Lab2 Tutorial

RSA256 解密機

Table of Contents

1. 實驗目的

透過以硬體實作 RSA256 的解碼演算法，了解不同運算對於硬體效率的影響，進而體會硬體加速的不可取代性。而實作 RS232 的溝通架構，也能理解模組溝通的基礎模式，是為日後進一步邁入 SoC 等相關領域的重要概念建設。

2. 實驗器材

DE2-115 FPGA 板

電源線

USB 傳輸線

(1) RS232 ↔ RS232

(2) RS232 ↔ USB

RS232 傳輸線

(有兩種，只需其中一條，視需要拿取。檢查電腦主機背後是否有 RS232 接頭，沒有的話需要 RS232 ↔ USB)

3. Rsa256Core

3.1. RSA Cryptosystem

(1) RSA 通過公鑰(public key)與私鑰(private key)的形式確保資料不被第三者得知。

(2) 公鑰和私鑰

所謂「鑰」由三個數字構成：N、e、d。N 為兩個極大的質數乘積，而 e 與 d 滿足：$(e \cdot d) \mod N \equiv 1$。(N, e)為公鑰，而(N, d)為私鑰。公鑰為可為大眾取得，做加密之用；私鑰則不公開，用以解密被相對應公鑰加密的資料。更詳細有關 RSA 加密方式的推導請上網搜尋。

(3) RSA 加密：傳送方以接收方的公鑰加密

$$\text{encrypted message}(m_e) = \big(\text{message}(m)\big)^e \mod(N)$$

(4) RSA 解密：接收方以自己的私鑰解密

$$m = (m_e)^d \mod(N)$$

3.2. How to compute $y^d \mathbf{mod}(N)$

(1) Exponentiation of RSA256

y^d可視為 d 個 y 的連乘。在連乘完之後一口氣做 mod 運算不切實際(考量連乘所需要的 word length，以及 mod 對極大數字運算非常昂貴)。因此，運用 mod 的特性，可對連乘作以下的演算法改進：

首先，將 d 以 binary 表示。若假定 d = 12，

$y^d = y^{12_{10}} = y^{1100_2} = (1 \cdot y^8) \cdot (1 \cdot y^4) \cdot (0 \cdot y^2) \cdot (0 \cdot y^1)$ 。

在這裡，二進制帶來了好處: $y^8 = (y^4)^2 = ((y^2)^2)^2 = (((y)^2)^2)^2$。亦即，每個乘數之間是平方關係。由此出發，可得在餘數系統下有效率的次方算法：

Algorithm 1 RSA256 with exponentiation by squaring

```
1: function ExponentiationSquaring(N, y, d)
2:     t ← y
3:     m ← 1
4:     for i ← 0 to 255 do
5:         if i-th bit of d is 1 then
6:             m ← m · t (mod N)
7:         end if
8:         t ← t^2 (mod N)                          ▷ t → t^2 → t^4 → ···
9:     end for
10:    return m
11: end function
```

需要特別注意的是，由於在每一個 iteration 中均取 mod，因此運算結果不會如直接做次方一般有指數性的成長。這裡，我們簡單說明一下餘數運算對乘法的分配律如何使餘數運算可以拆在連乘過程中執行：

假設 $a = c \cdot N + r,\ c, N, r \in \mathbb{Z}$，則

$$((a \ mod(N)) \cdot a) \ mod(N) \dots a) \ mod(N)$$
$$= ((r) \cdot a) mod(N) \dots a) mod(N) = r^d mod(N) \equiv a^d mod(N)$$

(2) Modulo of Products

經過(1)的化簡，$y^d \bmod N$變成了多次的$ab \bmod N$。同樣的，乘法器是一個昂貴的硬體，我們希望以多次的加法代替乘法。假設 a = 12：

$ab \bmod N = 12_{10} \cdot b \bmod N = 1100_2 \cdot b \bmod N = (8b + 4b) \bmod N$。

類似(1)的想法，這裡得益於$8b = 2 \cdot 4b = 2 \cdot 2 \cdot 2b$，可得 Algorithm 2。同樣地，在每個 iteration 中，均對結果取餘數，以免有 overflow 等狀況發生。證明方式和(1)雷同，在此不贅述。

Algorithm 2 Modulo of products

```
1: function ModuloProduct(N, a, b, k)              ▷ k is number of bits of a
2:     t ← b
3:     m ← 0
4:     for i ← 0 to k do
5:         if i-th bit of a is 1 then
6:             if m + t ≥ N then
7:                 m ← m + t − N                   ▷ perform modulo operation in each iteration
8:             else
9:                 m ← m + t
10:            end if
11:        end if
12:        if t + t ≥ N then
13:            t ← t + t − N                       ▷ perform modulo operation in each iteration
14:        else
15:            t ← t + t
16:        end if
17:    end for
18:    return m
19: end function
```

(3) Montgomery Algorithm

在(2)中，為了確保不會有 overflow 發生，採用的是每個 iteration 做餘數運算，造成需要每個 iteration 均需要和 N 比較大小。然而，除了每個 iteration 均取餘數，可以有其他方法確保 overflow 不會發生，同時不需要每個 iteration 做餘數運算。

考慮以$\mathrm{ab}\cdot 2^{-\mathrm{i}}\mathrm{mod}(\mathrm{N})$代替原本的$\mathrm{ab}\ \mathrm{mod}(\mathrm{N})$。$\mathrm{ab}\cdot 2^{-\mathrm{i}}$可以被拆解成(假設 a = 12，取 i = 4)：

$\mathrm{ab}\cdot 2^{-4} = 4'\mathrm{b1100}\cdot \mathrm{b}\cdot 2^{-4} = \Big(\big((b\cdot 2^{-1}+b)\cdot 2^{-1}+0\big)\cdot 2^{-1}+0\Big)\cdot 2^{-1}$。

由於每個 iter 均乘2^{-1}，避免「累加」溢位的問題。注意：這裡的$\mathrm{b}\cdot 2^{-1}+\mathrm{b}$仍需要 5 個 bit 來避免加的溢位。為了理解$\mathrm{ab}\cdot 2^{-\mathrm{i}}\mathrm{mod}(\mathrm{N})$，先來看一下$\mathrm{i}=1$這個特殊的情況：

$$\mathrm{a}\cdot 2^{-1}\equiv \mathrm{b}\big(\mathrm{mod}(\mathrm{N})\big)\Rightarrow \mathrm{a}\equiv 2\mathrm{b}\big(\mathrm{mod}(\mathrm{N})\big)$$

b 可以簡單地被找到：

$$\mathrm{if(a\ \ is\ \ even)}\ \ b=\frac{a}{2};$$

$$\mathrm{if(a\ \ is\ \ odd)}\ \ b=\frac{a+N}{2}$$

結合(2)與(3)，可以得到計算$\mathrm{ab}\cdot 2^{-\mathrm{i}}\mathrm{mod}(\mathrm{N})$的 Montgomery Algorithm 如下：(要注意此處的 m 要是 257 bits)

Algorithm 3 Montgomery algorithm for calculating $ab2^{-256} \bmod N$

```
1:  function MontgomeryAlgorithm(N, a, b)
2:      m ← 0
3:      for i ← 0 to 255 do
4:          if i-th bit of a is 1 then
5:              m ← m + b
6:          end if                          ▷ 4~6: replace multiplication with
                                              successive addition
7:          if m is odd then
8:              m ← m + N
9:          end if
10:         m ← m/2                         ▷ 7~10: calculate the modulo of a·2^-1
                                              → Montgomery reduction
11:     end for
12:     if m ≥ N then
13:         m ← m − N
14:     end if
15:     return m
16: end function
```

不過我們在使用 Montgomery Algorithm 來做 RSA 前還要先把 y (b)乘上 2^{256} 才會是正確的答案，因此最終 RSA 的演算法如下：

Algorithm 4 RSA256 with exponentiation by squaring and Montgomery algorithm

```
function RSA256MONT(N, y, d)
    t ← ModuloProduct(N, 2^256, y, 256)
    m ← 1
    for i ← 0 to 255 do
        if i-th bit of d is 1 then
            m ← MontgomeryAlgorithm(N, m, t)
        end if
        t ← MontgomeryAlgorithm(N, t, t)
    end for
    return m
end function
```

3.3. 使用 Rsa256Core Testbench

```
ncverilog +access+r tb.sv Rsa256Core.sv
```

4. Rsa256Wrapper

4.1. Wrapper 功能說明

Wrapper 負責的工作主要有兩個：

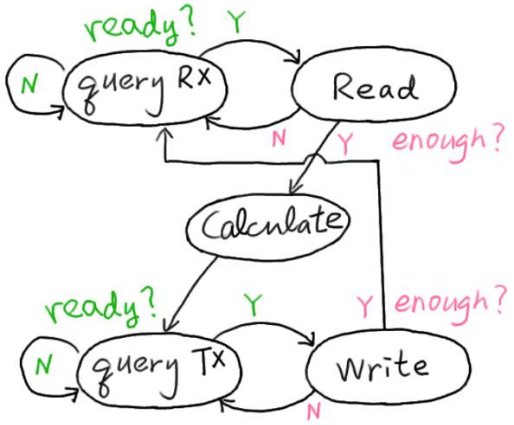
(1) 在恰當的時候讀資料 (key & encoded data)。

(2) 在恰當的時候寫資料 (decoded data)。

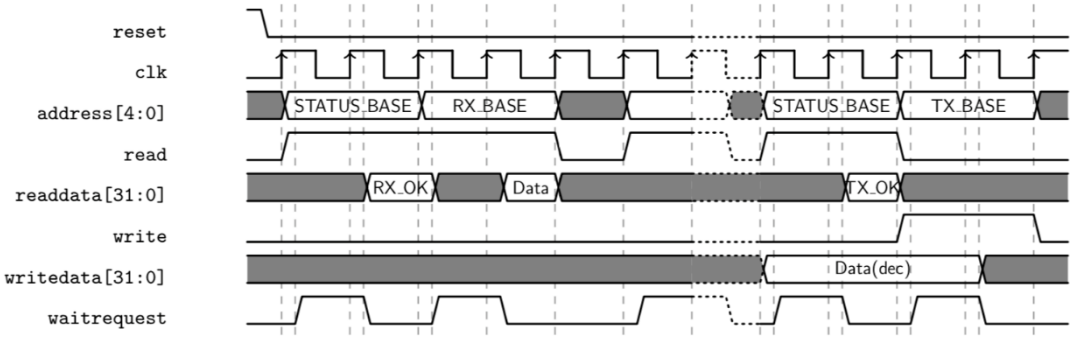
4.2. 訊號說明

Signal	Type	Description
avm_clk	I	clock
avm_rst	I	Reset
avm_read	O	當“讀”資料時，avm_read = 1, 其餘時間為 0
avm_write	O	當“寫”資料時，avm_write = 1, 其餘時間為 0
avm_address	O	讀寫資料時的資料位置
avm_waitrequest	I	當 avm_waitrequest = 0 時，才能進行讀寫動作
avm_readdata	I	讀進來的資料
avm_writedata	O	寫出去的資料

4.3. 簡易 FSM 設計



4.4. 波形說明



注意事項：觀察上方的波形圖，可以發現不論讀資料或者寫資料，address 都要先回歸 STATUS_BASE，再視情況將 avm_read 或 avm_write 輸出 1。

4.5. 解密機運作流程

(1) Query RX & Read data

本次實驗中，送資料的順序為先送 key (其中先送 N 再送 e)，再送 encoded data。綜合上述說明，當 avm_waitrequest = 0 且 RX_OK = 1 (即 avm_readdata[RX_OK] = 1，可參照下方 RS-232 Qsys 的說明)時即可將 address 調到 RX_BASE，並開始讀資料，每次讀資料時都要重複上述的步驟。此外，每次讀資料僅能讀取 8 bits = 1 byte (即

avm_readdata[7:0])，而不論 key(N, e)或 encoded data 都是 256 bits (32 bytes)，因此各自需要 32 次讀資料的過程。

(2) Calculation

Rsa256Core.sv 在這個階段藉由拿到的 key 和 encoded data 進行運算。

(3) Query TX & Write data

得到 RsaCore256.sv 計算完成後的 decoded data，即開始寫資料的過程。和讀資料的程序相似，在每次讀資料前 address 都要先回歸 STATUS_BASE，當 avm_waitrequest = 0 且 TX_OK = 1 (即 avm_readdata[TX_OK] = 1)時即可將 address 調到 TX_BASE，並開始寫資料，每次寫資料時都要重複上述的步驟。此外，每次送出去的資料同樣都是 8 bits，decoded data 總共有 248 個 bits，因此需要 31 次寫資料的過程。值得注意的是，寫資料的順序為從第 247 bit 開始直到第 0 個 bit 結束。例如以下寫法：

```
assign avm_writedata = dec_r[247:240]
```

4.6. 參考資料 (RS232 Qsys Module)

Offset	Register Name	R/W	Description/Register Bits													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	Reserved					1	1	Receive Data						
1	txdata	WO	Reserved					1	1	Transmit Data						
2	status 2	RW	Reserved	eop	cts	dcts	1	e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	Reserved	ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor 3	RW	Baud Rate Divisor													
5	endof-packet 3	RW	Reserved					1	1	End-of-Packet Value						

4.7. 使用 Rsa256Wrapper Testbench

```
ncverilog +access+r test_wrapper.sv PipelineCtrl.v \
PipelineTb.v Rsa256Wrapper.sv Rsa256Core.sv
```

5. 其他建議

燒到板子上前請善用 testbench，切勿直接在板子上 debug，會崩潰。