大家好, 我是來自彰化師大的劉原福

今天我要報告的研究是可用於加密系統的雙重錯誤更正算術乘積編碼AN-code

我的指導教授是黃宗柱教授

Outline:

這是我的Outline, 一開始先介紹研究動機以及發想的技術

接著說明之前的一些有關AN-code的previous work

之後會講解我們所提出的技術特點及可用性

最後會透過實驗及評估的結果去證明我們的技術是比之前的技術效果來的更好

Introduction 1:

在這個硬體迅速發展的時代, 不管是運算量還是運算資料的大小都快速地增加,

而在資料運算過程中, 資料的保密以及資料的正確性儼然成為一個關鍵性的問題

這方面議題是值得我們去探討的

Introduction 2:

而加密技術在資料的保密中扮演一個相當重要的角色, 其中像是RSA:非對稱加密系統更是常見的加密技術之一

圖中列出了RSA在加密及解密的兩個重要算式,可以看到在這運算的過程涉及大量的指數乘法及模數運算

而且RSA處理的數值又相當的大, 高達2048位元, 資料一旦在這過程出錯將會造成嚴重的損失

Introduction 3:

自從參考文獻[10]的作者提出乘積碼和通道碼的關聯後, 從未有人對雙重錯誤更正進行進一步的解析

而不像傳統的AN-code只能解單一算術權重錯誤, 我們提出基於查找表的AN-code是可以用來更正雙重算術權重錯誤

除了提出基於查找表的方法, 我們還針對LUT面積過大的問題提出了一個折衷方法:藉由搭配軟體搜尋的方式用以減少硬體的使用, 使得整體的運算資源更為平均, 更適合用於加密運算中進行容錯

Previous Work 1:

在說明AN-code之前, 先來講解算術權重錯誤模型

所謂算術權重是指把一個數轉為二進制後, 1和0的個別佔比, 當一個數發生算術權重錯誤時, 其數值會發生+or- 2的冪次方的變化

舉例來說: 當數字3發生一個+21 的算術權重錯誤, 數值的二進制會從0011 🡺 0101 發生2位元的位元翻轉

而對通道碼而言, 發生錯誤僅僅只是一般的位元錯誤, 即 0 🡺 1, 1 🡺 0 的位元翻轉

舉例來說: 當數字3發生一個位元錯誤, 數值的二進制會從0011 🡺 0001 發生1位元的位元翻轉

Previous Work2:

接著介紹AN-code乘積碼: 它是透過把資料乘上一個數A以確保接下來經過運算後還能保有容錯並更正錯誤的能力

以A=13來說, 能編碼的資料為0~4, 編碼後的codeword我們稱為C, 也就是AN

假設在運算的過程中發生了一個錯誤e, 使的codeword變成C‘ = C + e, 那我們可以透過模A來確認是否有發生錯誤

若模完的結果餘數為0, 則沒有發生錯誤可直接輸出結果, 但若餘數不為0, 代表過程中有發生錯誤, 我們可以透過 Galois Field 來更正這個錯誤, 再輸出結果

值得注意的是限制於 Galois Field 的特性, 此AN-code只能解單一算術權重錯誤

Previous Work 3:

為了更進一步了解AN-code和通道碼channel code的不同, 我們針對資料經過通道傳輸後, 進行算術運算的整個過程進行分析

可以看到通道碼只有在通道傳輸時具有錯誤偵測及更正的能力, 只要資料一經過運算, 通道碼即喪失容錯效果

相反的AN-code不僅在算術運算時保有錯誤偵測及更正的能力, 它也可以取代channel code, 針對通道的資料進行編碼容錯

且也可以在channel Encoder 、channel Decoder 的地方加入AN-code, 確保通道編解碼的正確性

Proposed Technique 1:

為了能更正兩個算術權重錯誤, 我們使用了左邊這張演算法的流程圖以尋找適合的A值

首先我們先定出位元的範圍, 之後針對在範圍內的資料去乘上A值, 得到codeword: AN, 接著我們針對AN加入單一或雙重算術權重錯誤, 以產生錯誤資料, 最後再將所有的錯誤資料去模A, 得到相對應的餘數

若所有的餘數皆不相同,則這個A是可以解雙重算術權重錯誤的A, 若餘數有相同的部分, 則要回到挑選A值的部分, 重複這個動作迴圈直到找出符合要求的A值

一旦找出適合的A, 我們就可以根據不同的算術權重錯誤對應不同的餘數這個特性去建構一個1對1的查找表(LUT), 如右圖所示

這個查找表由兩個部分所組成, 一個是可以解單一算術權重錯誤的LUT, 另一個是可以解雙重算術權重錯誤的LUT。由於雙重算術權重錯誤的LUT的面積複雜度遠比單一算術權重錯誤的LUT大的多, 複雜度幾乎是N倍, 因此我們後面會提到另一個方法只需要用單一算術權重錯誤的LUT就能解雙重算術權重錯誤

Proposed Technique 2:

這兩張圖顯示了A值在不同位元數的占比, 可以看得出解兩個算術權重錯誤的A值在位元數較小的時候佔整體的codeword比例很大, 但隨著位元數的增加, A值佔比越來越小, 這代表其非常適合用來當作大整數運算的容錯, 例如RSA加密

Proposed Technique 3:

由於剛才提到,整體的LUT的面積成本過高, 我們提出了另一個折衷方法, 透過線性分解的方式, 把兩個算術權重錯誤的餘數拆成2個單一算數權重錯誤的餘數, 再搭配圖中的Pseudo code執行for-loop以找出對應的錯誤並對其進行更正

在Pseudo code中可以看到用顏色標記的兩個LUT, 一個是從錯誤的位置找出餘數, 一個是從餘數找出錯誤位置

由於l-LUT及r-LUT的大小皆是單一算術權重錯誤LUT的大小, 因此使用這兩個LUT搭配搜尋的方式, 就能節省面積複雜度N倍, 同時因為使用到for-loop搜尋時間成本會增加

Experimental Results 1:

為了證明更正雙重算術權重錯誤有更好的可靠度, 我在乘法器裡的硬體添加雜訊, 來觀察不同容錯技術的錯誤率

這兩張圖分別為18bits 及 26bits 乘法器的錯誤率模擬, 可以看到我們的更正雙重算術權重AN-code(藍色曲線)有最佳的可靠度

由於三模組冗餘 (TMR) (橘色曲線)並沒有使用解碼器來更正錯誤, 因此它的曲線屆於傳統AN-code和 未編碼的之間

Experimental Results 2:

為了比較各個方法的成本🡪即單純用查找表(Parallel)、純用搜尋法跑for-loop不使用任何LUT(Sequential) 及 跑for-loop搭配LUT(Trade-off)

我們假定總成本是一個定值, 以分析這三個方法的面積和時間成本

面積的部分我們使用Synopsys的Design Compiler搭配U18製程來進行合成, 時間的部分我們統一在Zynq7000板子上的Python平台執行程式, 結果如下表所示。

由於軟體執行所需的面積及硬體在python上的執行時間很難得知, 因此在此我們用評估的方式來做推估

Experimental Results 3:

為了更清楚觀察這三個方法的成本差異, 我們將上一頁表中的數據給量化, 得到圖中的曲線

可以看到不管是Parallel法還是Sequential法, 都會有一項成本是極為巨大的, 但Trade-off法的時間和面積成本在這張圖裡分布的極為平均,不會有過大或過小的情形, 代表Trade-off法優化了整體的運算效率及所需的資源, 使其不僅保有容錯的能力, 更使得成本分配更有效率, 更適合用在大整數的運算, 像是RSA加密當中

這是我的參考文獻

謝謝大家的聆聽