第2章 背景介紹

本章先介紹第三代安全雜湊演算法（SHA-3）的背景、相關資料，然後介紹SHA-3 的原理。

2.1 安全雜湊演算法

安全雜湊演算法屬於密碼學中的一個分支，如圖 2-1 所示。密碼學分為四大類：編碼、雜湊、對稱式加密、非對稱式加密，本篇論文為 SHA-3 (Secure Hash Algorithm)，屬於雜湊函數這一類。

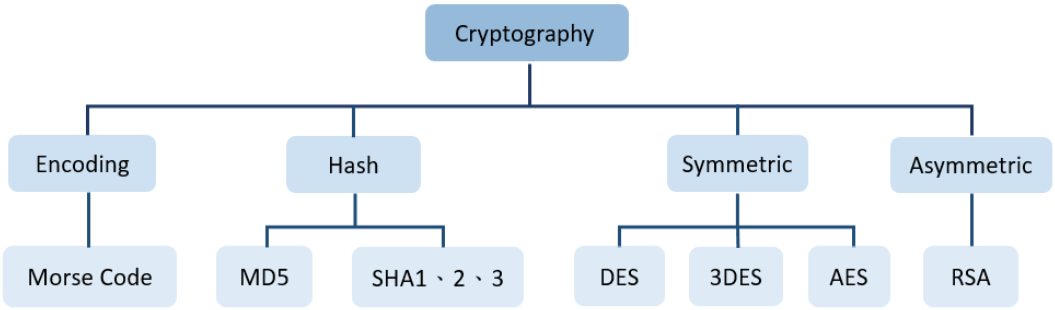

圖 2-1 密碼學的種類

雜湊函數（Hash Function），又稱哈希函數、散列函數，擁有單向且不可逆的性質，而其主要目的是將一可變長度的輸入明文（或訊息）生成需要長度的輸出值（或訊息）。經過雜湊函數計算出來的值稱為雜湊值，它可以用來識別傳送的訊息（資料）是否被竄改，以保證接收到的訊息是正確且完整的。

- 雜湊函數的特性：
  - 任意長度的明文輸入、可變長度的雜湊值輸出。
  - 原像防禦（Preimage resistant）為單向性(one-way property)：即無法利用雜湊值回推明文的內容。
  - 弱碰撞防禦（Second preimage resistant 或 weak collision resistant)：就是給定一個明文 m1 時，很難找到另一個明文 m2，它們的雜湊值是相同的。
  - 強碰撞防禦(Collision resistance 或 strong collision resistant)：就是很難找到兩個不同的明文 m1 和 m2，使得 h (m1) = h (m2) 它們的雜湊值是相同的。
  - 假亂數性（Pseudorandomness）：雜湊值看起像一組隨機亂數，但它並不是隨機亂數，而是經由運算而產生的數值。

2.2 SHA-3 的誕生

此篇論文的 SHA-3 正是雜湊演算法，但是與其它演算法不同的是雜湊演算法並沒有加密、解密的功用，其主要目的是產生一組驗證碼來確認資料是否被修改過。此種運算屬於單向的，無法利用驗證碼回推明文的內容，而接收者對照驗證碼看是否一致，確認收到的資料正確。

雜湊函數有分好幾種，像是：MD2、MD4、MD5 (Message-Digest Algorithm)、SHA-1、SHA-2、SHA-3，而最先開始由 MD2 演化到 MD5，並基於 MD5 的設計架構做延伸，做出 SHA-1，並由 NIST 在 1995 年發布，但是已經在 2005 年被研究人員發現了有效的攻擊方法，代表此雜湊函數已經不夠安全，所以使用 SHA-2 來替代。雖然目前沒有明確的研究發現可攻擊 SHA-2 的漏洞，不過因為 SHA-1 與 SHA-2 的設計結構非常相似，擔心 SHA-2 也會不安全，因此 NIST 開始號召、投稿新的標準，最後以 KECCAK 演算法勝出，在 2015 年由 NIST 通過聯邦資訊處理標準 (Federal Information Processing Standard，FIPS) 正式發表。

SHA-3 的家族成員分了兩種如圖 2-2 所示，四個加密雜湊函數、兩個可擴展輸出函數。加密雜湊函數與可擴展輸出函數的差別在於前者是選擇好哪一個雜湊容量後就決定好輸出的長度；後者是可以決定輸出的長度，而輸入的明文長度則一樣都是沒有限制的。

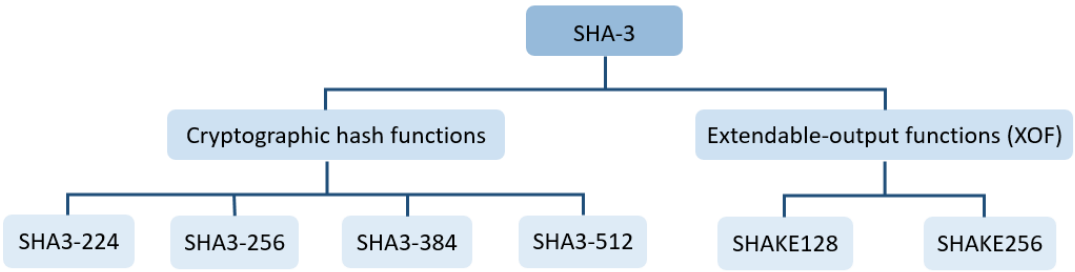

圖 2-2 SHA-3 家族類別

2.3 海綿結構

SHA-3 的核心架構為海綿結構(Sponge construction)又稱為海綿函數(Sponge function)，它的特性是接收任意長度的輸入，並且可以輸出任意長度的輸出，而它的運作也直接反映在名字身上。海綿結構可以像海綿一樣，將明文做「吸收」、並且依照需要的輸出長度做「擠壓」，雜湊函數就是利用此架構來完成任意長度的輸入和任意長度的輸出的特性。

所有的 KECCAK-f 置換都以海綿函數作為基礎，這是一種簡單的迭代構造並且以多速率填充 (multi-rate padding) 規則來進行填補。

迭代結構如圖 2-3 所示，在海綿函數上要先組成一變數 s，此變數 s 是 $b = r + c$ 位元，該變數的初始化是全為零的值，並在每次迭代後進行更新。

- 名詞介紹：

i. ***r*** (Bitrate, 比特率)：

此值是用來表示輸入消息進行分塊時的大小位元數，也就是說每次處理迭代函數時的位元數之大小為 r，因此 r 值越大，海綿結構處理訊息的速率也越快。

ii. ***c*** (Capacity, 容量)：

容量可以衡量海綿結構可實現的複雜性以及可實現的安全級別的量度，因為 $b = r + c$，b 為固定的大小，所以可以透過相應地增加容量 c 和降低比特率 r 來換取安全性。

iii. ***b*** (寬度)：

KECCAK-p 置換的所需寬度位元數。

➢ 海綿函數分為兩個階段：

i. 吸收階段(Absorbing phase)：每次迭代使用零值填充欲處理的輸入塊，即是將 r 與填滿零值的 c 擴充到 b 所需的長度，形成擴展消息塊。並且與 s 進行 XOR，生成迭代函數的輸入值，而 s 則是每次迭代後的輸出值。

ii. 擠壓階段(Squeezing phase)：將 s 的前面幾個 r 位元數當輸出，輸出區塊稱為 $Z_0, Z_1, \ldots\ldots, Z_{j-1}$，若擠壓的 r 位元數還沒達到所需輸出位元數，則會繼續將 s 進行迭代繼續擠壓，直到滿足所需的輸出值。

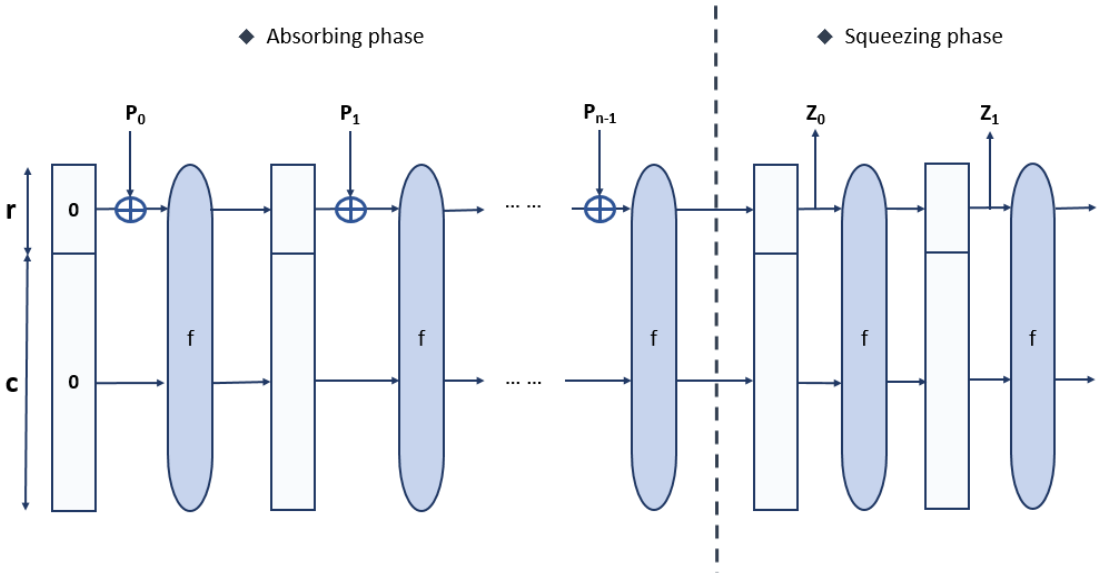

圖 2-3 海綿函數吸收、擠壓示意圖

➢ 海綿構造的運行：

i. 由圖 2-4 可以看到海綿結構事先把欲處理的明文先進行寬度為 r 位元的切割分塊，到最後一塊不夠 r 位元時，對該方塊進行填充，擴展為 r 位元。

ii. 圖 2-5 顯示基於海綿結構的 KECCAK 函數使用已分好的區塊 P_0 進行處理，先與 s 區塊做 XOR 的運算再進到迭代函數 f 裡運算，直到欲處理的區塊都處理完後，吸收階段才結束。

iii. 擠壓階段則是每次都擠出 r 位元的輸出，直到長度符合需要的輸出長度。

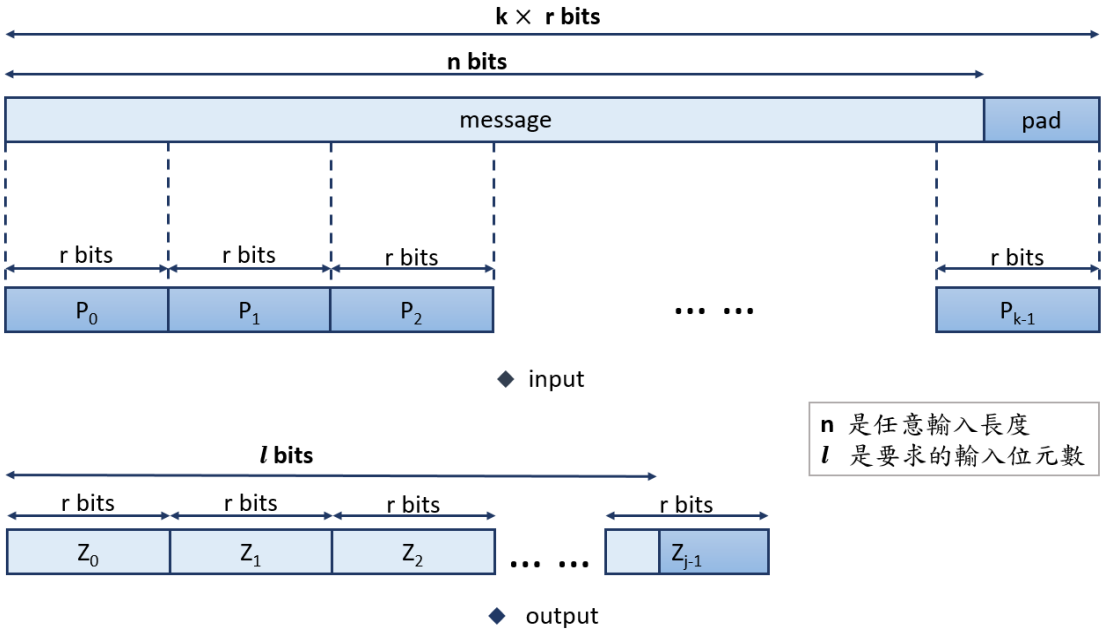

圖 2-4 海綿函數的輸出與輸入

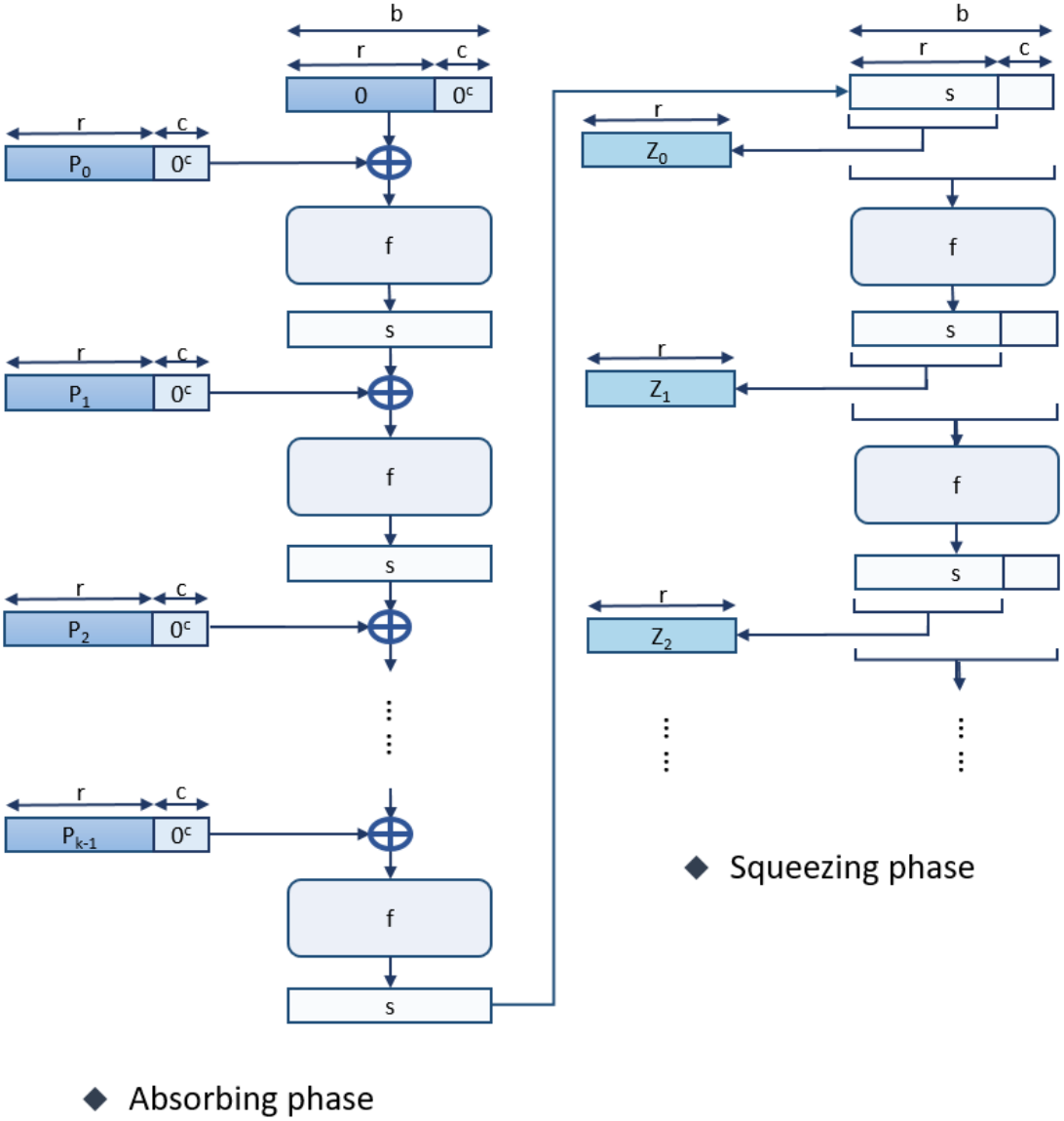

圖 2-5 海綿結構的吸收階段與擠壓階段

以上是海綿結構的運行，接下來 2.4 小節介紹 KECCAK 演算法的迭代函數 f。

2.4 KECCAK-p 排列

在此小節裡，我們將要介紹 KECCAK-p 排列，此函數有兩個參數：

i. 要被置換、排列的字串的固定長度，稱為 b。

ii. 內部被置換、排列的迭代次數，稱為 n_r (回合)。

並記做 KECCAK-p [b, n_r]，表 2-1 為相關變數對照表。

- b 可以為{ 25 , 50, 100, 200, 400, 800, 1600 }。
- n_r 可以為任意整數。

KECCAK-p 排列的名詞介紹：

i. Rnd：表示在 KECCAK-p 裡的回合數。

ii. 映射步驟（Step mapping）：表示五個置換、排列的映射步驟組成。

iii. 狀態（State）：每一次置換、排列都是根據更新後的 b 個位元數進行運算，其更新後的 b 個位元數稱為狀態。而最初狀態為排列的輸入值。

置換的狀態由 b 個位元數組成，然而相關的變量：

a. w：其算法為 b / 25。

b. ℓ：其算法為 $\log_2(b/25)$。

表 2-1 KECCAK-p 寬度對照表

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
ℓ	0	1	2	3	4	5	6

- 狀態矩陣 (State array)：

  狀態矩陣是狀態的三維表示形式，用此方式進行索引。

  如圖 2-6 所示，我們將它呈現成 5 × 5 × w 的位元數的三圍矩陣，把此矩陣的每個位元數都以一個座標為 A [x, y, z]的方式表示，其中 $0 \leq x$、$0 \leq y$ 、$0 \leq z \leq w$。

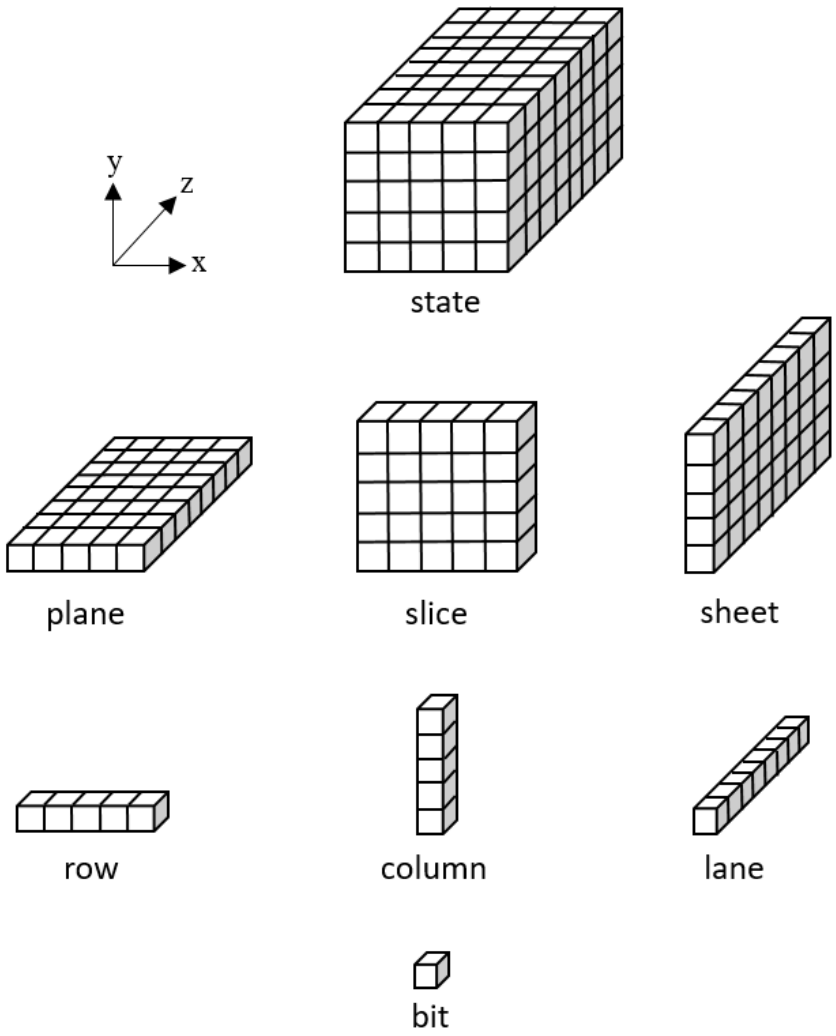

圖 2-6 狀態數組示意圖

- KECCAK 的默認值：

  i. $c = 1024$ 位元

  ii. $r = 576$ 位元

  iii. $b = 1600$ 位元

由上面的默認值我們得出 SHA-3 的 $w = 1600 / 25 = 64$ 位元

- 映射步驟：

  每一回合都是由五個步驟的排列、置換運算組合成的分別叫做：

  i. θ (Theta)

  ii. ρ (Rho)

  iii. π (Pi)

  iv. χ (Chi)

  v. ι (Iota)

以上為 KECCAK 的介紹，接下來的 2.5 小節將會詳細介紹 SHA-3 的運算，其中也有上面那五個步驟的迭代函數運算。

2.5 SHA-3 的作法

2.5.1 加密雜湊函數

由上面 2.2 小節可以得知 SHA-3 分成了兩類，一個是加密雜湊函數和一個是可擴展輸出函數。而本篇論文是做加密雜湊函數，然而這類的雜湊函數有四種區塊大小，分別為：224、256、384、512，每選項的安全度不同，數字越大越安全，而數字乘 2 後可以計算出 c 的大小，再由 $b-c=r$ 推算 r 的大小，每個相關參數的大小由表 2-2 呈現。

表 2-2 加密雜湊函數參數

Encryption type	**224**	**256**	**384**	**512**
Message Digest Size	224	256	384	512
Message Size	Unlimited	Unlimited	Unlimited	Unlimited
b	1600	1600	1600	1600
Capacity：*c*	448	512	768	1024
bitrate：r (Block Size)	1152	1088	832	576
Word Size	64	64	64	64
Number of Rounds	24	24	24	24
Collision Resistance	2^{112}	2^{128}	2^{192}	2^{256}
Second Preimage Resistance	2^{224}	2^{256}	2^{384}	2^{512}

2.5.2 常規的狀態矩陣：

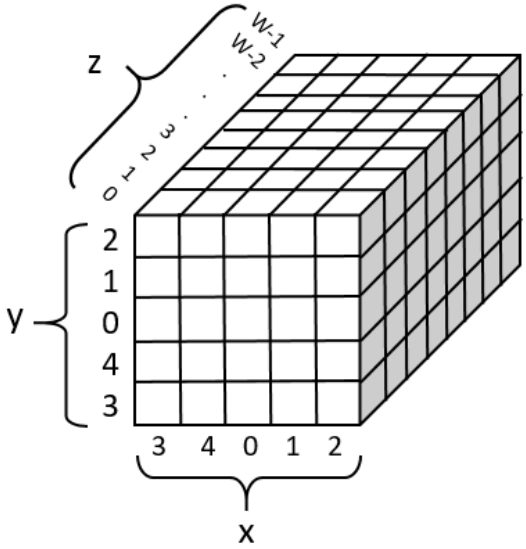

圖 2-7 狀態矩陣的 A [*x, y, z*]座標位置

由表 2-2 已經得知此篇論文的 *w* 寬度為 64 位元，圖 2-7 可以看到狀態矩陣的常規座標，此三維矩陣有 1600 個位元數，而內容會隨選擇的安全強度而定。以 SHA3-384 為例如圖 2-8 所示，我們得知 *c* 的部分是 $384 \times 2 = 768$ (位元)，所以這部分全填零值，而剩下 $1600 - 768 = 832$ (位元)的部分就是儲存欲處理的明文資料(*r*)，組合起來變成一個區塊(*b*)，然後先與變數 *s* 進行 XOR，再進到迭到函數 *f* 裡面進行運算。

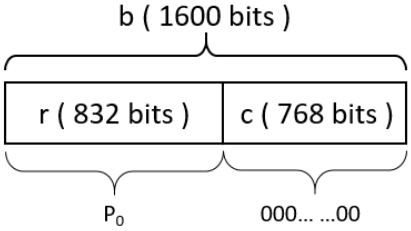

圖 2-8 SHA3-384 矩陣示意圖

2.5.3 填補

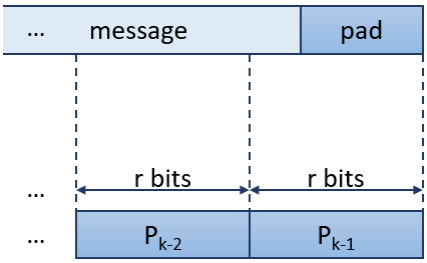

圖 2-9 填補的示意圖

如圖 2-9 所示，填補 (Padding) 是為了在明文無法完整整除 *r* 位元的時候，將不夠的地方補齊的部分，而填補的分成兩種方式，分別為：簡單填補 (Simple padding)、多速率填補 (Multi-rate padding)。此篇論文的 SHA-3 為多速率填補，接下來要講解兩種的差別。

1. 簡單填補：

   填補方式為 pad10*，作法是先補一個位元數為 1 的位元後再用把長度不夠 *r* 的地方全部都補 0。

2. 多速率填補：

   填補方式為 pad 10*1，作法是先補兩個為 01 的位元後再填一個 1 後就開始用 0 把長度補齊，直到最後一個位元時補 1。

在 KECCAK 內部是採取小在前排列法(Little endian)，即是起始位址的第零位元為最小有效位元（Least Significant Byte，LSB），最後位址為最大有效位元(Most Significant Byte，MSB)，然而與 NIST 採用的大在前排列法(Big endian)的方法完全相反，NIST 是起始位址為最大有效位元，而結束位址在最小有效位元，為了讓 KECCAK 內部約定和 API (Application Programming Interface)外部約定兼容，設計者在進入迭代函數前要進行處理，即為 Formal bit-reordering。下面將演示兩個如何轉換由大在前排列法到小在前排列法的方法，轉換的資料以一個位元組為單位，因為預設明文為字符串，轉換成 ACSII 也是八位元、以一個位元組為基本單位。

第一個方法為每個位元組都翻轉一遍再做多速率填補；第二個方法會把每個位元組按照區塊的大小先將每個位元組轉換排好再填補上轉換過後的多速率填補[16]。

方法（一）:

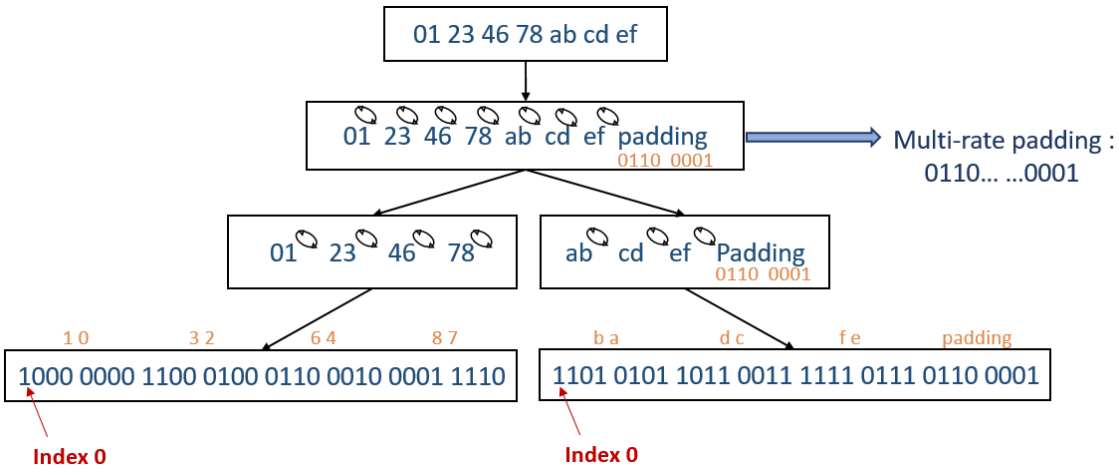

圖 2-10 多速率填補的填補方法(一)

由圖 2-10 做例子，四個位元組為一個區塊，需要分成兩組，而後面一組的訊息不夠，因此需要填充填補位元（padding），然而分完後便把每個字都反轉，即由大在前排列法轉成小在前排列法，而填補位元則不用翻轉直接進行填

補，填補後即完成多速率填補的動作，此方法的讀取一樣是從最大有效位元進行讀取。

方法（ 二 ）：

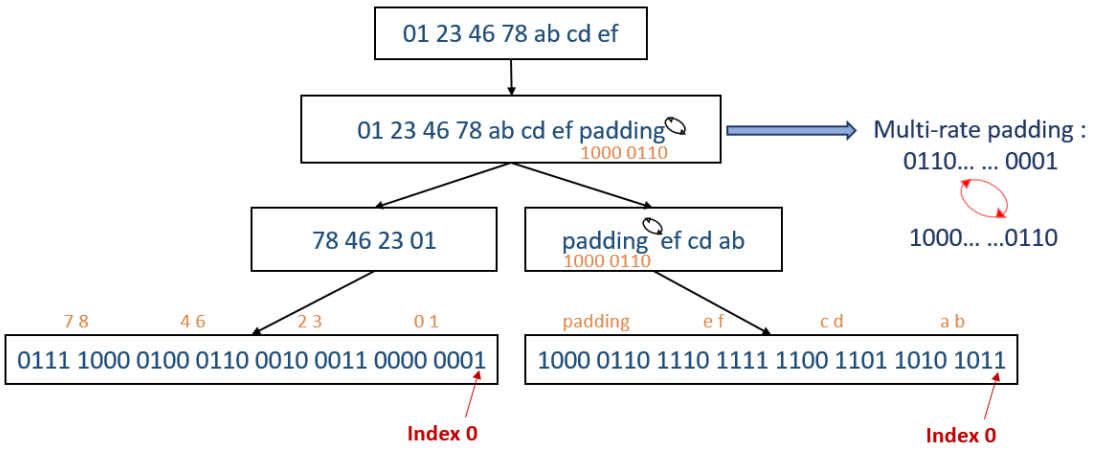

圖 2-11 多速率填補的填補方法(二)

方法二如圖 2-11 所示是使用與上面一樣的例子以做比較，它一樣分成兩個區塊，第二個不夠要補上填補位元。它與第一個例子的不同之處在於它在分成兩個區塊的時候已經先更換位置，前面的移到後面，後面移到前面，然後在下一步看得出來就是照翻成二進制，翻轉的是填補位元的部分，翻轉後加上去，不過這次是從低序位元讀取至高序位元。

以上是兩種多速率填補的填充說明以及將大在前排列法轉換成小在前排列法的方法。本篇論文採取的是第二種方法，在排序上會較有效率。

2.5.4 迭代函數 f

此小節要介紹的是迭代函數 f (Iteration function f)，它用來運算已經被分塊的輸入訊息，也就是將 r 個位元與 c 個位元的 1600 個位元之 s 變數進行運算，s 為 $5 \times 5 \times 64$ 的矩陣。

64 位元為一個通道，而 $5 \times 5 \times 64$ 的矩陣擁有 25 條通道。此矩陣會經過 24 回合的處理，每一回合都有 5 個步驟，每個步驟都是置換或是替換矩陣的運算操作，只有最後一個步驟不同，最後一個步驟是使用不同的回合常數進行置換，而這五個步驟為迭代函數的組合如圖 2-12 所示。

記做： $\mathrm{Rnd}(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$

總結上面 5 個步驟，前面的四個步驟都是由簡單的布林運算 (NOT、AND、OR)、旋轉來完成，不需要算術運算式或是依賴數據做運算，只有最後一個步驟需查表，因此 SHA-3 可以在硬體中輕鬆有效地實現。

- 迭代函數：
  - 處理位元數 ：1600 位元
  - 處理矩陣 ：$5 \times 5 \times 64$
  - f 函數回和數：24 回合
  - 每回合 5 個步驟

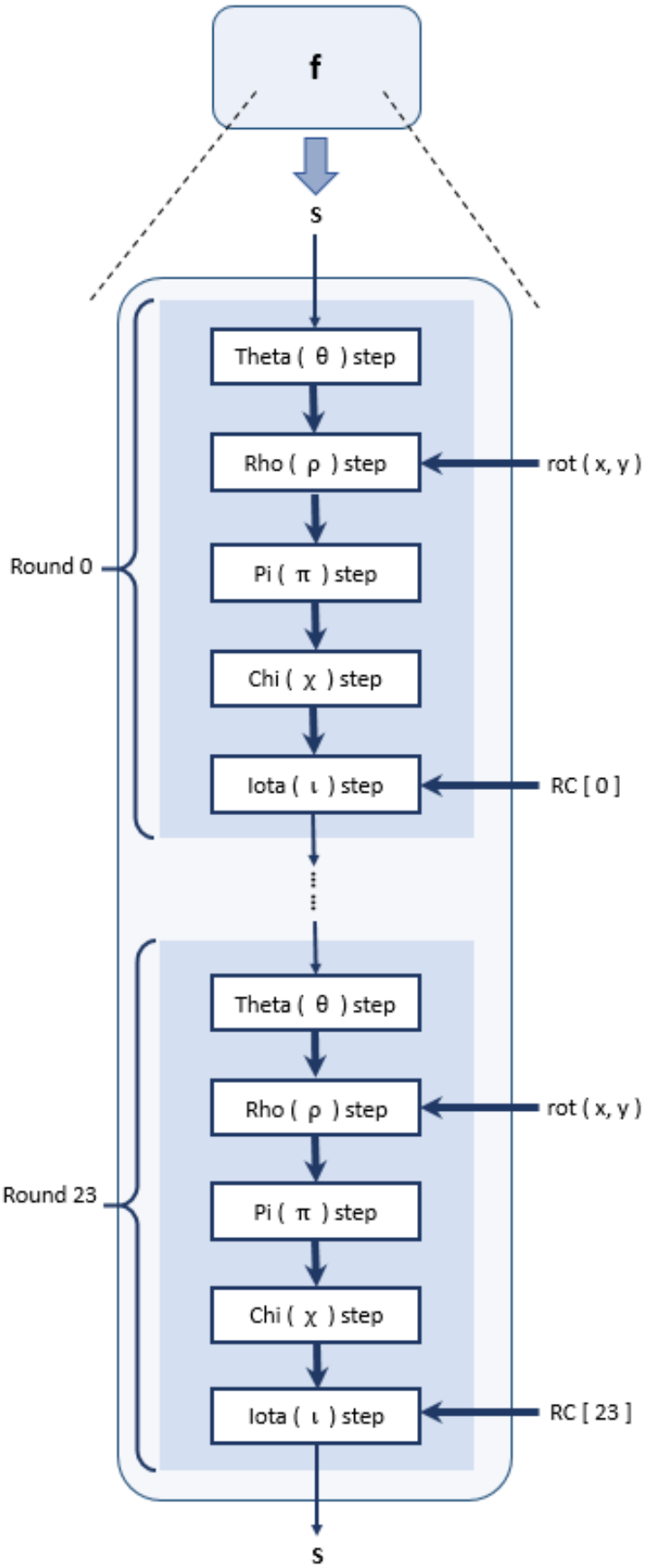

圖 2-12 迭代函數 f 步驟