$K_{3,6}$	$K_{3,7}$

(c)

圖 2.18 AES 金鑰狀態矩陣， $Nk =$ (a)4、(b)6、(c)8

	Nb	Nk	Nr
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

圖 2.19 AES 不同金鑰所對應的 Nb 、 Nk 、 Nr

AES 加、解密流程如圖 2.20 所示。接下來會說明金鑰擴展以及加、解密的細節。

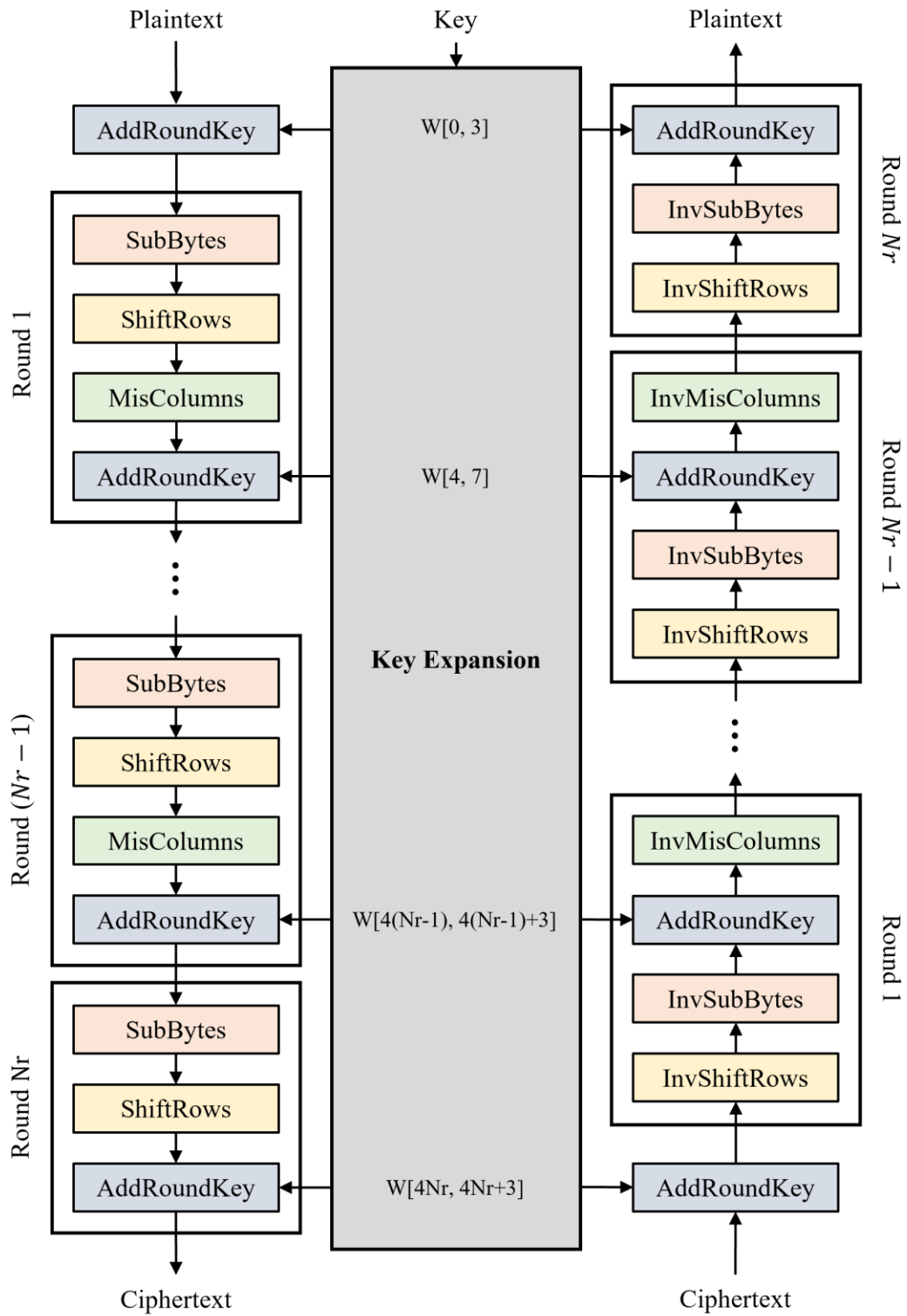


圖 2.20 AES 加、解密流程

2.5.1 金鑰擴展

金鑰擴展 (KeyExpansion) 是對稱式加密算法中的一個重要步驟，它確保了加密和解密過程的一致性和有效性，且因為每個加密回合都使用不同的子金鑰，也增加了加密的安全性。

原始金鑰透過此程序進行擴展和變換，生成每一回合所需的回合金鑰。不同的金鑰長度，所對應的回合金鑰個數也不同。加密過程中，金鑰擴展總共會產生 $Nb(Nr + 1)$ 個行字元，包含一組初始金鑰與 Nr 組回合金鑰。每個行字元由 4 個位元組組成，表示為 $w[i]$ ，其中 $0 < i < Nb(Nr + 1)$ 。每回合所需的金鑰為 $w[i] \sim w[i + 3]$ 。例如：AES-128，總共會產生 $4 \times (4 + 1) = 44$ 個行字元， $w[1] \sim w[3]$ 為初始金鑰， $w[4] \sim w[7]$ 為第一回合所需的回合金鑰。解密過程中，每回合所使用的擴充金鑰，則為反向提取加密過程中產生的擴充金鑰。如表 2.2 所示。

表 2.2 AES-128 加密與解密每所需的回合金鑰

回合數	加密回合金鑰	解密回合金鑰
0	$w[0] \sim w[3]$	$w[40] \sim w[43]$
1	$w[4] \sim w[7]$	$w[36] \sim w[39]$
2	$w[8] \sim w[11]$	$w[32] \sim w[35]$
3	$w[12] \sim w[15]$	$w[28] \sim w[31]$
4	$w[16] \sim w[19]$	$w[24] \sim w[27]$
5	$w[20] \sim w[23]$	$w[20] \sim w[23]$
6	$w[24] \sim w[27]$	$w[16] \sim w[19]$
7	$w[28] \sim w[31]$	$w[12] \sim w[15]$
8	$w[32] \sim w[35]$	$w[8] \sim w[11]$
9	$w[36] \sim w[39]$	$w[4] \sim w[7]$
10	$w[40] \sim w[43]$	$w[0] \sim w[3]$

金鑰擴展虛擬程式碼如圖 2.21 所示。AES-128 金鑰擴展流程如圖 2.22 所示。

```
KeyExpansion (byte key[4 * Nk] , word w[Nb * (Nr + 1)] , Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word (key[4 * i] , key[4 * i + 1] , key[4 * i + 2] , key[4 * i + 3] )
        i = i + 1
    end while
    i = Nk
    while (i < Nb * (Nr+1) )
        temp = w[i - 1]
        if (i mod Nk = 0)
            temp = SubWord (RotWord (temp) ) xor Rcon[i / Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord (temp)
        end if
        w[i] = w[i - Nk] xor temp
        i = i + 1
    end while
end
```

圖 2.21 金鑰擴展虛擬碼

主要包含三個運算方式，SubWord()的功能為接收一個行字元，經過替換盒（S-box）將行字元中每一個位元組轉換，產生相對應的行字元。RotWord()的功能是將行字元中經過一個位元組左旋運算，例如： $w[i] = \{a_0, a_1, a_2, a_3\}$ 經過位元組左旋後，變為 $w[i] = \{a_1, a_2, a_3, a_0\}$ 。Rcon 為回合常數，根據此回合數計算出該回合的回合常數，定義如式(2-3)所示，公式中的乘法為伽羅瓦有限場之運算。

$$\begin{cases} Rcon[i] = \{temp[i], 0, 0, 0\} \\ temp[i] = 2 \cdot temp[i - 1] \\ temp[1] = 1 \end{cases} \quad for \ 1 \leq i \leq Nr \quad (2-3)$$

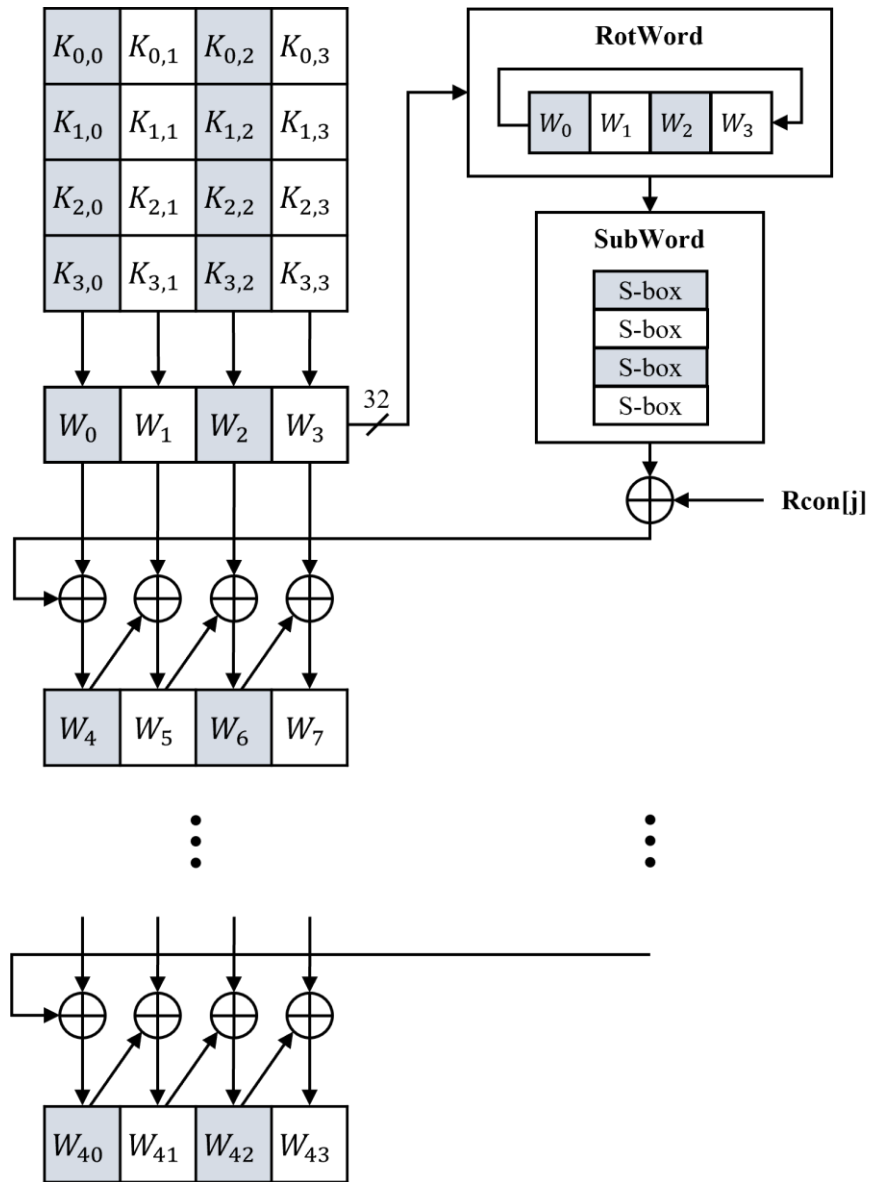


圖 2.22 AES-128 金鑰擴展架構

2.5.2 AES 加密演算法

AES 演算法會根據不同的金鑰長度對應到不同的加解密回合數，加密回合包括 $Nr - 1$ 個完整回合以及一個不完整的最後回合。

加密虛擬碼如圖 2.23 所示。加密回合中有四個運算單元，分別為位元組替代 (SubBytes) 轉換、列位移 (ShiftRows) 轉換、混合行 (MixColumns) 轉換以及加入回合金鑰 (AddRoundKey) 轉換。

```
Cipher (byte in[4 * Nb] , byte out[4*Nb] , word w[Nb * (Nr + 1) ] )
begin
    byte state[4 , Nb]
    state = in
    AddRoundKey (state , w[0 , Nb - 1] )
    for round = 1 step 1 to Nr - 1
        SubBytes (state)
        ShiftRows (state)
        MixColumns (state)
        AddRoundKey (state , w[round * Nb , (round + 1) * Nb - 1] )
    end for
    SubBytes (state)
    ShiftRows (state)
    AddRoundKey (state , w[Nr * Nb , (Nr + 1) * Nb - 1] )
    out = state
end
```

圖 2.23 AES 加密演算法虛擬碼

2.5.2.1 位元組替代轉換

位元組替代轉換(SubBytes Transformation)是一種非線性的轉換，藉此混淆明文，增加了加密的複雜性和安全性。將狀態矩陣中的每個位元組透過替換盒（S-box）轉換為另一個位元組，得到一個新的狀態矩陣。如圖 2.24 所示。

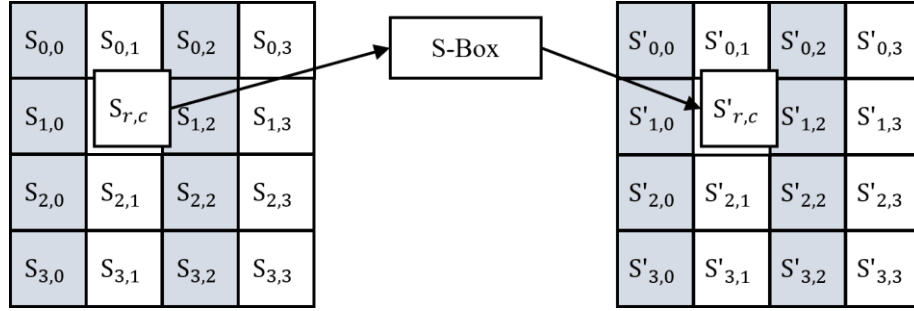


圖 2.24 位元組替代轉換操作

加密演算法中的替換盒是由以下兩個運算所組成：

- (1) 乘法反元素：將欲轉換的位元組以有限場 $GF(2^8)$ 表示，求出其在有限場 $GF(2^8)$ 的乘法反元素。有限域的乘法反元素定義為在有限場中，對於一個非零元素 Z ，存在一個元素使得 $(Z \cdot Z^{-1}) \bmod m(x) = 1$ ，則稱 Z^{-1} 為 Z 的乘法反元素。
- (2) 仿射（Affine）運算：轉換式如式(2-4)所示，其中 c_i 為常數位元組 $\{63\}_{16}$ 中的第 i 個位元。也可以矩陣型態表示，如式(2-5)。

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (2-4)$$

$$\text{for } 0 \leq i < 8$$

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (2-5)$$

以上兩個運算可以算出替換盒中的所有數值。表 2.3 列出 AES 加密演算法所使用的替換盒，替換盒大小為 16×16，包含了 256 個不同的替換值，表中數值以 16 進制表示。查表方式為使用位元組中最高有效位元之四位元視為列（row）位址，最低有效位元之四位元視為行（column）位址，即可找到相對應的替換值。

表 2.3 AES 加密替換盒

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

2.5.2.2 列位移轉換

列位移轉換（ShiftRows Transformation）是對狀態矩陣的每一列進行循環左移操作，藉此增加密文的分散性和複雜性。將狀態矩陣的第一列保持不變，第二列循環左移一個位置，第三列循環左移兩個位置，第四列循環左移三個位置。如圖 2.25 所示。

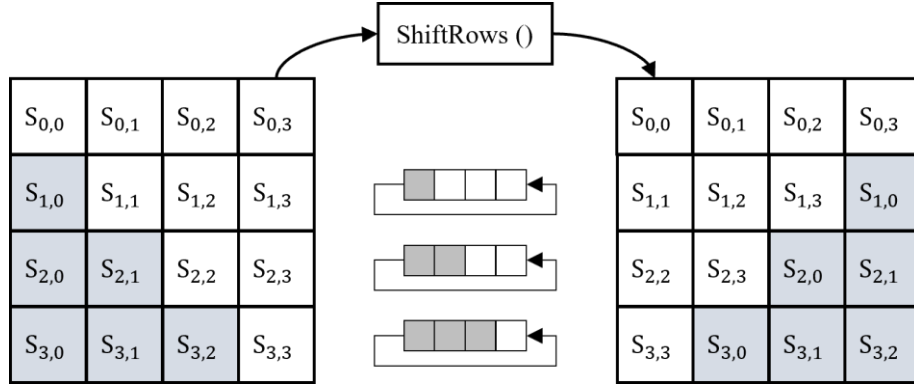


圖 2.25 列位元轉換操作

2.5.2.3 混合行轉換

混合行轉換（MixColumns Transformation）用於對狀態矩陣每一行的四個位元進行線性轉換互相混合，增加密文的擴散性和安全性。

將狀態矩陣中每一行視為一個在 $GF(2^8)$ 上的多項式，並使用固定多項式 $a(x)$ 進行乘法運算，如式(2-6)及式(2-7)所示。若發生溢位再將其模於 $x^4 + 1$ 。

$$a(x) = \{03\}x^3 + \{03\}x^2 + \{01\}x + \{02\} \quad (2-6)$$

$$S'(x) = a(x) \otimes S(x) \quad (2-7)$$

也可將狀態矩陣中每一行的四個位元組視為一個四元素的向量，並使用有限域乘法，乘以一個固定的混合矩陣，如式(2-8)所示。矩陣相乘後的結果，如式(2-9)所示。

圖 2.26 為混合行轉換操作示意圖。

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad for \ 0 \leq c < Nb \quad (2-8)$$

$$\begin{cases} S'_{0,c} = (\{02\} \cdot S_{0,c}) \oplus (\{03\} \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} = S_{0,c} \oplus (\{02\} \cdot S_{1,c}) \oplus (\{03\} \cdot S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \cdot S_{2,c}) \oplus (\{03\} \cdot S_{3,c}) \\ S'_{3,c} = (\{03\} \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \cdot S_{3,c}) \end{cases} \quad (2-9)$$

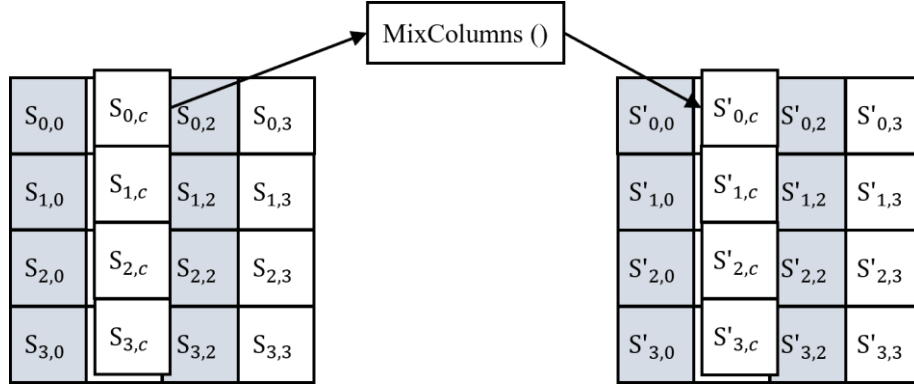


圖 2.26 混合行轉換操作

2.5.2.4 加入回合金鑰轉換

加入回合金鑰轉換(AddRoundKey Transformation)，是對狀態矩陣做互斥或運算，增加密文的混淆度和安全性。將每回合金鑰擴展的回合金鑰矩陣與當前的狀態矩陣進行互斥或運算，形成新的狀態矩陣。如式(2-10)所示。圖 2.27 為加入回合金鑰操作示意圖，其中 W_i 為金鑰擴展生成的第 i 個行字元。

$$\begin{aligned} [S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] &= [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [W_{round \cdot Nb + c}] \\ &\text{for } 0 \leq c < Nb, 0 \leq round < Nr \end{aligned} \quad (2-10)$$

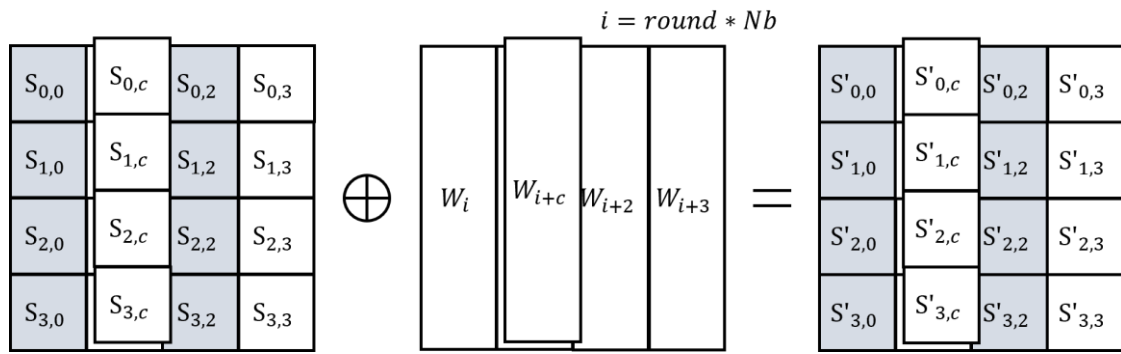


圖 2.27 加入回合金鑰操作

2.5.3 AES 解密演算法

AES 演算法在加、解密上擁有對稱性，回合數與加密相同，故將加密演算法逆向操作即為解密演算法。

解密虛擬碼如圖 2.28 所示。解密回合中有四個運算單元，分別為反位元組替代（InvSubBytes）轉換、反列位移（InvShiftRows）轉換、反混合行（InvMixColumns）轉換以及加入回合金鑰（AddRoundKey）轉換。

```

InvCipher (byte in[4 * Nb] , byte out[4 * Nb] , word w[Nb * (Nr + 1) ] )
begin
    byte state[4 , Nb]
    state = in
    AddRoundKey (state , w[Nr * Nb , (Nr + 1) * Nb-1] )
    for round = Nr - 1 step -1 downto 1
        InvShiftRows (state)
        InvSubBytes (state)
        AddRoundKey (state , w[round * Nb , (round + 1) * Nb - 1] )
        InvMixColumns (state)
    end for
    InvShiftRows (state)
    InvSubBytes (state)
    AddRoundKey (state, w[0 , Nb-1] )
    out = state
end

```

圖 2.28 AES 解密演算法虛擬碼

2.5.3.1 反位元組替代轉換

反位元組替代轉換 (InvSubBytes Transformation) 是位元組替代轉換的逆向運算。因為位元組替代轉換具有可逆性，故可將狀態矩陣中的每個位元組透過反替換盒 (Inverse S-box) 還原為未經替代轉換的數值。反替換盒如表 2.4 表 2.4 AES 解密替換盒所示。反位元組運算也可以透過乘法反元素以及仿射運算組成。

表 2.4 AES 解密替換盒

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

2.5.3.2 反列位移轉換

反列位移轉換 (InvShiftRows Transformation) 是對狀態矩陣的每一列進行循環右移操作。將狀態矩陣的第一列保持不變，第二列循環右移一個位置，第三列循環右移兩個位置，第四列循環右移三個位置。如圖 2.29 所示。

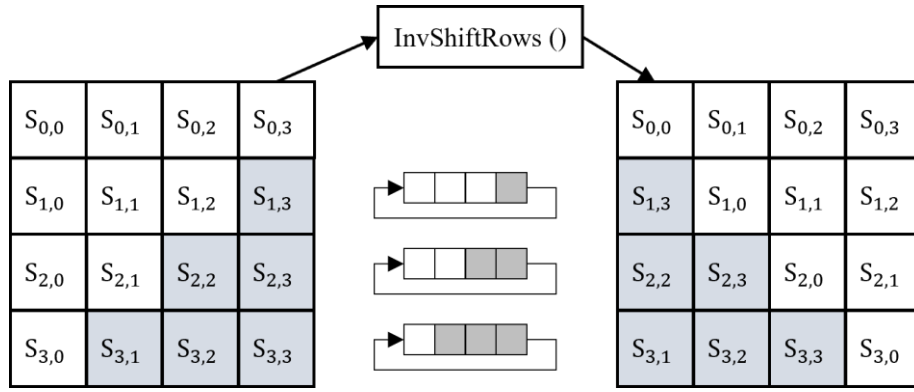


圖 2.29 反列位元轉換操作

2.5.3.3 反混合行轉換

反混合行轉換 (InvMixColumns Transformation) 是將狀態矩陣中每一行視為一個在 $GF(2^8)$ 上的多項式，並使用固定多項式 $a(x)$ 進行乘法運算，如式(2-11)及式(2-12)所示。

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \quad (2-11)$$

$$S'(x) = a^{-1}(x) \otimes S(x) \quad (2-12)$$

也可將狀態矩陣中每一行的四個位元組視為一個四元素的向量，並使用有限域乘法，乘以一個固定的混合矩陣，如式(2-13)所示。矩陣相乘後的結果，如式(2-14)所示。

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < Nb \quad (2-13)$$

$$\begin{cases} S'_{0,c} = (\{0e\} \cdot S_{0,c}) \oplus (\{0b\} \cdot S_{1,c}) \oplus (\{0d\} \cdot S_{2,c}) \oplus (\{09\} \cdot S_{3,c}) \\ S'_{1,c} = (\{09\} \cdot S_{0,c}) \oplus (\{0e\} \cdot S_{1,c}) \oplus (\{0b\} \cdot S_{2,c}) \oplus (\{0d\} \cdot S_{3,c}) \\ S'_{2,c} = (\{0d\} \cdot S_{0,c}) \oplus (\{09\} \cdot S_{1,c}) \oplus (\{0e\} \cdot S_{2,c}) \oplus (\{0b\} \cdot S_{3,c}) \\ S'_{3,c} = (\{0b\} \cdot S_{0,c}) \oplus (\{0d\} \cdot S_{1,c}) \oplus (\{09\} \cdot S_{2,c}) \oplus (\{0e\} \cdot S_{3,c}) \end{cases} \quad (2-14)$$

2.5.3.4 加入回合金鑰轉換

加入回合金鑰轉換 (AddRoundKey Transformation)，是將每回合金鑰擴展的回合金鑰矩陣與當前的狀態矩陣進行互斥或運算，形成新的狀態矩陣。與加密運算不同的地方在於使用回合金鑰的順序是相反的，即第一回合使用第 Nr 個回合金鑰，第二回合使用第 $Nr - 1$ 個回合金鑰，以此類推。

2.6 AES-GCM 演算法

AES-GCM (Advanced Encryption Standard-Galois/Counter Mode) 是一種結合了高級加密標準 (Advanced Encryption Standard, AES) 和伽羅瓦/計數器模式 (Galois/Counter Mode)，G 就是指伽羅瓦消息認證碼 (Galois Message Authentication Code, GMAC)，C 就是指 CTR 操作模式。提供了同時加密和身份驗證的功能，確保資料傳輸時的機密性 (confidentiality) 和完整性 (integrity)，是一種高效且安全的加密模式，常被用於保護敏感資料的傳輸和儲存。目前屬於最嚴謹的加密模式，TLS 1.2 標準使用的就是 AES-GCM 演算法。

AES-GCM 具有幾個重要的優點。首先，結合了高效的對稱加密算法 AES 和強大的認證機制 GCM，提供了高速的加密和完整性保護。其次，由於 GCM 支援並行處理，能夠在多個平行處理器上進行加密和解密操作。此外，AES-GCM 也被廣泛應用於網絡通信和存儲設備中，以實現安全的數據傳輸和儲存。

為了說明 AES-GCM 演算法，以下將會解釋此演算法中常見的專有名詞，如表 2.5。

表 2.5 AES-GCM 演算法專有名詞解釋

專有名詞	解釋
GHASH 函數 (GHASH)	雜湊函數的一種，在 AES-GCM 中用於實現資料的認證碼生成與認證。 GHASH_K 表示以 K 為雜湊金鑰的 GHASH 函數。
加密函數 (CIPH)	AES-GCM 使用 AES 加密演算法作為加密函數。 CIPH_K 表示以 K 為加密金鑰的加密函數。
附加認證資料 (Associated data, AAD)	用於保護需要驗證但不需要加密的資訊，長度介於 0 到 $2^{64} - 1$ 個位元。
初始向量 (Initial Vector, IV)	可以以隨機方式生成，用來避免使用到相同金鑰加密的密文遭到破解，長度介於 0 到 $2^{64} - 1$ 個位元。
認證碼 (Tag)	用於驗證資料在傳輸或儲存中是否遭到竄改。長度介於 0~128 之間的任意整數。認證碼的長度記為 t 。
附加認證資料長度 ($\text{len}(A)$)	附加認證資料的長度，函數 $\text{len}()$ 會回傳一個 64 位元字串。
密文長度 ($\text{len}(C)$)	密文的長度，函數 $\text{len}()$ 會回傳一個 64 位元串。
最高有效位元 (Most Significant Bit, MSB)	MSB_t 表示只回傳最高有效位元（最左側）往下取 t 個位元。

2.6.1 AES-GCM 加密演算法

AES-GCM 加密流程如圖 2.30 所示，四個輸入分別為金鑰、初始化向量、明文以及附加認證資料，明文長度介於 0 到 $2^{39} - 256$ 個位元，附加認證資料長度則介於 0 到 $2^{64} - 1$ 個位元，兩者長度若是不足 128 的倍數，在演算法中會補足 0 以滿足條件；兩個輸出則分別為密文以及認證碼，密文的長度會與輸入的明文相同，認證碼是依據需求做選擇，通常為 128、120、112、104 或 96 位元。

假設有兩個正整數 n 與 u ，使得明文的總位數為 $(n - 1)128 + u$ ，其中 $1 \leq u \leq 128$ 。明文由 n 個位元串的序列組成，其中最後一個位元串的位長為 u ，而其他位元串的位長都是 128。將序列表示為 $P_1, P_2, \dots, P_{n-1}, P_n^*$ ，位元串稱為數據區塊，其中最

後一個位元串 P_n^* ，可能不是一個完整的數據區塊。同樣地，密文總位元數與明文相同，故也是由 n 個位元串組成，序列表示為 $C_1, C_2, \dots, C_{n-1}, C_n^*$ ，其中最後一個位元串 C_n^* 的位元數為 u 。另外，假設有兩個正整 m 與 v ，使得附加認證資料的總位元數為 $(m-1)128 + v$ ，其中 $1 \leq v \leq 128$ 。附加認證資料 AAD 則是由 m 個位元串的序列組成，表示為 $A_1, A_2, \dots, A_{m-1}, A_m^*$ ，其中最後一個位元串 A_m^* 是位元數為 v 。

以下定義 AES-GCM 加密操作：

$$(1) \ H = CIPH_K(0^{128})$$

$$(2) \ Y_0 = \begin{cases} IV \parallel 0^{31} \parallel 1 & , \text{len}(IV) = 96 \\ GHASH\left(H, (IV \parallel 0^{64+128[\text{len}(IV)/128]-\text{len}(IV)} \parallel \text{len}(IV)_{64})\right) & , \text{len}(IV) \neq 96 \end{cases}$$

$$(3) \ Y_i = \text{incr}(Y_{i-1}) \text{ for } i = 1, \dots, n$$

$$(4) \ J_0 = CIPH_K(Y_0)$$

$$(5) \ u = 128[\text{len}(C)/128] - \text{len}(C) \text{ and } v = 128[\text{len}(A)/128] - \text{len}(A)$$

$$(6) \ C_i = P_i \oplus GCTR_K(Y_i) \text{ for } i = 1, \dots, n-1$$

$$(7) \ C_i = P_n^* \oplus MSB_u(GCTR_K(Y_n))$$

$$(8) \ T = MSB_t(GHASH(H, A, C) \oplus J_0)$$

$$(9) \ \text{return}(C, T)$$

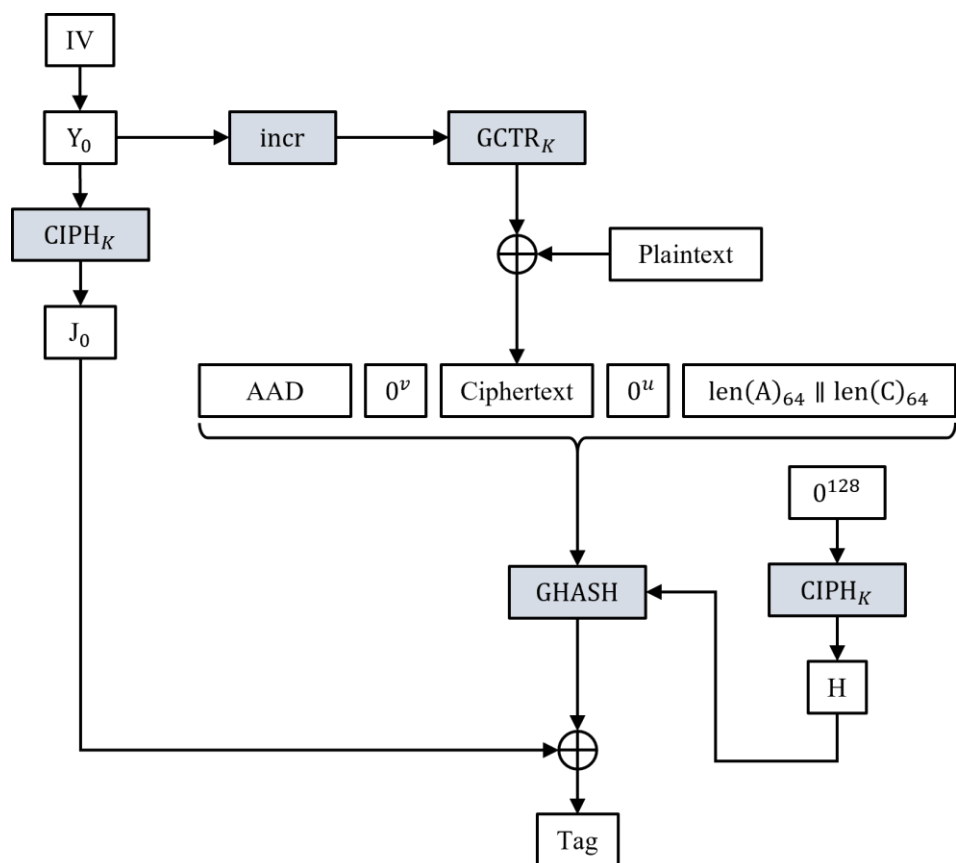


圖 2.30 AES-GCM 加密流程圖

2.6.2 AES-GCM 解密演算法

AES-GCM 解密流程如圖 2.31 所示，五個輸入分別為金鑰、初始化向量、明文、附加認證資料以及認證碼；只有一個輸出，即明文或是表示認證碼不合法的符號 FALL。

以下定義 AES-GCM 解密操作：

- (1) $H = CIPH_K(0^{128})$
- (2)
$$Y_0 = \begin{cases} IV \parallel 0^{31} \parallel 1 & , \text{len}(IV) = 96 \\ GHASH\left(H, (IV \parallel 0^{64+128[\text{len}(IV)/128]-\text{len}(IV)} \parallel \text{len}(IV)_{64})\right) & , \text{len}(IV) \neq 96 \end{cases}$$
- (3) $Y_i = \text{incr}(Y_{i-1}) \text{ for } i = 1, \dots, n$
- (4) $J_0 = CIPH_K(Y_0)$
- (5) $u = 128[\text{len}(C)/128] - \text{len}(C)$ and $v = 128[\text{len}(A) / 128] - \text{len}(A)$

- (6) $P_i = C_i \oplus GCTR_K(Y_i)$ for $i = 1, \dots, n - 1$
- (7) $P_i = C_n^* \oplus MSB_u(GCTR_K(Y_n))$
- (8) $T' = MSB_t(GHASH(H, A, C) \oplus J_0)$
- (9) $return \begin{cases} P & , if T == T' \\ FALL & , else \end{cases}$

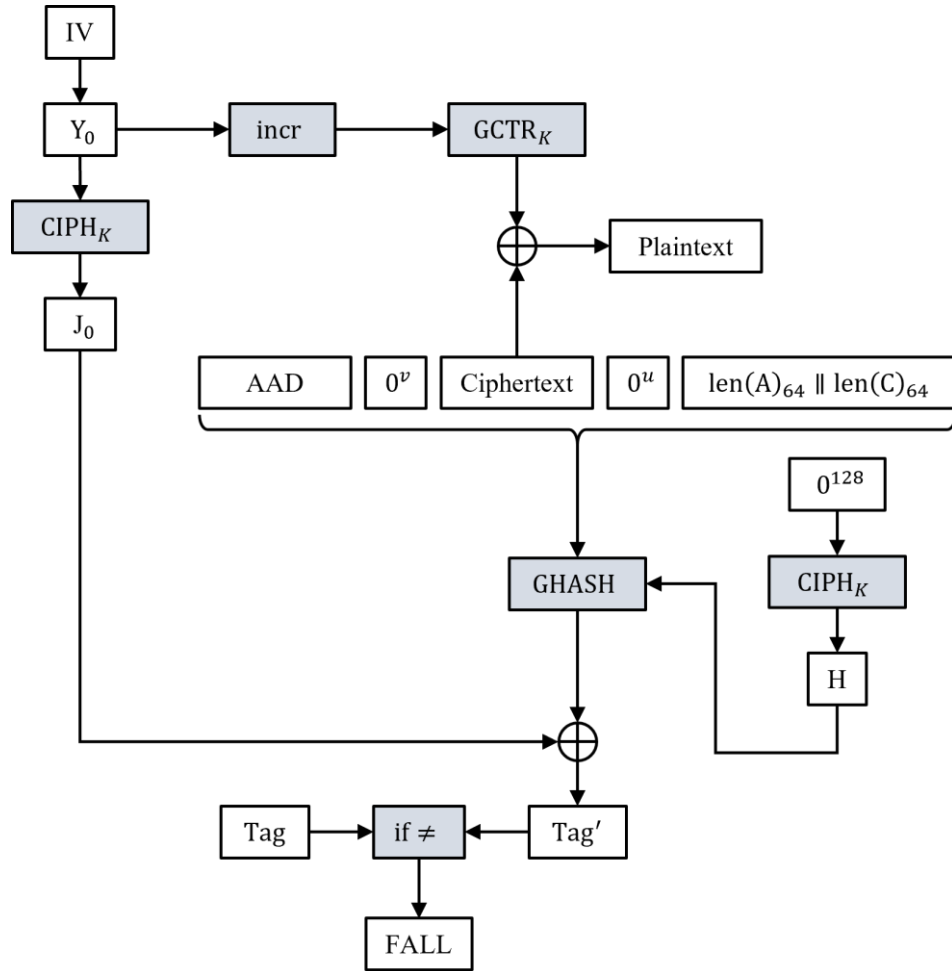


圖 2.31 AES-GCM 解密流程圖

2.6.3 GCTR 加密

GCTR (Galois/Counter Mode with Counter Blocks) 是一種對稱加密模式，在 AES-GCM 中負責對訊息進行加、解密。

如圖 2.32 所示，GCTR 加密演算法使用一個初始計數器區塊 (Initial Counter Block, ICB) 作為加密的起始點，並通過對 ICB 進行增量 (Increment, incr) 操作來生成下一個計數器值，使得每個數據區塊都有一個唯一的計數器值，確保了唯一性和隨機性。

以下定義 GCTR 加密演算法：

- (1) 如果輸入 X 是空字串，則輸出空字串 Y
- (2) $n = \lceil \text{len}(X) / 128 \rceil$
- (3) 令 $X = X_1 \parallel X_2 \parallel \dots \parallel X_{n-1} \parallel X_n^*$ ，其中 $1 \leq \text{len}(X_n^*) \leq 128$
- (4) 令 $CB_1 = \text{ICB}$
- (5) $CB_i = \text{incr}_{32}(CB_{i-1})$
- (6) $Y_i = X_i \oplus \text{CIPH}_K(CB_i)$
- (7) $Y_n^* = X_n^* \oplus \text{MSB}_{\text{len}(X_n^*)}(\text{CIPH}_K(CB_n))$
- (8) $Y = Y_1 \parallel Y_2 \parallel \dots \parallel Y_{n-1} \parallel Y_n^*$
- (9) $\text{return}(Y)$

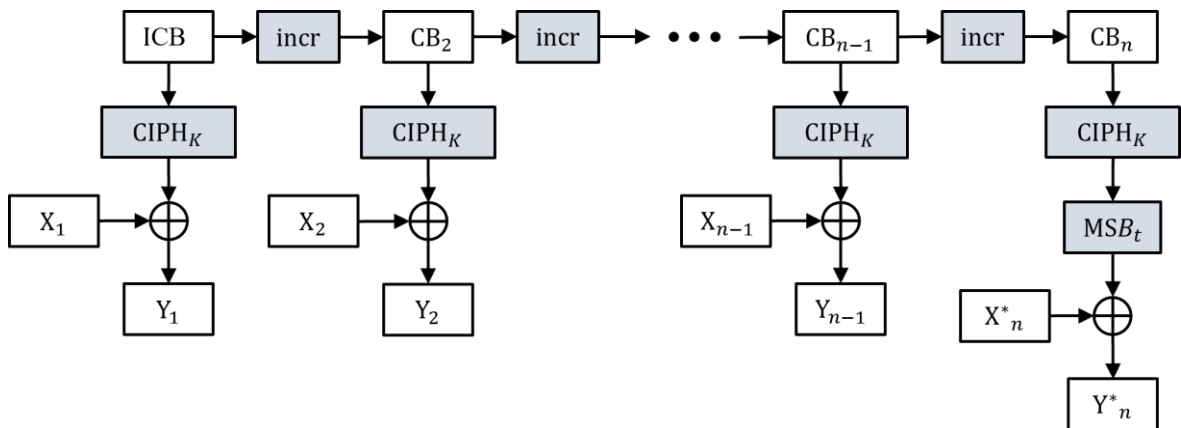


圖 2.32 GCTR 加密操作

2.6.4 GHASH 雜湊

GHASH (Galois Hash) 雜湊在 AES-GCM 演算法中負責對訊息進行認證。其計算效率和強大的防篡改特性成為 AES-GCM 演算法中重要的認證機制。GHASH 多項式與加密後的資料區塊進行乘法運算會計算出認證碼。如圖 2.33 所示。

GHASH 雜湊函數的定義如式(2-15)及式(2-16)所示，其中輸入參數 H 、 A 和 C ，分別代表雜湊金鑰 (Hash Key) 附加認證資料 (AAD) 及密文 (Ciphertext)。

$$GHASH(H, A, C) = X_{m+n+1} \quad (2-15)$$

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, \dots, m-1 \\ (X_{m-1} \oplus (A_m^* \parallel 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_i) \cdot H & \text{for } i = m+1, \dots, m+n-1 \\ (X_{m+n-1} \oplus (C_m^* \parallel 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\text{len}(A) \parallel \text{len}(C))) \cdot H & \text{for } i = m+n+1 \end{cases} \quad (2-16)$$

以下定義 GHASH 雜湊操作：

- (1) 令 $X = X_1 \parallel X_2 \parallel \dots \parallel X_{m-1} \parallel X_m$ ，表示位元長度為 $128m$ 的一段資料
- (2) 令 $Y_0 = 0^{128}$
- (3) $Y_i = (Y_{i-1} \oplus X_i) \cdot H \quad \text{for } i = 1, \dots, m$
- (4) $\text{return}(Y)$

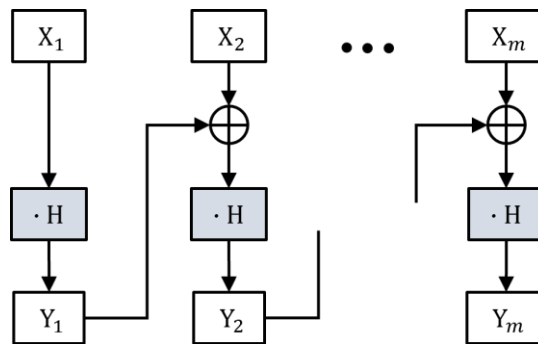


圖 2.33 GHASH 雜湊操作

第3章 設計與分析

為了實現低面積的 AES-GCM 電路設計，必須解決 AES 回合運算以及 GHASH 演算法中的乘法器複雜度高的問題。本章將針對如何簡化電路以及探討管線化處理對於整體電路的效能影響，分析出最適合的方式。

3.1 AES 回合運算單元

本節針對 AES 演算法中每個回合的步驟進行電路分析和最佳化。

3.1.1 位元組替代轉換模組

由 2.5.2 節可知，位元組替代轉換可以用兩種方式實現。第一種方式是查表法，將替代盒直接使用邏輯映射實現。第二種方式是直接使用邏輯運算實現，使用組合邏輯電路實現複合場運算[12][13]，可以得到替代盒的位元組替代轉換值。然而使用組合邏輯電路實現有限場 $GF(2^8)$ 的乘法反元素相當複雜，故為了降低電路的複雜度，可以使用複合場運算將有限場 $GF(2^8)$ 降幂成有限場 $GF(2^4)$ ，再求取乘法反元素。在加密過程中，需先求出乘法反元素，再進行仿射運算。以上兩個運算如下所示：

(1) 仿射運算

仿射運算的公式如式(3-1)所示，以矩陣乘法表示如式(3-2)所示。其中，輸入為一個位元組 b ，以位元表示為 $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$ ，輸出為 b' 。

$$\begin{cases} b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \\ c_i = \{63\}_{10} = \{01100011\}_2 \end{cases} \quad (3-1)$$

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (3-2)$$

(2) 乘法反元素

定義若 $a \cdot a^{-1} = 1$ ，則 a^{-1} 為 a 的乘法反元素。由於實現 AES 演算法中的位元組替代轉換中，需先求出在有限場 $GF(2^8)$ 的乘法反元素，但此運算電路相當複雜。因此根據文獻[22]，將有限場 $GF(2^8)$ 的數值降冪為有限場 $GF(2^4)$ 的方法為乘上 δ 轉換矩陣(Isomorphic Function)，如式(3-3)所示。接著在有限場 $GF(2^4)$ 求出乘法反元素，如式(3-4)所示，此運算較為簡單，故可以降低電路複雜度。算出有限場 $GF(2^4)$ 的乘法反元素後再乘上 δ^{-1} 反轉換矩陣 (Inverse Isomorphic Function)，如式(3-5)所示，轉回有限場 $GF(2^8)$ 。乘法反元素電路圖如圖 3.1 所示。其子電路如圖 3.2~圖 3.6 所示。

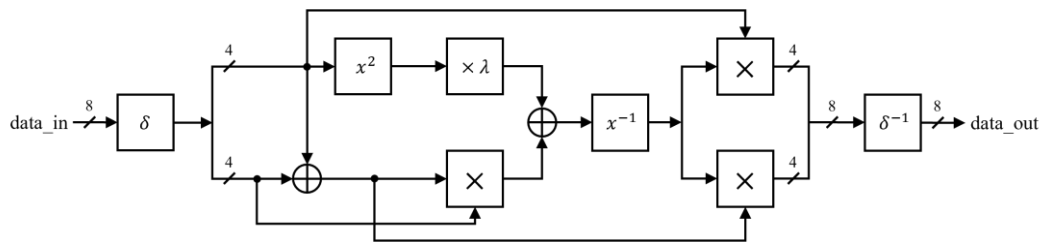


圖 3.1 乘法反元素架構圖[13]

$$\delta = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \quad (3-3)$$

$$\delta = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} b_6 \oplus b_1 \oplus b_0 \\ b_6 \oplus b_4 \oplus b_1 \\ b_7 \oplus b_4 \oplus b_3 \oplus b_2 \oplus b_1 \\ b_7 \oplus b_6 \oplus b_2 \oplus b_1 \\ b_7 \oplus b_5 \oplus b_3 \oplus b_2 \oplus b_1 \\ b_7 \oplus b_5 \oplus b_3 \oplus b_2 \\ b_7 \oplus b_6 \oplus b_4 \oplus b_3 \oplus b_2 \oplus b_1 \\ b_7 \oplus b_5 \end{bmatrix}$$

$$\begin{cases} x_0^{-1} = x_3 x_2 x_1 + x_3 x_2 x_0 + x_3 x_1 + x_3 x_1 x_0 + x_3 x_0 + x_2 + x_2 x_1 \\ \quad + x_2 x_1 x_0 + x_1 + x_0 \\ x_1^{-1} = x_3 + x_3 x_2 x_1 + x_3 x_1 x_0 + x_2 + x_2 x_0 + x_1 \\ x_2^{-1} = x_3 x_2 x_1 + x_3 x_2 x_0 + x_3 x_0 + x_2 + x_2 x_1 \\ x_3^{-1} = x_3 + x_3 x_2 x_1 + x_3 x_0 + x_2 \end{cases} \quad (3-4)$$

$$\delta^{-1} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \quad (3-5)$$

$$\delta^{-1} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} b_6 \oplus b_5 \oplus b_4 \oplus b_2 \oplus b_0 \\ b_5 \oplus b_4 \\ b_7 \oplus b_4 \oplus b_3 \oplus b_2 \oplus b_1 \\ b_5 \oplus b_4 \oplus b_3 \oplus b_2 \oplus b_1 \\ b_6 \oplus b_5 \oplus b_4 \oplus b_2 \oplus b_1 \\ b_6 \oplus b_5 \oplus b_1 \\ b_6 \oplus b_3 \\ b_7 \oplus b_6 \oplus b_5 \oplus b_1 \end{bmatrix}$$

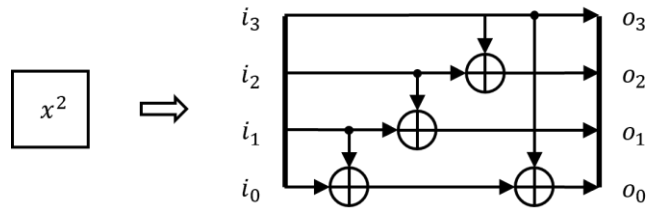


圖 3.2 乘法反元素內部子電路， x^2

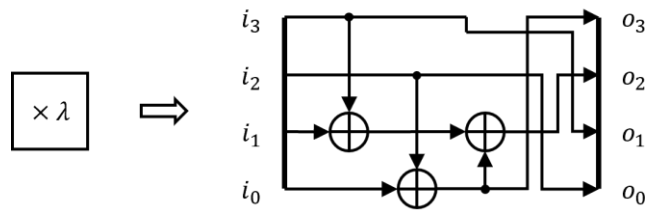


圖 3.3 乘法反元素內部子電路， $\times \lambda$

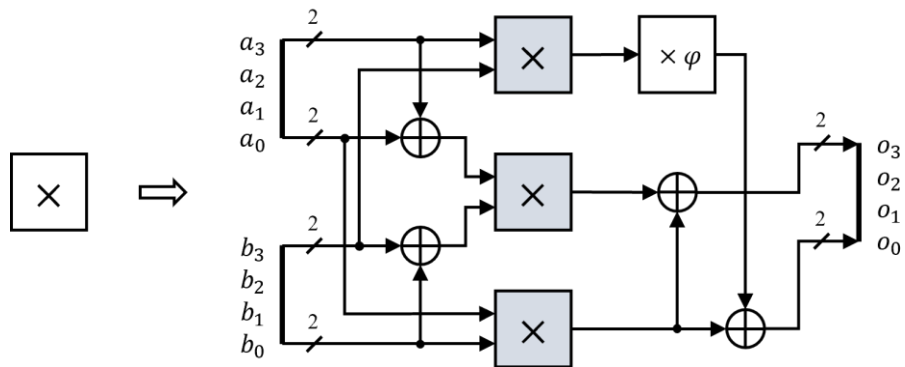


圖 3.4 乘法反元素內部子電路， $GF(2^4)$ 乘法器

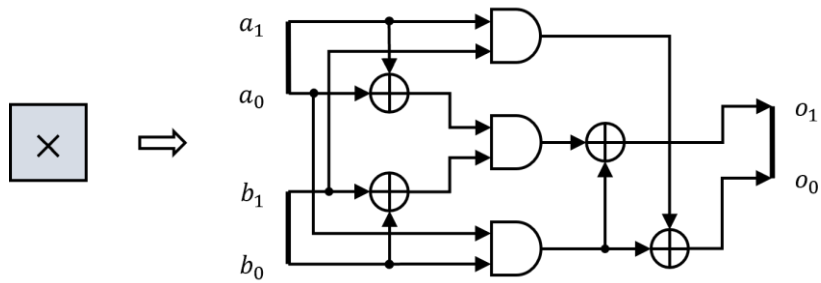


圖 3.5 乘法反元素內部子電路， $GF(2^8)$ 乘法器

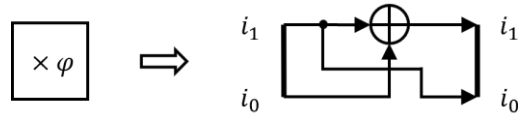


圖 3.6 乘法反元素內部子電路， $\times \varphi$

δ^{-1} 反轉換矩陣與仿射運算皆為矩陣乘法運算，因此可以透過矩陣乘法的結合律簡化電路，簡化後的結果如式(3-6)所示。組合乘法反元素電路和仿射運算電路組成的位元組替代轉換電路，如圖 3.7 位元組替代轉換電路架構所示。

$$\delta^{-1} \times affine = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (3-6)$$

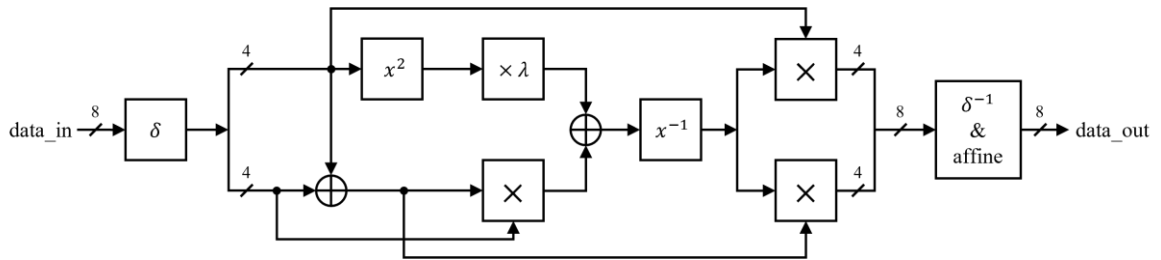


圖 3.7 位元組替代轉換電路架構[13]

表 3.1 為使用 Xilinx Virtex 7 系列的 XC7VX330T，使用兩種方法實現位元組替代轉換模組的合成結果。由表可以得知，在 FPGA 設計上使用直接邏輯映射的方法，會使用更少的 LUTs 數量，節省了約 32.9%。因此在 FPGA 上採取直接邏輯映射的方式實現位元組替代轉換的模組。

表 3.1 位元組替代轉換模組之 FPGA 合成結果比較

實現方法	直接邏輯映射	複合場運算
資源消耗	65 LUTs	97 LUTs
Delay Time	2.288 ns	2.995 ns

表 3.2 為使用台灣半導體研究中心 (Taiwan Semiconductor Research Institute, TSRI) 提供的 tsmc 0.18 μm 標準元件庫，使用兩種方法實現位元組替代轉換模組的合成結果。由表可以得知，在 ASIC 設計上使用複合場運算的方法，會使用更少的等效邏輯閘數量，節省了約 55.6%。因此在 ASIC 上採取複合場運算的方式實現位元組替代轉換的模組。

表 3.2 位元組替代轉換模組之 ASIC 合成結果比較

實現方法	直接邏輯映射	複合場運算
Gate Count	1955.9	868.1
Delay Time	2.60 ns	4.64 ns
Power	3.1345 mW	3.0480 mW

3.1.2 列位移轉換模組

列位移轉換為將狀態矩陣中的位元組進行左旋的轉換，其中除了第一列外，其餘三列會根據不同的位移長度進行左旋，在 2.5.2 節有詳細的演算法介紹。硬體電路實現上，可以使用硬體連線的方式，如圖 3.8 所示。

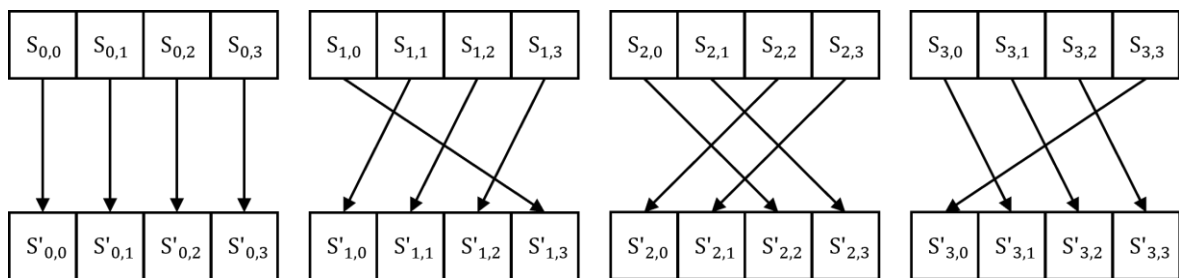


圖 3.8 列位移轉換電路架構

3.1.3 混合行轉換模組

混合行轉換為多項式的矩陣乘法運算，狀態矩陣中的每一行是一個在限場 $GF(2^8)$ 的三次多項式，將每一行乘上一個固定多項式 $a(x) = \{03\}x^3 + \{03\}x^2 + \{01\}x + \{02\}$ ，若發生溢位再將其模於 $x^4 + 1$ ，相乘結果如式(3-7)所示。演算法於 2.5.2 節有詳細的介紹。

$$\begin{cases} S'_{0,c} = (\{02\} \cdot S_{0,c}) \oplus (\{03\} \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} = S_{0,c} \oplus (\{02\} \cdot S_{1,c}) \oplus (\{03\} \cdot S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \cdot S_{2,c}) \oplus (\{03\} \cdot S_{3,c}) \\ S'_{3,c} = (\{03\} \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \cdot S_{3,c}) \end{cases} \quad (3-7)$$

為了簡化混合行轉換的電路，使用變數代換的原理，找到一個能讓四個方程式共用的因數。經過轉換後的方程式如式(3-8)所示。硬體實現上，使用 *xtime* 電路取代乘法運算，其演算法定義為將輸入左旋一個位元，若發生溢位再將其與 $\{1b\}$ 做互斥或運算，電路如圖 3.9 所示。根據化簡後的方程式所設計出的混合行轉換電路架構如圖 3.10 所示。

$$\begin{cases} S'_{0,c} = \{02\} \cdot (S_{0,c} \oplus S_{1,c}) \oplus \{01\} \cdot (S_{1,c} \oplus (S_{2,c} \oplus S_{3,c})) \\ S'_{1,c} = \{02\} \cdot (S_{1,c} \oplus S_{2,c}) \oplus \{01\} \cdot (S_{2,c} \oplus (S_{0,c} \oplus S_{3,c})) \\ S'_{2,c} = \{02\} \cdot (S_{2,c} \oplus S_{3,c}) \oplus \{01\} \cdot (S_{3,c} \oplus (S_{0,c} \oplus S_{1,c})) \\ S'_{3,c} = \{02\} \cdot (S_{0,c} \oplus S_{3,c}) \oplus \{01\} \cdot (S_{0,c} \oplus (S_{1,c} \oplus S_{2,c})) \end{cases} \quad (3-8)$$

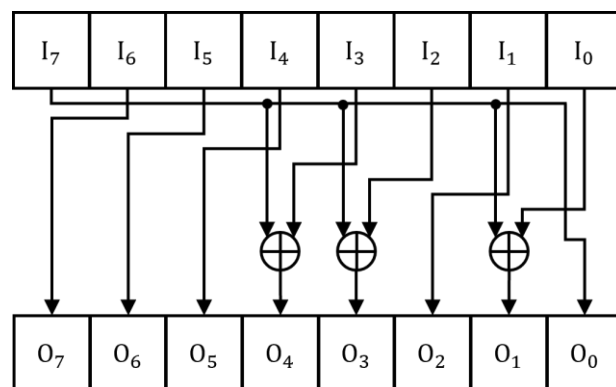


圖 3.9 *xtime* 電路架構

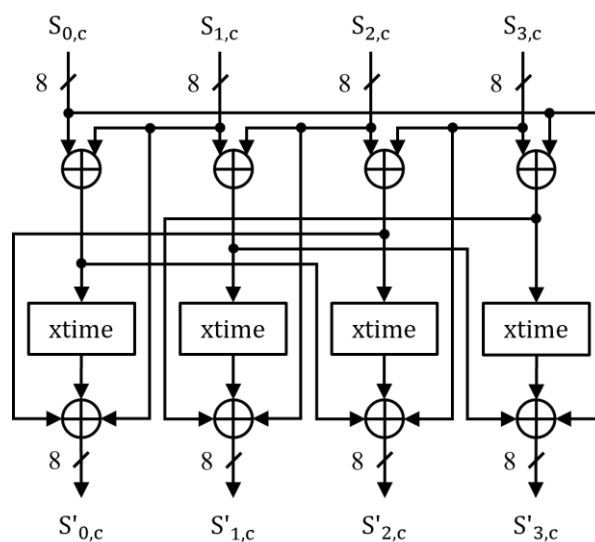


圖 3.10 混合行轉換電路架構

3.1.4 加入回合金鑰模組

在 AES-128 演算法中，加入回合金鑰是將狀態矩陣與回合金鑰矩陣進行互斥或運算。在硬體實現上，需要使用 128 個互斥或邏輯閘，如圖 3.11 所示。

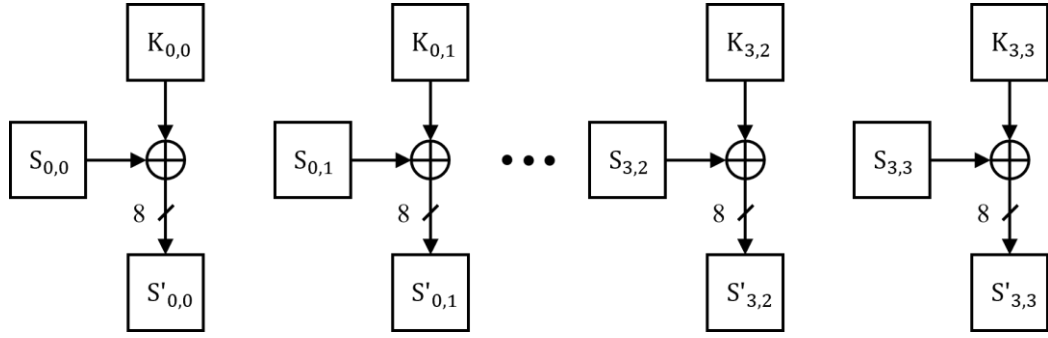


圖 3.11 加入回合金鑰電路架構

3.2 金鑰擴展單元

金鑰擴展單元的功能是利用初始金鑰生成每一回合所需的回合金鑰，如何設計才能配合回合單元運算並且符合本論文的需求，則需要妥善評估。

3.2.1 金鑰擴展單元設計方法分析

兩種設計金鑰擴展單元的方法，一種為即時運算設計，另一種為預先運算設計。在設計中需要仔細評估即時運算設計和預先運算設計之間的優缺點，並根據特定應用的需求、平台資源和性能要求做出適當的選擇。

3.2.1.1 即時運算設計

即時運算設計（On-the-Fly Computation）是指在加密運算過程中，即時生成每一回合所需的回合金鑰。在這種方法中，金鑰擴展單元使用上一回合的回合金鑰來即時生成當前回合所需的金鑰，因此金鑰擴展程序單元只需要保存一組回合金鑰。這意味著在設計中只需要使用一個大小為 Nk 的暫存器來儲存金鑰。以下是對即時運算設計方法進行的優缺點分析。

優點：

- (1) 節省儲存空間：

因為只需要用一個大小為 Nk 的暫存器來儲存當前回合所需的金鑰，故不需要大量的記憶體或暫存器來儲存所有回合的金鑰，因此有效地減少了所需的記憶體容量或暫存器數量，使得晶片面積降低。

(2) 降低硬體複雜度：

即時運算設計中不需要實現完整的金鑰擴展單元，降低了硬體電路的複雜度。

(3) 提高效率：

即時生成回合金鑰可以減少回合之間的等待時間，提高加密運算的效率。特別是在大規模的資料加密中，減少尤為明顯。

缺點：

(1) 複雜度高：

由於即時運算需要在每個回合進行金鑰生成，對硬體系統的運算速度有一定的要求，否則可能會導致加密過程中引入一定的時間延遲。因此，在進行即時運算設計時，需要對硬體系統的運算速度進行評估，並考慮時間延遲的因素，確保能夠在合理的時間內完成金鑰擴展操作。

3.2.1.2 預先運算設計

預先運算設計 (Pre-Computation) 是指在加密運算之前預先生成每個回合所需的回合金鑰。由於所有回合金鑰已經預先計算，並連同初始金鑰一起儲存在記憶體或暫存器中，所以當進行加密時，只需要將當前要使用的回合金鑰直接從對應的記憶體或暫存器中讀取。以下是對預先運算設計方法進行的優缺點分析。

優點：

(1) 提高加密速度：

因為回合金鑰已經預先生成並存儲，可以直接從記憶體或暫存器中讀取，節省了金鑰生成的時間，提高了加密速度。

(2) 降低延遲：

預先生成回合金鑰可以避免在每個回合進行金鑰生成的延遲，使加密運算可以更快地進行。

缺點：

(1) 硬體資源使用高：

由於需要將初始金鑰以及所需的所有回合數儲存在記憶體或暫存器中，故需要較大的硬體資源，會提升晶片面積。

(2) 安全性問題：

由於回合金鑰事先生成並儲存，可能會增加了保護和管理金鑰的難度，所以需要特別注意安全性。

3.2.2 金鑰擴展單元設計

金鑰擴展單元演算法已於 2.5.1 節詳細介紹過。相較於預先運算設計，即時運算設計節省了硬體資源的使用，對於嵌入式系統或有限資源環境下的設計尤其有利，除了節省了硬體資源並提高了整體效能，也更有效地管理金鑰。雖然即時運算設計需要考慮硬體運算速度，且可能會造成時間延遲，但只要在設計的時候注意此問題，即可避免可能會出現的問題。並且減少了所需的邏輯閘數量和電路元件數量，從而降低了設計和實現的難度。由於本論文的目的是應用於 IoT，硬體資源有限，因此採取只需要大小為 Nk 的暫存器為回合金鑰儲存單元的即時運算設計方式。

3.3 GHASH 運算單元

GHASH 運算單元是 AES-GCM 演算法中用於實現認證的關鍵部分。使用有限域乘法運算和位元組替代轉換來計算一個 128 位的雜湊值。GHASH 運算使用了多個步驟，包括對明文和附加認證資料進行位元組替代轉換，將結果與金鑰進行互斥或運算，然後使用有限域乘法對結果進行連續的運算。最終得到的雜湊值用於驗證數據的完整

性和真實性。GHASH 雜湊的計算效率和強大的防篡改特性使其成為 AES-GCM 中重要的認證機制。

GHASH 運算單元內部使用的 $GF(2^{128})$ 乘法器和 $GF(2^{128})$ 加法器是核心運算單元， $GF(2^{128})$ 乘法器的實現方式對於硬體資源的使用有很大的影響。傳統乘法器實現有限場乘法的方式，如圖 3.12 所示。硬體實現大位元數的有限場乘法，會導致邏輯閘的數量隨著位元數的增加呈平方增長[14]，其使用量會是 $O(n^2)$ 的量級，這將導致硬體資源的大量消耗且實現方式會變得更加困難。

因此需要使用更有效的方法來實現 $GF(2^{128})$ 的乘法運算，以減少資源消耗並提高運算效率。在 GHASH 運算單元中，可以採用 Karatsuba 有限場乘法器，可以在保證運算正確性的同時，提高效能並節省硬體資源。

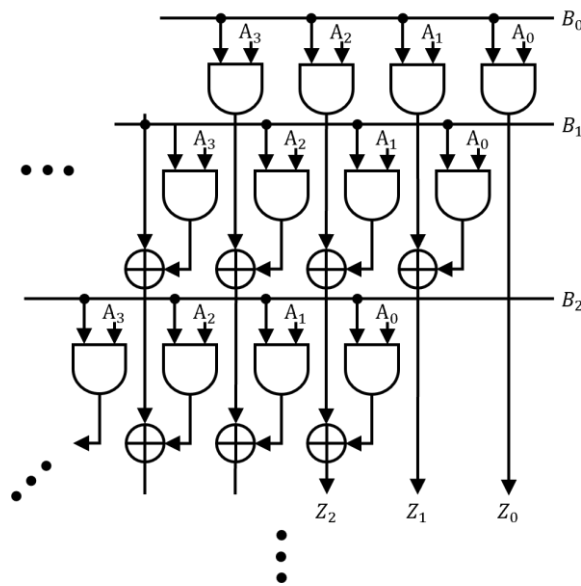


圖 3.12 傳統有限場乘法器

3.3.1 Karatsuba 演算法

Karatsuba 演算法[15]是一種快速乘法演算法，主要用於計算兩個大數的乘積。基於分治法，將兩個大數拆分成位元數相等但較小的數字，並使用遞迴的方法進行乘法

運算，最後結合所有乘積得到最終結果。Karatsuba 演算法的虛擬碼如圖 3.13 所示。

```
Karatsuba (num1 , num2)
begin
    if (num1 < 10 or num2 < 10)
        return num1 * num2
    m = max(size_base_10 (num1) , size_base_10 (num2) )
    m2 = m / 2
    high1 , low1 = split_at (num1 , m2)
    high2 , low2 = split_at (num2 , m2)
    z0 = Karatsuba (low1 , low2)
    z1 = Karatsuba ( (low1 + high1) , (low2 + high2) )
    z2 = Karatsuba (high1 , high2)
    return (z2 * 10 ^ (2 * m2) ) + ( (z1 - z2 - z0) * 10 ^ m2) + z0
end
```

圖 3.13 Karatsuba 演算法虛擬碼

3.3.2 Karatsuba $GF(2^{128})$ 乘法器

Karatsuba 快速乘法演算法的硬體實現，如圖 3.14 所示。該實現方式利用遞迴的方法，將兩個大數切分成較小的部分，並使用三次乘法運算和少量的加法和移位操作來完成乘法運算。這種方法能夠大幅降低乘法運算的複雜度，同時節省了晶片資源的使用。相較於傳統乘法器，Karatsuba 演算法有效地減少乘法操作的次數，降低了運算的時間複雜度。對於兩個 n 位元的數字而言，傳統乘法的时间複雜度為 $O(n^2)$ ，而 Karatsuba 演算法的时间複雜度則為 $O(3n^{\log_2 3}) \sim O(3n^{1.585})$ ，若 n 為 2 的次方，則其時間複雜度剛好等於 $O(3n^{\log_2 3})$ [16][16]。

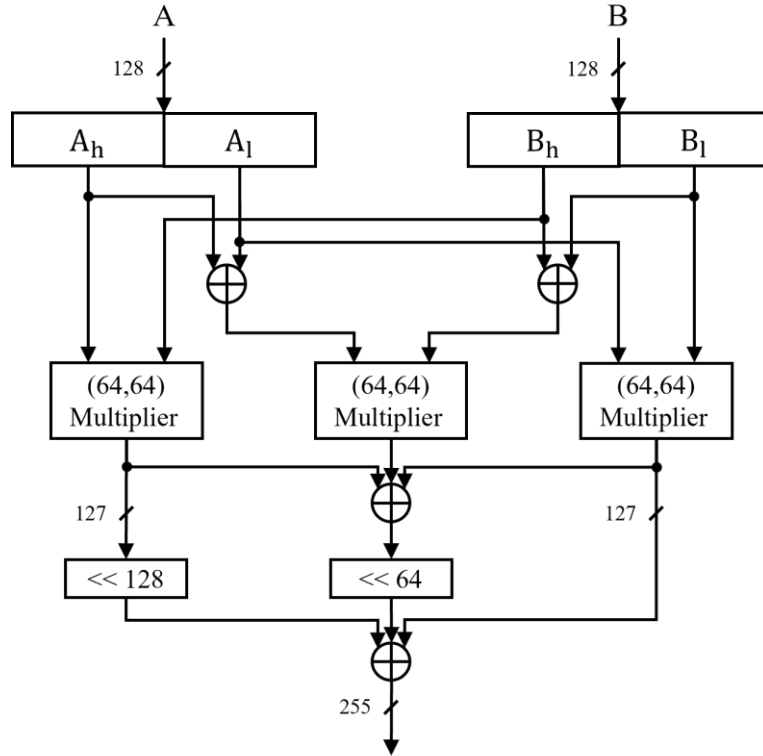


圖 3.14 Karatsuba 有限場乘法器

Karatsuba 快速乘法演算法採取遞迴的方法，不同的遞迴數對應的空間複雜度也不同。將兩個 n 位元的乘法元素 A 及 B 切割成 $\frac{n}{2}$ 位元，分別為 A_h 、 A_l 及 B_h 、 B_l ；一開始先將 A_h 、 A_l 及 B_h 、 B_l 分別做互斥或運算，結果表示為 A_{h+l} 及 B_{h+l} ，需要使用二元域中的兩個 $\frac{n}{2}$ 位元加法器，接著將 A_h 、 B_h 與 A_{h+l} 、 B_{h+l} 以及 A_l 、 B_l 分別做乘法運算，結果表示為 AB_h 、 AB_{h+l} 及 AB_l ，需要使用二元域中的三個 $\left(\frac{n}{2} \times \frac{n}{2}\right)$ 位元乘法器，再將 n 位元的 AB_h 、 AB_{h+l} 及 AB_l 做互斥或運算，結果表示為 AB_{h+l} ，需要使用二元域中的兩個 n 位元加法器，最後將左移 128 位元的 AB_h 、左移 64 位元的 AB_{h+l} 及 AB_l 做互斥或運算，需要使用二元域中的兩個 $\frac{n}{2}$ 位元加法器。故一個遞迴需使用三個 $\left(\frac{n}{2} \times \frac{n}{2}\right)$ 位元乘法器以及四個 n 位元加法器，1 至 4 階的公式推倒如式(3-9)~(3-12)所示。

$$3 \times \left(\frac{n}{2} \times \frac{n}{2}\right) + 4n \quad (3-9)$$

$$3 \times \left(3 \times \left(\frac{n}{4} \times \frac{n}{4}\right) + 4 \times \frac{n}{2}\right) + 4n \quad (3-10)$$

$$3 \times \left(3 \times \left(3 \times \left(\frac{n}{8} \times \frac{n}{8}\right) + 4 \times \frac{n}{4}\right) + 4 \times \frac{n}{2}\right) + 4n \quad (3-11)$$

$$3 \times \left(3 \times \left(3 \times \left(3 \times \left(\frac{n}{16} \times \frac{n}{16}\right) + 4 \times \frac{n}{8}\right) + 4 \times \frac{n}{4}\right) + 4 \times \frac{n}{2}\right) + 4n \quad (3-12)$$

根據上述公式，當 $n = 128$ 時，傳統有限場與 Karatsuba 有限場乘法器 1 階至 7 階的空間複雜度結果如表 3.3 所示。

表 3.3 有限場乘法器之空間複雜度比較

乘法器種類	空間複雜度結果
傳統乘法器	16384
Karatsuba 乘法器 1 階遞迴	12800
Karatsuba 乘法器 2 階遞迴	10496
Karatsuba 乘法器 3 階遞迴	9344
Karatsuba 乘法器 4 階遞迴	9344
Karatsuba 乘法器 5 階遞迴	10640
Karatsuba 乘法器 6 階遞迴	13556
Karatsuba 乘法器 7 階遞迴	18659

根據文獻[22]使用 Xilinx Virtex 5 系列的 XC5VLX220，合成傳統有限場與 1 階至 7 階的 Karatsuba 有限場乘法器，比較的資源消耗量結果如表 3.5 所示。其資源消耗與表 3.3 的空間複雜度結果相同，4 階的 Karatsuba 有限場乘法器空間複雜度低且使用的 LUT 資源最少。與傳統乘法器相比節省了約 50.7% 的 LUT 使用量，大幅降低了硬

體資源的使用。表 3.4 為使用 Xilinx Virtex 7 系列的 XC7VX330T，合成 Karatsuba 有限場乘法器三階及四階遞迴結果，其結果與預期相符。因此本論文的设计選擇使用四階遞迴的 Karatsuba 乘法器做為 $GF(2^{128})$ 乘法器的核心運算。

表 3.4 Karatsuba 乘法器資源消耗比較

實現方法	Karatsuba 乘法器 3 階遞迴	Karatsuba 乘法器 4 階遞迴
資源消耗	4330 LUTs	4228 LUTs
Delay Time	4.473 ns	4.629 ns

表 3.5 有限場乘法器之資源消耗比較[22]

乘法器種類	資源消耗
傳統乘法器	8179 LUTs
Karatsuba 乘法器 1 階遞迴	6330 LUTs
Karatsuba 乘法器 2 階遞迴	5260 LUTs
Karatsuba 乘法器 3 階遞迴	4469 LUTs
Karatsuba 乘法器 4 階遞迴	4033 LUTs
Karatsuba 乘法器 5 階遞迴	4686 LUTs
Karatsuba 乘法器 6 階遞迴	5551 LUTs
Karatsuba 乘法器 7 階遞迴	5684 LUTs

由於有限場乘法器的輸出為 255 位元，如圖 3.14 所示，超過了有限場 $GF(2^{128})$ 所能表示的範圍，因此需要利用二元場化簡函數（Binary Field Reduction）將乘法器的輸出結果化簡為 128 位元，使結果能保持在有限場 $GF(2^{128})$ 中。此化簡方法為將 255 位元的二進制數值視為一個一元 254 次多項式，將該多項式對應於有限場 $GF(2^{128})$ 的本原多項式 $p(x) = x^{128} + x^7 + x^2 + x + 1$ ，並進行模除運算。最終得到的餘式即為 $GF(2^{128})$ 乘法器的輸出結果，其位元寬度為 128 位元。如圖 3.15 所示。二元場化簡函數能夠確保結果在 $GF(2^{128})$ 有限場中，使得乘法運算結果正確且符合

所需的位元寬度。此化簡函數不但可以降低硬體資源的使用，還可以提高運算效率。

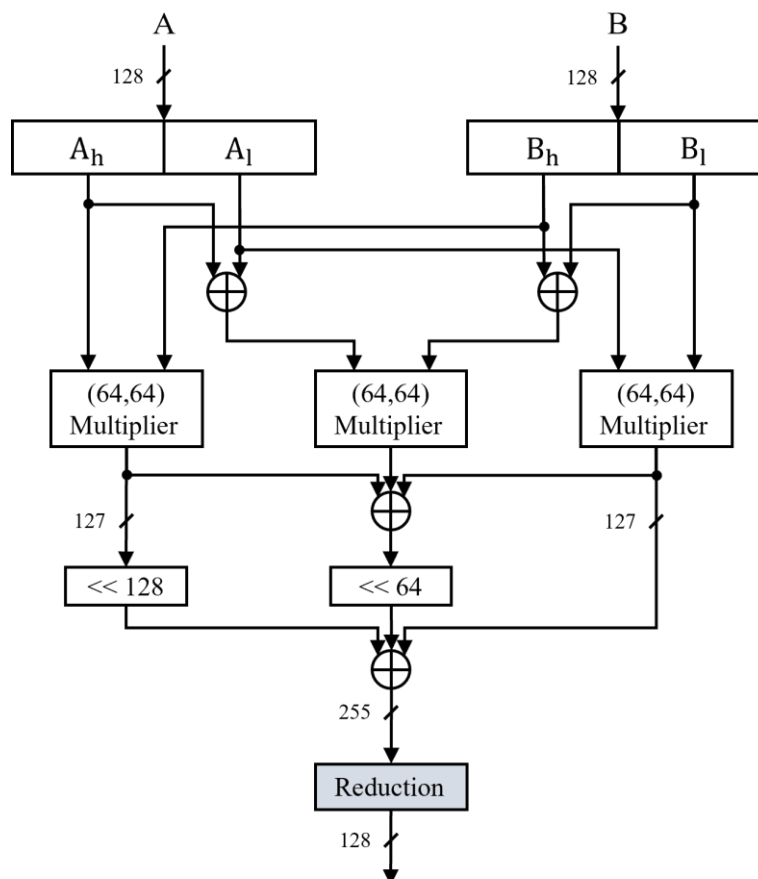


圖 3.15 Karatsuba GF(2¹²⁸)乘法器

在二元場化簡函數的電路實現上，使用了邏輯預先化簡的方法，以避免過長的組合邏輯關鍵路徑延遲。由於減法在有限場中的運算與加法相同，都可以表示為互斥或運算，因此可以利用同餘的觀念和互斥或的運算來設計更簡潔的電路。根據式(3-13)的同餘方程式，可以將大於 127 次方的項次進行模除，並將結果表示為本原多項式 $g(x)$ 的互斥或結果。例如，項次 x^{128} 可以轉換為 $x^7 + x^2 + x + 1$ ，因此當輸入的第 128 位元為 1 時，輸出的第 7 位元、第 2 位元、第 1 位元 and 第 0 位元的結果將與其他影響這些位元數值的項次進行互斥或運算。透過此種方法的邏輯預先化簡，電路中最長的組合邏輯輸出延遲時間為 3 個互斥或邏輯閘的延遲時間總和，能夠有效地降低電路的延遲，提高整體的運算效率，同時也能節省硬體資源，使得二元場化簡函數的電

路更加精簡和高效。

$$p(x) = x^{128} + x^7 + x^2 + x + 1$$

$$x^{128} \equiv x^7 + x^2 + x + 1 \mod p(x)$$

$$x^{129} \equiv x(x^7 + x^2 + x + 1) \mod p(x) = x^8 + x^3 + x^2 + x \quad (3-13)$$

...

$$x^{254} \equiv x^{127} + x^{126} + x^{12} + x^6 + x^5 + x^2 + x + 1 \mod p(x)$$

第4章 AES-GCM 硬體架構

本章根據第三章中對各個運算單元的設計與分析，設計一個低面積的 AES-GCM 矽智財架構。將依序介紹每個運算模組的內部架構，包括 AES 運算模組和 GHASH 運算模組。

4.1 AES-GCM 矽智財架構

本論文實現的 AES-GCM 矽智財方塊圖如圖 4.1 所示。為了可以應用於嵌入式系統，輸入方面，使用一條 8 位元的資料輸入通道，其中金鑰、附加認證資料、明文（加密時）、密文（解密時）、認證碼（解密時）皆共用相同的輸入通道；並且利用 in_type 訊號線指示目前輸入資料的類別。本設計事先並不會知道資料的長度，因此會有一個 last 信號，用於指示是否為最後一筆輸入資料。輸出方面，同樣使用了一條 8 位元的資料輸出通道，所有的輸出都共用相同的輸出通道。各個腳位詳細功能定義請參考表 4.1 AES-GCM 矽智財腳位功能定義。

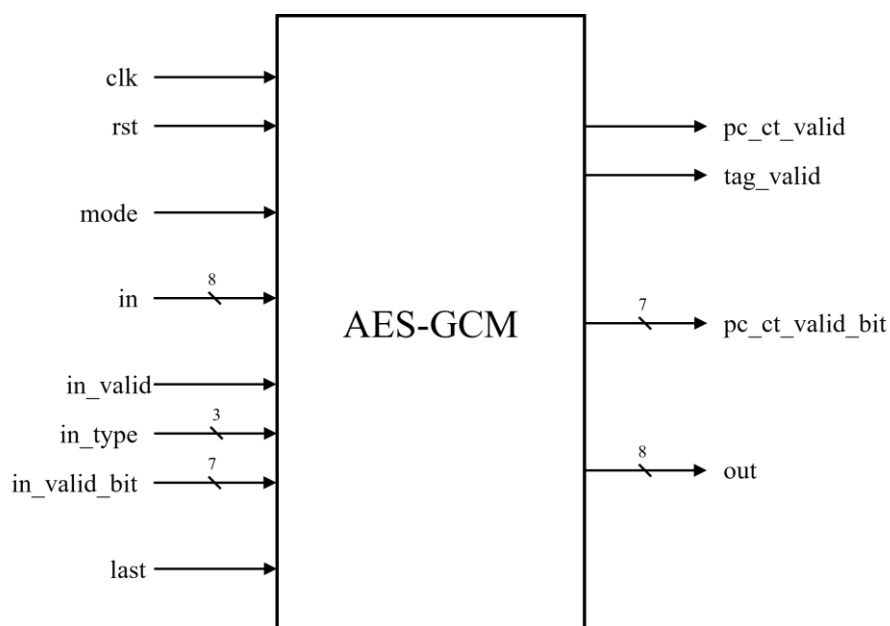


圖 4.1 AES-GCM 矽智財方塊圖

表 4.1 AES-GCM 矽智財腳位功能定義

腳位名稱	I/O	位元寬度	功能
clk	I	1	系統時脈
rst	I	1	系統同步重置信號
mode	I	1	模式選擇信號。邏輯 0 表示認證加密，邏輯 1 表示認證解密
in	I	8	初始向量、金鑰、附加認證資料、明文（加密時）、密文（解密時）輸入通道
in_valid	I	1	輸入的有效信號
in_type	I	3	輸入的資料類型，0 表初始向量，1 表示金鑰，2 表示附加認證資料，3 表示明文，4 表示密文，5 表示認證碼（解密時），6 表示無效
in_valid_bit	I	7	in 的有效位元數
last	I	1	最後一筆輸入的指示信號
pc_ct_valid	O	1	輸出密文（加密時）或明文（解密時）的有效信號
tag_valid	O	1	認證碼的有效信號（加密時）或資料是否合法的指示信號（解密時）
pc_ct_valid_bit	O	7	輸出密文（加密時）或明文（解密時）的有效位元數
out	O	8	密文（加密時）、明文（解密時）、認證碼的輸出通道

AES-GCM 矽智財內部架構如圖 4.2 所示。整體架構包含初始向量編碼模組、CTR 模式計數器模組、AES 運算模組、GHASH 運算模組、比較模組與控制單元模組。

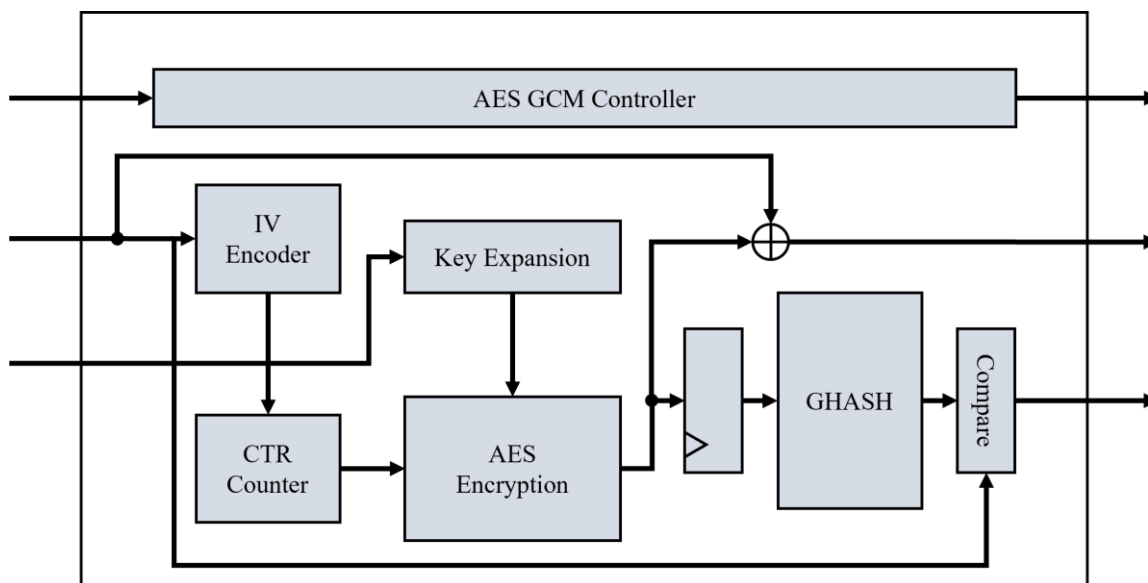


圖 4.2 AES-GCM 設計之內部架構圖

4.2 AES 運算模組

AES 運算模組的各個腳位詳細功能定義請參考表 4.2。模組內包括 AES-CTR 運算單元與即時金鑰擴展單元，如圖 4.3 所示。不斷遞增的計數器輸入 AES 運算單元，經過 13 個時脈後輸出，將加密資料與明文（或密文）進行互斥或運算即可得到密文（或明文）。

表 4.2 AES 運算模組腳位功能定義

腳位名稱	I/O	位元寬度	功能
clk	I	1	系統時脈
rst	I	1	系統同步重置信號
in_valid	I	1	輸入的有效信號
in	I	128	資料需加密或解密的輸入通道
key	I	128	金鑰的輸入通道
out_valid	O	1	輸出的有效信號
out	O	128	資料經由加密或解密的輸出通道

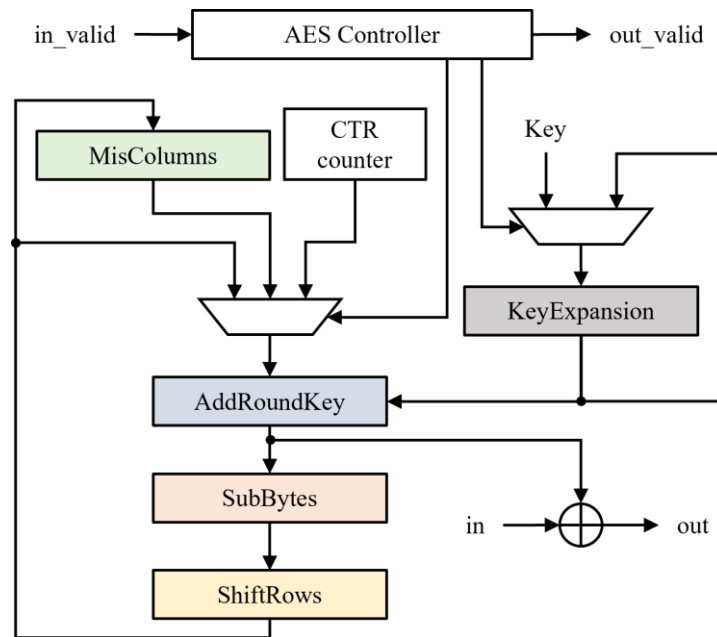


圖 4.3 AES 運算模組電路架構圖

4.3 GHASH 運算模組

GHASH 運算模組的各個腳位詳細功能定義請參考表 4.3。

表 4.3 GHASH 運算模組腳位功能定義

腳位名稱	I/O	位元寬度	功能
clk	I	1	系統時脈
rst	I	1	系統同步重置信號
hash_in	I	128	雜湊金鑰的輸入通道
in	I	128	欲認證資料的輸入通道
h_en	I	1	H 暫存器的控制信號
x_en	I	1	$GF(2^{128})$ 乘法器輸出暫存器的控制信號
gf_mul_sel1	I	1	管線暫存器的輸入選擇信號
ghash_out	O	128	認證碼的輸出通道

GHASH 運算模組的電路架構圖如圖 4.4 所示。如圖 4.5 所示，基於 3.3.2 節的 Karatsuba $GF(2^{128})$ 乘法器電路架構新增雜湊金鑰的外部輸入路徑，用於接收雜湊金

鑰，並且在 $GF(2^{128})$ 乘法器前加入兩個管線化暫存器，在不影響 GHASH 運算單元運算原理以及硬體資源的前提下縮短關鍵路徑的延遲。

在 GHASH 雜湊運算的過程中，首先將雜湊金鑰 H 通過 h_en 腳位輸入 GHASH 運算模組並存入 H 暫存器。當前置運算準備好後，GHASH 運算模組即可以接受欲認證資料的輸入。通過一個 128 位元的輸入腳位，進行 GHASH 雜湊運算，最終將雜湊結果經由 128 位元的輸出腳位輸出。

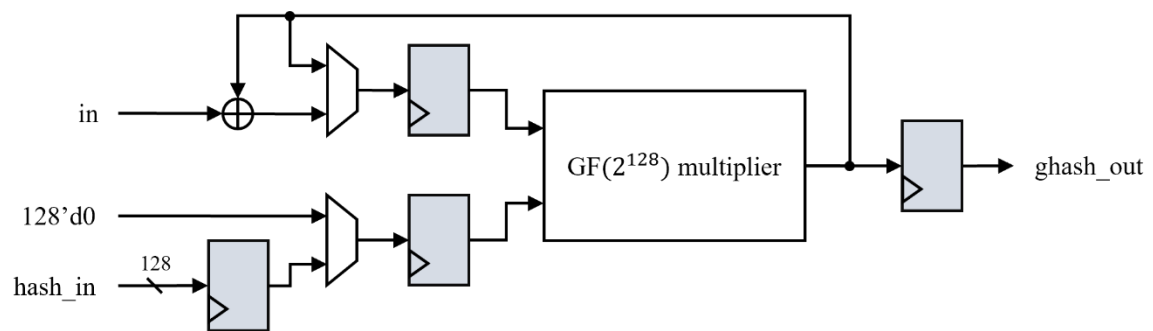


圖 4.4 GHASH 運算模組電路架構

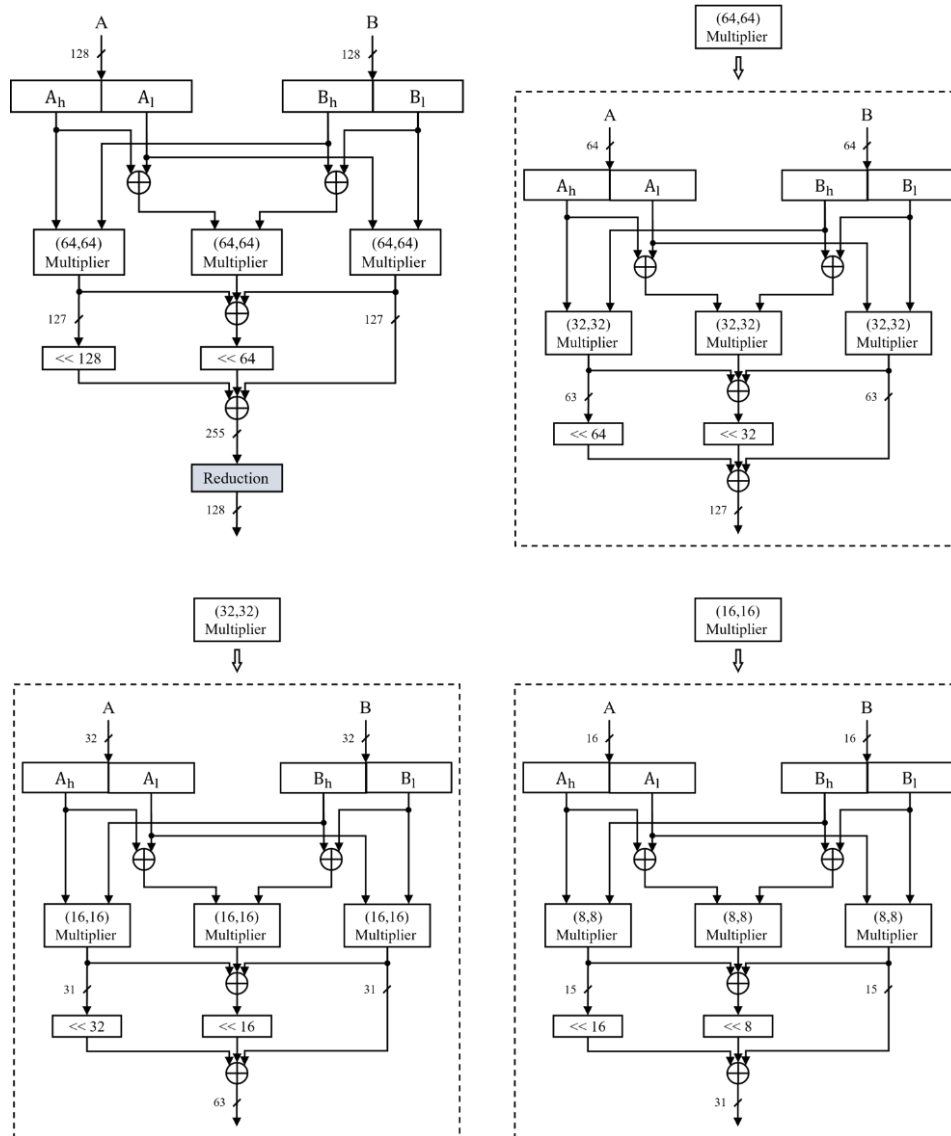


圖 4.5 Karatsuba $GF(2^{128})$ 乘法器電路架構[22]

4.4 AES-GCM 資料路徑模組

AES-GCM 資料路徑模組包括初始向量編碼模組、AES-CTR 模組、AES 運算模組、GHASH 運算模組，AES-GCM 資料路徑模組電路架構圖，如圖 4.6 所示。

在進行在加密認證之前，先將 y 暫存器歸零（以為生成雜湊金鑰做準備），金鑰透過 in 腳位輸入 AES 運算模組，直到第一組加密資料出現在 AES 運算模組的輸出為止。此時出現在 AES 運算模組輸出的資料為雜湊金鑰 H，雜湊金鑰輸入 GHASH

運算模組並儲存於 H 暫存器。初始向量透過 in 腳位輸入 AES 運算模組，直到加密資料出現在 AES 運算模組的輸出為止。此時出現在 AES 運算模組輸出的資料為 j0，儲存在 j0 暫存器。當 GHASH 的雜湊金鑰儲存於 H 暫存器後，即可接受外部的資料輸入（包含附加認證資料與明文），並進行資料的 AES-CTR 模式加密

與 GHASH 雜湊運算。由於 AES-GCM 運算模組不需要知道資料的長度即可運算，因此需要兩個暫存器以記錄附加認證資料與明文的資料長度。當資料進入 GHASH 運算模組之前，會先經過加零（Add0）模組，此模組會根據外部輸入的有效位元信號，決定需要補 0 的位元數。當 GHASH 模組計算完所有資料，將結果與 j0 做互斥或運算，即為認證碼。

進行資料解密認證的電路與加密認證的電路，只差別於需要使用一個認證碼暫存器儲存待認證的認證碼與一個比較器用於確認認證碼的合法性。

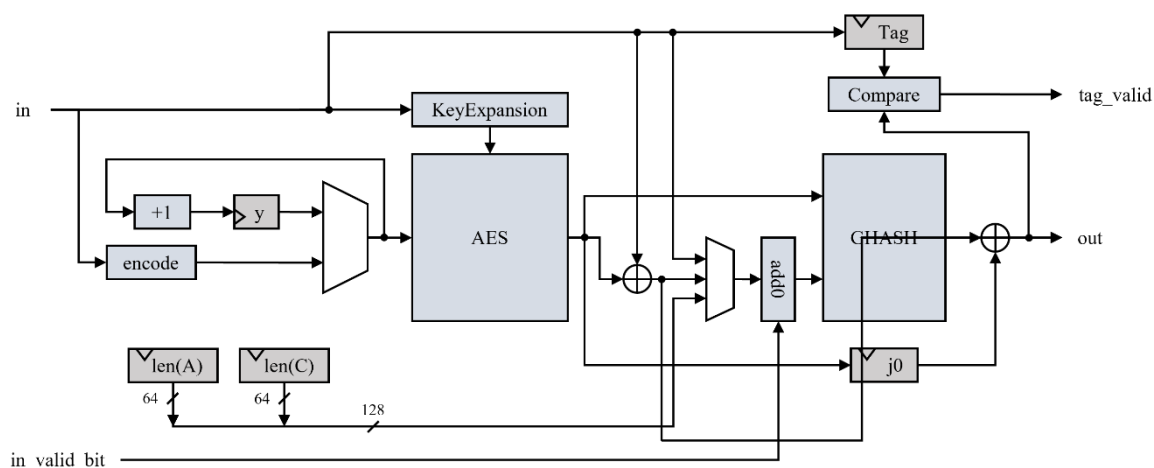


圖 4.6 AES-GCM 資料路徑模組

4.5 AES-GCM 控制單元

AES-GCM 控制單元是以有限狀態機（Finite State Machine）的方式實現，本論文採用獨熱碼（One-Hot Code）的設計方式實現有限狀態機。獨熱碼是以一個位元表示一個狀態，代表在任何狀態下只會有一個位元為 1，其餘位元皆為 0，且每

次的狀態轉換只會有量個位元發生變化。使用此方法設計狀態機可以節省組合邏輯的晶片面積且能取得更加簡單且快速的設計。圖 4.7 為 AES-GCM 控制單元的 ASM (Algorithmic State Machine) 圖。

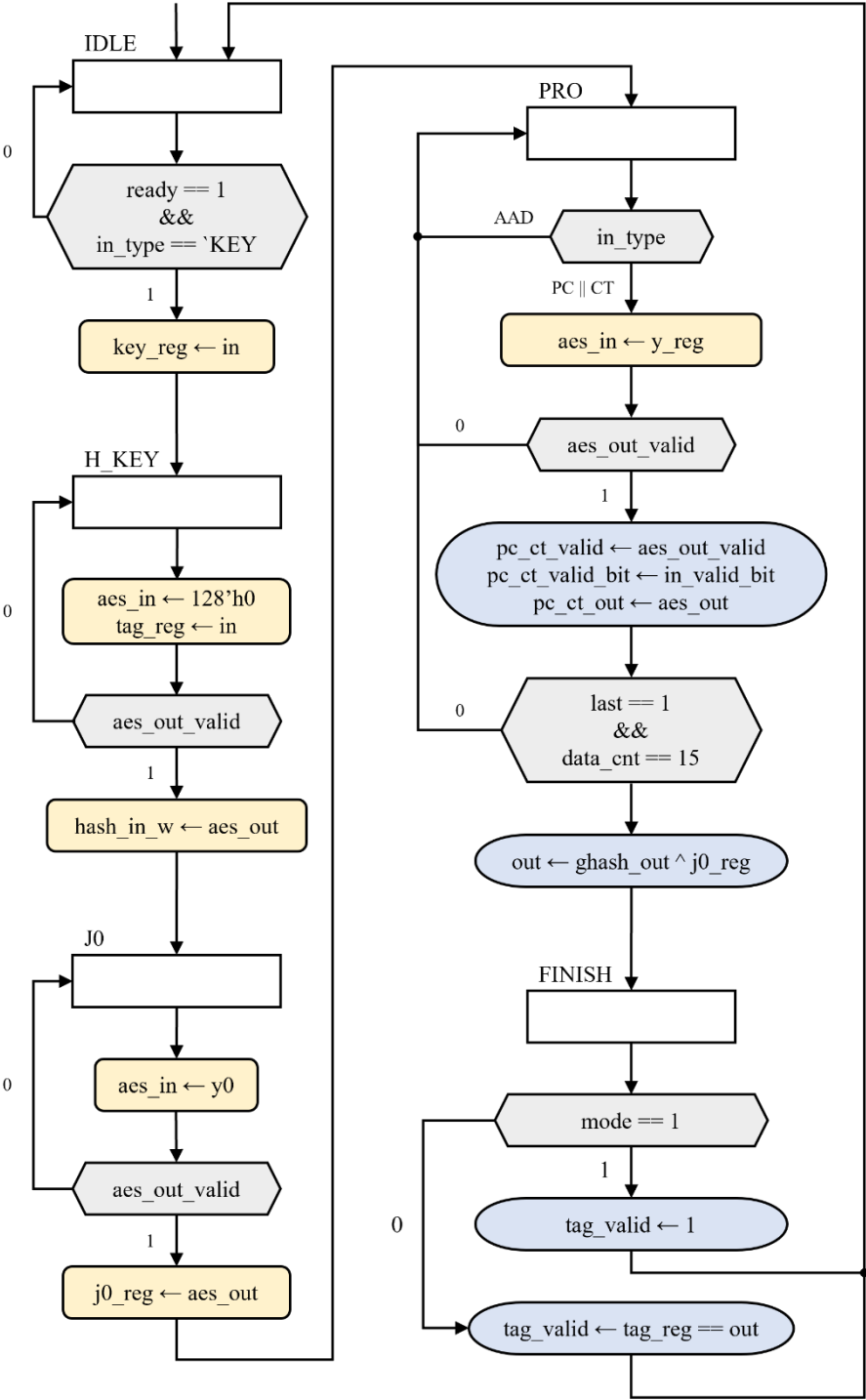


圖 4.7 AES-GCM 控制單元 ASM 圖

第5章 FPGA 與 ASIC 實現

本章將驗證所設計的 AES-GCM 加解密電路的功能和效能，展示其模擬結果。在 FPGA 的實現方面，使用了 Xilinx Virtex 7 系列的 XC7VX330T；在 ASIC 的實現方面，使用了台灣半導體研究中心（TSRI）提供的 tsmc 0.18 μm 標準元件庫。

5.1 FPGA 模擬與實現

FPGA 的實現上，選擇使用 Xilinx Virtex 7 系列的 XC7VX330T，並使用 Xilinx ISE 14.7 軟體工具進行開發。在開發過程中，使用 Verilog 硬體描述語言來描述硬體的電路的功能和結構，完整的設計與驗證流程包括以下步驟：

- (1) 行為模擬（Behavior Simulation）：使用模擬器對設計的 Verilog 檔案進行功能驗證，確保電路的正確運作。
- (2) 合成（Synthesis）：將設計的 Verilog 檔案進行邏輯合成（Logic Synthesis），生成綜合後的邏輯閘層級網表（Gate-Level Netlist）。
- (3) 實現（Implementation）：將合成後的邏輯閘層級網表轉譯（Translate）成 FPGA 的配置文件，並進行佈局與繞線（Place & Route），以確保電路能夠正確映射到目標 FPGA 設備上。
- (4) 靜態時序分析（Static Timing Analysis）：對佈局與繞線後的電路進行時序分析，確保信號在正確的時間內到達目標，以滿足時序要求。
- (5) 繞線後模擬（Post-route Simulation）：對佈局與繞線後的電路進行模擬，驗證電路的功能和時序。

5.1.1 行為模擬結果

為了確保本論文的设计符合 AES-GCM 演算法的規範，我們進行了多個層次的驗證步驟。首先對 NIST 提供的測試範例進行行為模擬，驗證了設計在暫存器轉移層次（Register Transfer Level, RTL）上的程式是否正確實現了 AES-GCM 演算法。接著在 FPGA 的合成和實現完成後，將測試範例與邏輯閘層次（Gate Level）的電路檔案以及標準延遲格式（Standard Delay Format, SDF）檔案進行佈局後的模擬驗證，確保經過合成和實現的電路仍然符合 AES-GCM 演算法的要求，並且在執行時沒有發生任何時序違反的情況。表 5.1 為 NIST 提供的其中一組測試範例，對該測試範例進行了繞線後的模擬驗證，包括加密模式和解密模式。

表 5.1 AES-GCM 測試範例

Variable（變數）	Hexadecimal Value（十六進制數值）
IV（初始向量）	cafebabefacedbaddecaf888
Key（金鑰）	feffe9928665731c6d6a8f9467308308
AAD（附加認證資料）	feedfacedeadbeeffeedfacedeadbeef abaddad2
Plaintext（明文）	d9313225f88406e5a55909c5aff5269a 86a7a9531534f7da2e4c303d8a318a72 1c3c0c95956809532fcf0e2449a6b525 b16aedef5aa0de657ba637b39
Ciphertext（密文）	42831ec2217774244b7221b784d0d49c e3aa212f2c02a4e035c17e2329aca12e 21d514b25466931c7d8f6a5aac84aa05 1ba30b396a0aac973d58e091
Tag（認證碼）	5bc94fbc3221a5db94fae95ae7121a47

圖 5.1 為加密模式下的前置輸入，設定 mode 信號為 1，並在 in_valid 信號設為 1 時將金鑰透過 in 通道輸入，接著再輸入初始向量，再依序輸入附加認證資料及明文。圖 5.2 為加密模式下的密文及認證碼輸出，當 pc_ct_valid 信號輸出為 1，表示輸出通

道 out 為密文輸出，tag_valid 信號輸出為 1，表示輸出通道 out 為認證碼輸出。

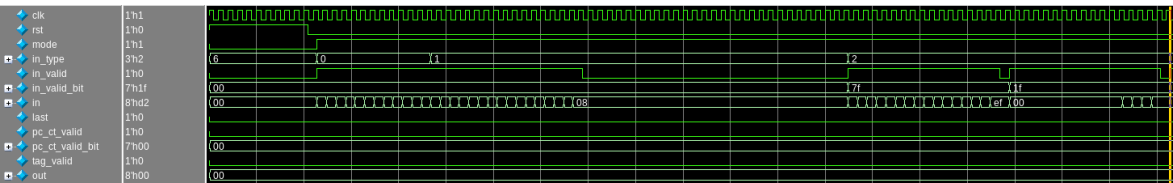


圖 5.1 加密模式_前置輸入

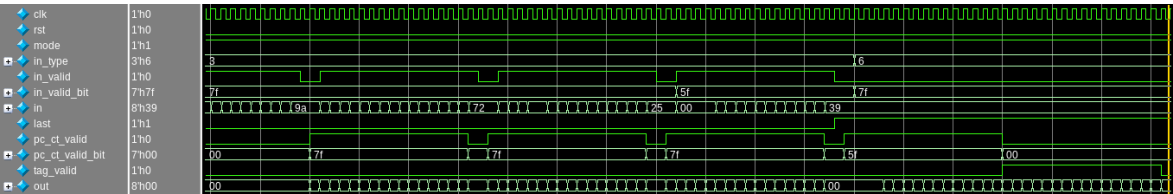


圖 5.2 加密模式_輸出

圖 5.3 為解密模式下的前置輸入，設定 mode 信號為 0，並在 in_valid 信號設為 1 時將金鑰透過 in 通道輸入，接著再輸入初始向量及認證碼，再依序輸入附加認證資料及密文。圖 5.4 為解密模式下的明文輸出，當 pc_ct_valid 信號輸出為 1，表示輸出通道 out 為密文輸出，tag_valid 信號為指示認證碼是否合法的信號。

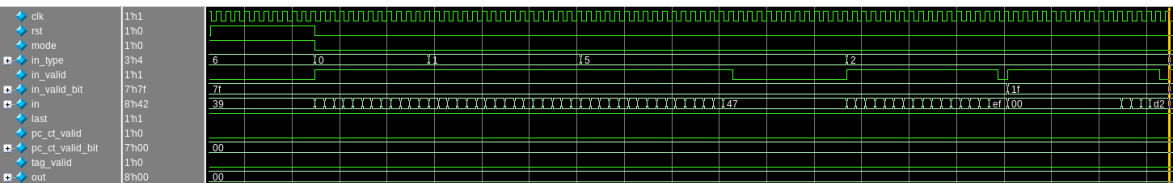


圖 5.3 解密模式_前置輸入

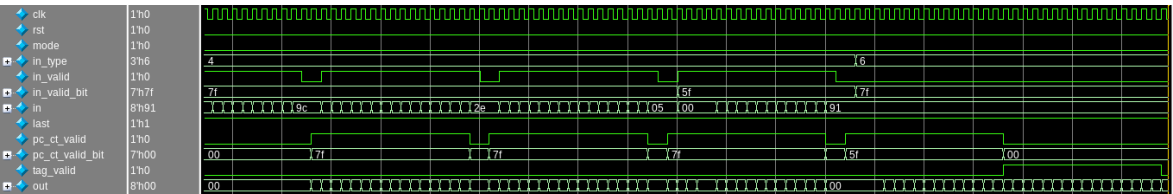


圖 5.4 解密模式_輸出

5.1.2 佈局結果

圖 5.5 至圖 5.8 為佈局後模擬，其結果與行為模擬結果相同。

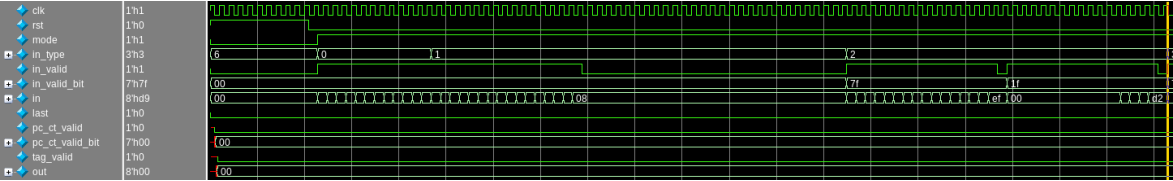


圖 5.5 佈局後加密模式模擬_前置輸入

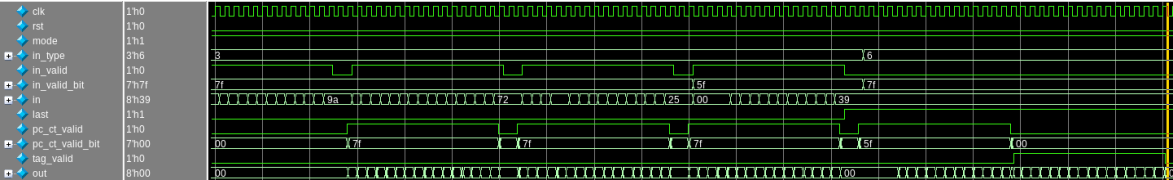


圖 5.6 佈局後加密模式模擬_輸出

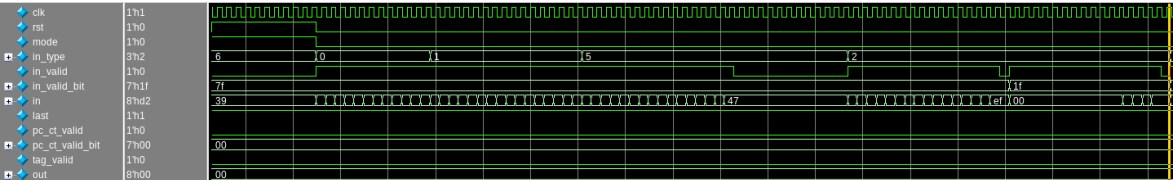


圖 5.7 佈局後解密模式模擬_前置輸入

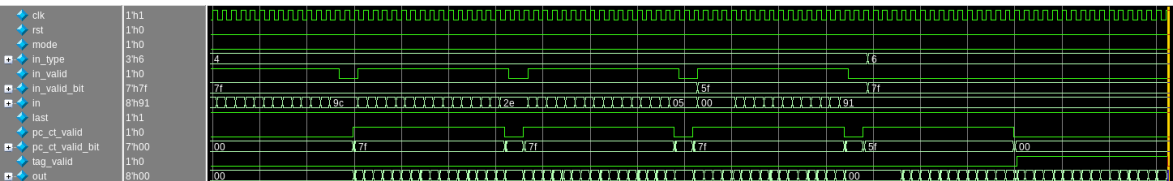


圖 5.8 佈局後解密模式模擬_輸出

5.1.3 FPGA 設計結果

圖 5.9 為 FPGA 的硬體使用資源報告。在設計上，使用了 2767 個 Registers、8801

個 LUTs 與 2815 個 Slices。圖 5.10 FPGA 的時序分析報告，根據報告可以得知，經過靜態時序分析後沒有出現任何時序違反（Timing Violation）的情況，整體工作時脈可達 181.917 MHz。

Device Utilization Summary					
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	2,767	408,000	1%		
Number used as Flip Flops	2,767				
Number used as Latches	0				
Number used as Latch-thrus	0				
Number used as AND/OR logics	0				
Number of Slice LUTs	8,801	204,000	4%		
Number used as logic	8,787	204,000	4%		
Number using O6 output only	8,382				
Number using O5 output only	112				
Number using O5 and O6	293				
Number used as ROM	0				
Number used as Memory	0	70,200	0%		
Number used exclusively as route-thrus	14				
Number with same-slice register load	12				
Number with same-slice carry load	2				
Number with other load	0				
Number of occupied Slices	2,815	51,000	5%		
Number of LUT Flip Flop pairs used	9,124				
Number with an unused Flip Flop	6,386	9,124	69%		
Number with an unused LUT	323	9,124	3%		
Number of fully used LUT-FF pairs	2,415	9,124	26%		
Number of unique control sets	23				
Number of slice register sites lost to control set restrictions	57	408,000	1%		

圖 5.9 FPGA 硬體資源使用報告

=====
Timing constraint: TS_clk = PERIOD TIMEGRP "clk" 5.5 ns HIGH 50% ;
For more information, see Period Analysis in the Timing Closure User Guide (UG612).
1462357 paths analyzed, 12091 endpoints analyzed, 0 failing endpoints
0 timing errors detected. (0 setup errors, 0 hold errors, 0 component switching limit errors)
Minimum period is 5.497ns.

圖 5.10 FPGA 時序分析報告

5.2 標準元件庫設計與實現

標準元件庫的實現上，選擇使用台灣半導體研究中心提供的 tsmc 0.18 μm 標準元

件庫，並使用 Synopsys 公司開發的 Design Compiler 與 IC Compiler 進行電路的合成與自動佈局繞線，Cell-based IC 電路設計與實現包含 RTL 設計階段、邏輯合成階段、晶片佈局與繞線階段與各階段之驗證方法，如圖 5.11 所示。

首先，需先制定晶片的規格，包括晶片使用的製程、效能與功耗的要求或是必須符合的協定（Protocol）等。當確定設計的規格之後，使用高階程式語言（High-level Programming Language）對設計進行功能性的測試，確保執行的正確性。正確性驗證完成後，即可進行硬體電路的開發。

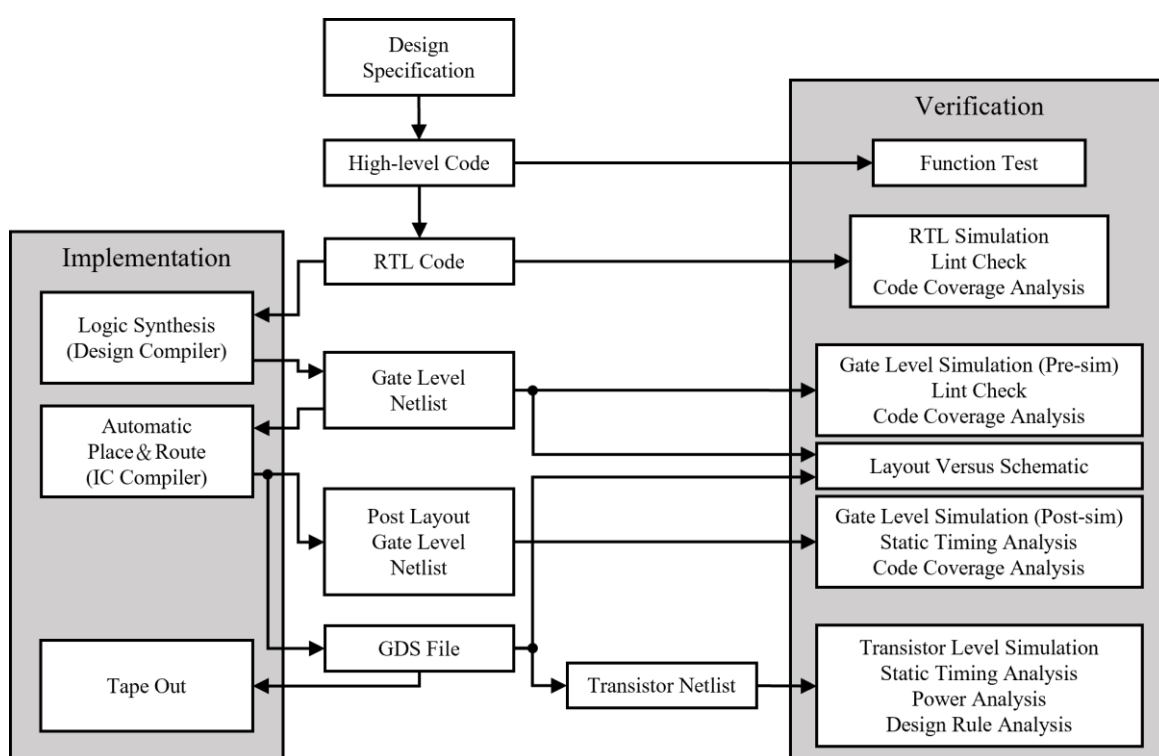


圖 5.11 Cell-based IC 設計流程

5.2.1 Design Compiler 模擬與實現結果

使用 Design Compiler 完成硬體合成後，會產生 Gate-level 的電路描述以及標準延遲訊息檔案（Standard Delay Format, SDF）。其中 SDF 檔主要記錄每個電路元件的延遲資訊，此延遲資訊能夠幫助分析電路的時序和產生更準確的模擬波形。與功能模擬

相比，考慮了時間延遲的模擬結果會更接近硬體實際運行的情況，並且能對電路效能進行更精確的評估。

使用 Cadence 的 NC-Verilog 進行帶有時間延遲的模擬，並使用 Synopsys 的 nWave 工具觀察並分析模擬的波形結果。由圖 5.12 及圖 5.13 可以觀察到其波形已帶有延遲，但結果還是正確的。

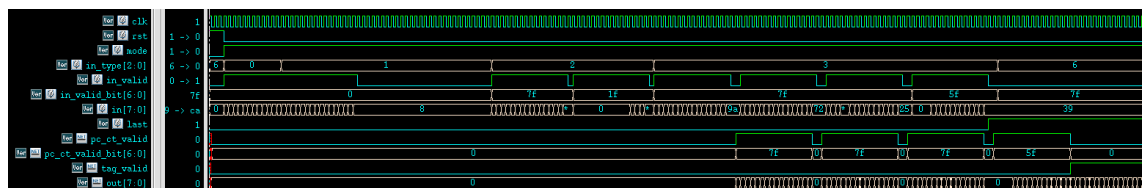


圖 5.12 合成後模擬波形圖_加密

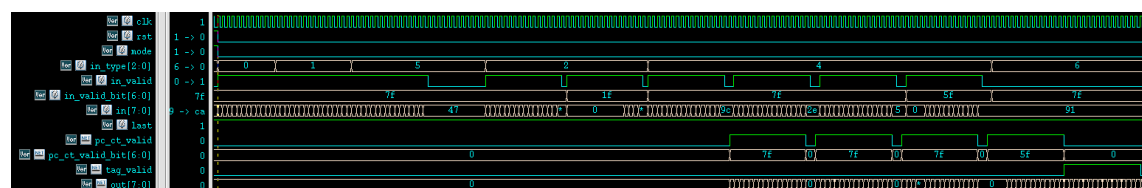


圖 5.13 合成後模擬波形圖_解密

5.2.2 IC Compiler 模擬與實現結果

Design Compiler 合成完並模擬正確後，將其輸出的 Gate-level 電路描述與電路約束（Synopsys Design Constraints, SDC）兩者匯入 Synology IC Compiler 進行佈局。佈局後的差別在於會考慮電路上金屬線的延遲，能夠得到較準確的延遲資訊，使模擬結果更接近電路真實的情況。模擬結果如圖 5.14 與圖 5.15 所示，經過佈局後的電路功能仍是正確的。佈局後的晶片規格，可參考表 5.2。佈局圖如圖 5.16。

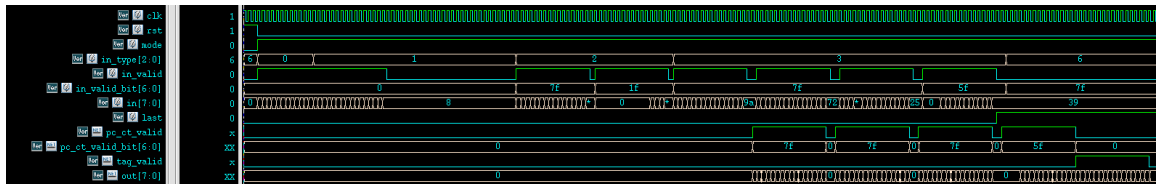


圖 5.14 佈局後模擬波形圖_加密

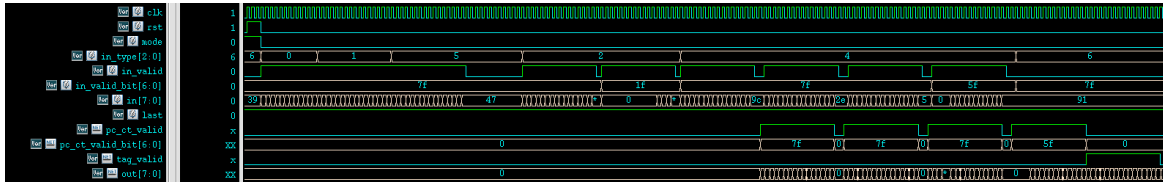


圖 5.15 佈局後模擬波形圖_解密

表 5.2 晶片規格

製程	TSMC 0.18 um
晶片面積	1417.345 μm \times 1421.21 μm
核心面積（不包含 I/O Pad）	1047.345 μm \times 1051.26 μm
等效邏輯閘數量	71.678 k
工作頻率	83.682 MHz
工作溫度範圍	0 $^{\circ}\text{C}$ \sim 125 $^{\circ}\text{C}$
工作電壓範圍	1.62 V \sim 1.98V
核心功率消耗	19.1656 mW
I/O Pad 功率消耗	1.4866 mW

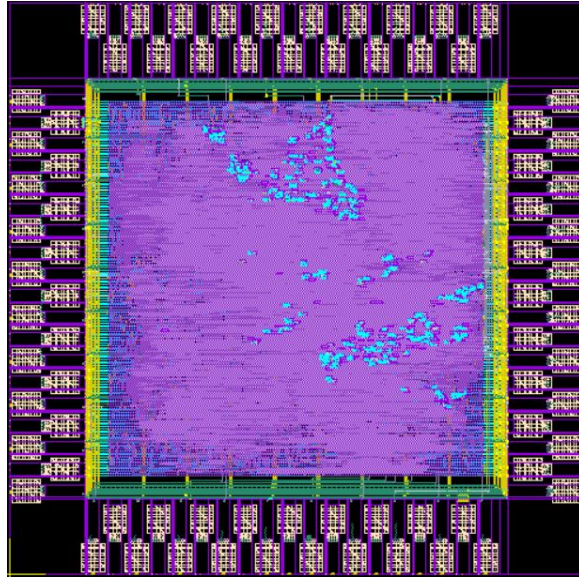


圖 5.16 晶片電路圖

5.2.3 LVS 與 DRC 結果

圖 5.17 為晶片佈局的 LVS 報告，其結果為正確，表示其晶片佈局與設計一致。

圖 5.18 為晶片佈局的 DRC 報告，其出現的 DRC 錯誤皆為 TSRI 所公告的 $0.18\ \mu\text{m}$ 製程可允許的 DRC 錯誤列表中。

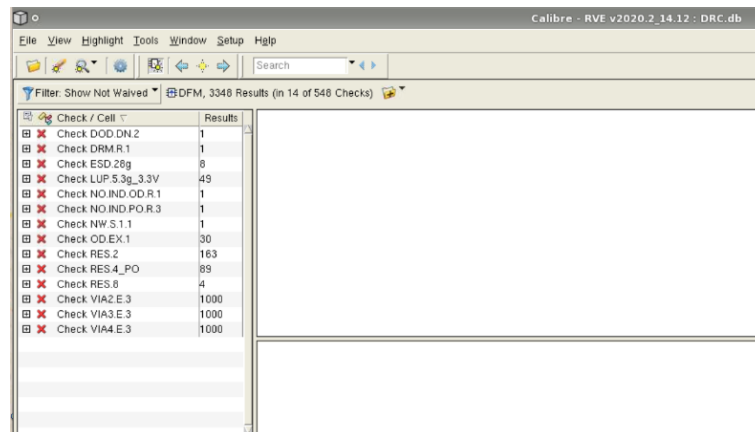
```
#####
##          C A L I B R E   S Y S T E M          ##
##          L V S   R E P O R T                   ##
#####

REPORT FILE NAME:      lvs.rep
LAYOUT NAME:           AES_GCM.spi ('DFM')
SOURCE NAME:           AES_GCM.spi ('DFM')
RULE FILE:             T18_LVS_CB.rule
HCELL FILE:            (-automatch)
CREATION TIME:         Wed Jun 21 16:07:31 2023
CURRENT DIRECTORY:     /Queue/Working/23-6-21_pw7607_CALIBRE_st2319_160705/calibrerun.9556
USER NAME:             quser
CALIBRE VERSION:       v2020.2_14.12   Thu Apr 2 15:39:27 PDT 2020

OVERALL COMPARISON RESULTS

#          #####          #
#          # CORRECT      #          #
#          #####          #
#          \_--\_/          #
```

圖 5.17 LVS 驗證



Check / Cell	Results
Check DOD.DN.2	1
Check DRM.R.1	1
Check ESD.28g	8
Check LUP.53g_33V	49
Check NO.IND.OD.R.1	1
Check NO.IND.PO.R.3	1
Check NW.S.1.1	1
Check OD.EX.1	30
Check RES.2	163
Check RES.4_PO	89
Check RES.8	4
Check VIA2.E.3	1000
Check VIA3.E.3	1000
Check VIA4.E.3	1000

圖 5.18 DRC 驗證

5.3 效能分析與比較

本節將分析電路的效能，並與其它文獻比較。

5.3.1 效能分析方式

本論文使用之吞吐量計算公式如式(5-1)，其中 n 為可同時處理多少筆 128 位元的

資料，在此固定為 1。T 為每處理一個 128 位元的資料所需的時脈數，在此固定為 18。
F 為工作頻率，在 FPGA 的最高頻率為 181.917 MHz，在 ASIC 的最高頻率為 83.682 MHz。

$$Throughput = \frac{128 \times n}{T} \times f \quad (5-1)$$

5.3.2 FPGA 效能比較

本論文在 FPGA 上實現 AES-GCM 電路架構，使用的硬體資源為 2767 個 Registers、8801 個 LUTs 與 2815 個 Slices。與其他文獻比較如表 5.3 所示。與其他文獻相比本設計擁有最低的資源使用量。

表 5.3 FPGA 相關文獻比較

Design	Device	Max Frequency	Resources	Throughput
Ours	Virtex 7 XC6VLX240T (28 μ m)	181.917 MHz	2767 Registers 8801 LUTs 2815 Slices	1.294 Gbps
[22]	Virtex 7 XC5VLX220 (28 μ m)	222.568 MHz	11,063 Registers 16078 LUTs 5,621 Slices	56.97 Gbps
[17]	SE5CSEMA4U23C6N (製程跟 Virtex 6 一樣) (40 μ m)	50 MHz	2,572 Registers 4,273 LUTs	0.29 Gbps
[18] GCM1	Virtex 4 XC4VLX40 (90 μ m)	119 MHz	13,523 Slices	15.23 Gbps
[18] GCM2	Virtex 4 XC4VLX40 (90 μ m)	161 MHz	16,396 Slices	20.61 Gbps

5.3.3 ASIC 效能比較

本論文在 ASIC 上實現 AES-GCM 電路架構，使用的有效邏輯閘為 71.678k 個。
與其他文獻比較如表 5.4 所示。與其他文獻相比本設計擁有最低的有效邏輯閘數量。

表 5.4 ASIC 相關文獻比較

Design	Technology	Max Frequency	Gates	Throughput	Power
Ours	0.18 μm	83.682 MHz	71.678 k	0.595 Gbps	19.1656 mW
[22]	0.18 μm	197.239 MHz	348.986 k	50.49 Gbps	860.648 mW
[19]	0.18 μm	271 MHz	498.658 k	34.69 Gbps	
[20]	0.13 μm	333.3 MHz	297.542 k	42.67 Gbps	
[21]	0.13 μm	200 MHz	600.44 k	102.4 Gbps	

第6章 結論與未來展望

本論文設計並實現了一個應用於 IoT 的 AES-GCM 電路架構。對於 AES 演算法中的位元組替代轉換單元，根據硬體資源特性，提供各自最佳化的設計方法。在 FPGA 中，使用直接邏輯映射的方法實現，可以節省約 32.9% 的 LUTs 數量。而在 ASIC 中，使用複合場運算的方法實現，可以節省約 55.6% 的等效邏輯閘數量。在 GHASH 模組中，為了降低硬體資源，使用 Karatsuba 乘法演算法實現的有限場乘法器，可以節省約 50.7% 的硬體資源。

完成的 AES-GCM 加解密設計分別使用 FPGA 與 ASIC 實現與驗證。在 FPGA 實現上，使用 Xilinx 公司的 Virtex 7 系列的 XC7VX330T，其合成結果為晶片工作頻率為 181.917 MHz，最高吞吐量為 1293.632 Mbps，使用 2767 個 Registers、8801 個 LUTs 及 2815 個 slices；在 ASIC 實作上，使用 tsmc 0.18 μm 製程元件庫，其合成結果為晶片工作頻率為 83.682 MHz，最高吞吐量為 595.072 Mbps，晶片核心面積為 1047.345 $\mu m \times 1051.26 \mu m$ ，等效邏輯閘數量約為 71677 個，核心功率消耗為 19.1656 mW，I/O Pad 功率消耗為 1.4866 mW。

在未來研究方面，希望可以對於延遲路徑的部分，協調每個模組之間的設計，降低關鍵路徑的延遲時間，也能提升整體的吞吐量。對於功率消耗的部分，分析每的小模組的電源消耗程度，適當地加入時脈閘控（Clock Gating）與相關控制信號來避免暫存器不必要的轉換動作（Switching Activity），以減少動態功率的消耗，期望能發展出低面積及低功耗的 AES-GCM 硬體架構。

參考文獻

- [1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” in *IEEE Communications Surveys & Tutorials*, volume 17, no. 4, pp. 2347-2376, 2015. doi: 10.1109/COMST.2015.2444095.
- [2] David A. McGrew and John Viega, “The Galois/Counter Mode of Operation (GCM),” <https://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf>, May 2005.
- [3] D.A. McGrew and J. Viega, “The Security and Performance of the Galois/Counter Mode (GCM) of Operation (Full Version),” *Lecture Notes in Computer Science*, volume 3348, pp 343-355, Springer, Berlin, 2004.
- [4] W. Diffie and M. Hellman, “New directions in cryptography,” in *IEEE Transactions on Information Theory*, volume 22, no. 6, pp. 644-654, November 1976. doi: 10.1109/TIT.1976.1055638.
- [5] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini and Y. Khamayseh, “Comprehensive study of symmetric key and asymmetric key encryption algorithms,” in *Proceedings of the 2017 International Conference on Engineering and Technology (ICET)*, pp. 1-7, Antalya, Turkey, 2017. doi: 10.1109/ICEngTechnol.2017.8308215.
- [6] M. Dworkin, “Recommendation for Block Cipher Modes of Operation: Methods and Techniques,” *NIST Special Publication 800-38A*, December 2001.
- [7] William Stallings, *Cryptography and Network Security Principles and Practice*, Seventh edition, Pearson, October 2016. doi: 10.1007/3-540-48658-5_22.
- [8] M. Bellare, O. Goldreich, and S. Goldwasser, “Incremental Cryptography: The Case of Hashing and Signing,” *Advances in Cryptology — CRYPTO '94*, volume 839, pp. 216–233, Springer, Berlin, Heidelberg, 1994.

- [9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Asymmetric Cryptography: Primitives and Protocols*, third edition, The MIT Press, July 2009.
- [10] Y. Huang, Y. Lin, K. Hung and K. Lin, "Efficient Implementation of AES IP," in *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems*, pp. 1418-1421, Singapore, 2006.
- [11] FIPS Publication 197, *Advanced Encryption Standard (AES)*, U.S. DoC/NIST, November 2001.
- [12] X. Zhang and K. K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 12, no. 9, pp. 957-967, September 2004.
- [13] A. Joshi, P. K. Dakhole and A. Thatere, "Implementation of S-Box for Advanced Encryption Standard," in *Proceedings of the 2015 IEEE International Conference on Engineering and Technology (ICETECH)*, pp. 1-5, Coimbatore, Tamil Nadu, India, March 20, 2015.
- [14] Zhengzheng Ge, G. Shou, Y. Hu and Z. Guo, "Design of low complexity $GF(2^m)$ multiplier based on Karatsuba algorithm," in *Proceedings of the 2011 IEEE 13th International Conference on Communication Technology*, pp. 1018-1022, Jinan, China, 2011. doi: 10.1109/ICCT.2011.6158033.
- [15] A. Karatsuba and Yu. Ofman, "Multiplication of Many-Digital Numbers by Automatic Computers," in *Proceedings of the USSR Academy of Sciences*, volume 14, no. 145, pp. 293-294, October 1962.
- [16] A. A. Karatsuba, "The Complexity of Computations," in *Proceedings of the Steklov Institute of Mathematics*, volume 211, pp. 169-183, January 1995.
- [17] S. Koteshwara, A. Das and K. K. Parhi, "Performance comparison of AES-GCM-SIV and AES-GCM algorithms for authenticated encryption on FPGA platforms," in *Proceedings of the 2017 51st Asilomar Conference on Signals, Systems, and Computers*,

pp. 1331-1336, Pacific Grove, CA, USA, November 2017.

- [18] G. Zhou, H. Michalik and L. Hinsenkamp, "Efficient and High-Throughput Implementations of AES-GCM on FPGAs," in *Proceedings of the 2007 International Conference on Field-Programmable Technology*, pp. 185-192, Kitakyusyu, Japan, December 2007.
- [19] Bo Yang, Sambit Mishra, and Ramesh Karri, "High Speed Architecture for Galois/Counter Mode of Operation (GCM)," *Cryptology ePrint Archive*, ECE Department Polytechnic University, Brooklyn, NY, Jun 2005.
- [20] A. Satoh, "High-Speed Hardware Architectures for Authenticated Encryption Mode GCM," in *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4831-4834, Kos, Greece, May 2006.
- [21] A. Satoh, "High-Speed Parallel Hardware Architecture for Galois Counter Mode," in *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1863-1866, New Orleans, USA, May 2007.
- [22] 莊任華，設計與實現一個高效能 AES-GCM 加密認證演算法之 IP，碩士論文-國立台灣科技大學電子工程系，2022 年。
- [23] 張祐菘，基於 AXI4 介面的管線是 AES 矽智財設計與驗證，碩士論文-國立台灣科技大學電子工程系，2019 年。
- [24] 陳思云，設計與實現一個高效能 AES-CCM 加密驗證演算法之 IP，碩士論文-國立台灣科技大學電子工程系，2021 年。