Indian Institute of Technology Kharagpur

SPRING Semester, 2023
COMPUTER SCIENCE AND ENGINEERING

CS60004: Hardware Security

Mid Semester Examination

Full Marks: 50

Time allowed: 2 hour

1. Consider two polynomials $A = \sum_{i=0}^{n-1} a_i x^i$ and $B = \sum_{i=0}^{n-1} b_i x^i$ in $GF(2^n)$ where n is even. The classical Karatsuba multiplication algorithm computes AB by splitting the polynomials in two halves as discussed in the class.

   (a) Assume the delay of an XOR gate is 1 TX. Compute the critical delay (in terms of TX) incurred in computing the partial products in classical Karatsuba algorithm. (5 marks)

   (b) Consider a variant of the Karatsuba algorithm wherein the polynomials are split into even and odd terms given as follows:

$$A = \sum_{i=0}^{n-1} a_i x^i = \sum_{i=0}^{n/2-1} a_{2i} x^{2i} + \sum_{i=0}^{n/2-1} a_{2i+1} x^{2i+1} = \sum_{i=0}^{n/2-1} a_{2i} x^{2i} + x \sum_{i=0}^{n/2-1} a_{2i+1} x^{2i}$$

$$B = \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{n/2-1} b_{2i} x^{2i} + \sum_{i=0}^{n/2-1} b_{2i+1} x^{2i+1} = \sum_{i=0}^{n/2-1} b_{2i} x^{2i} + x \sum_{i=0}^{n/2-1} b_{2i+1} x^{2i}$$

   Now let $y = x^2$, then $A_e(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i$, $A_o(y) = \sum_{i=0}^{n/2-1} a_{2i+1} y^i$ and $B_e(y)$ and $B_o(y)$ can be defined similarly. Thus, polynomials can be represented as $A = A_e(y) + x.A_o(y)$ and $B = B_e(y) + x.B_o(y)$. Derive the partial product terms involved in the computation of AB using the above equations. (5 marks)

   (c) Justify why this is more efficient compared to classical Karatsuba algorithm. Compute the critical delay (in terms of TX) incurred in computing the partial products in this case. (5 marks)

2. Inverse computation of an element $a \in GF(2^m)$ can be accelerated using a parallel *Itoh Tsujii* implementation. A parallel version of the *Itoh Tsujii* algorithm involves simultaneous computation of quad and quad-root operations. A quad operation raises an input $a \in GF(2^m)$ to a^4 whereas quad-root operation raises a to a^{-4}. For $q, r, i \in N$ such that $4q + r = m - 1$, using the expressions for $\alpha_i(a)$ and $\gamma_i(a)$

$$\alpha_i(a) = a^{4^i - 1} \qquad \gamma_i(a) = a^{1 - 4^i}$$

   Prove:

$$a^{-1} = \begin{cases} \{\alpha_{\frac{m-1}{4}}(a)\}^2 \cdot \gamma_{\frac{m-1}{4}}(a) & \text{if } 4 \mid (m-1); \\ \{\alpha_q(a)\}^{2^{r+1}} \cdot \gamma_q(a) \cdot a^{2^{r+1}-2} & \text{if } 4 \nmid (m-1). \end{cases}$$

   (10 marks)

3. Assuming the formulation in Question 2 for computing the inverse in $GF(2^m)$, answer the following questions for $m = 193$ for which the addition chain is [1, 2, 3, 6, 12, 24, 48, 96, 192]:

   (a) Write down the steps to calculate the inverse using the classical squaring-based Itoh-Tsujii Algorithm as studied in the class. (3 marks)

(b) Rewrite the steps to calculate the inverse using parallel quad and quad-root Itoh-Tsujii Algorithm. (3 marks)

(c) Considering there are Quad and Quad-root blocks in the hardware architecture for implementing Quad and Quad-root parallel Itoh-Tsujii Algorithm, give an estimate of the number of clock cycles required to compute the modified Itoh-Tsujii Algorithm. Compare it with the clock cycle requirement for the classical squaring-based Itoh-Tsujii Algorithm. (4 marks)

4. Solving exponentiation $s = y^x$ can be done with fewer calculations than simply multiplying the base number over and over again. That trick is known as the Square and Multiply method. In class, we have seen how to calculate exponentiation starting from the MSB. Consider the following code for the computation of exponentiation starting from LSB. In the function given below, y is the base, x is the exponent, s stores the intermediate values that lead to the final result. Since we are starting from LSB, the initial values for $from = 0$ and $to = w - 1$, where w is the length of binary expansion of x.

```
double power(int y, int x, int from, int to)
{int i, bit; long double s = 1;
for(i=from; i<=to; ++i)
{
    bit = (x >> i) & 1;
    if (bit == 1)
    { ....(i)....;}
    ....(ii)....;
}
return s; }
```

In this context, answer the following:

(a) Fill in the blanks to complete the code. (5 marks)

(b) Consider the above algorithm for computing exponentiation $s = y^x$. Our objective is to determine the $(b+1)^{th}$ bit using Kocher's timing attack assuming we know the last b bits of the exponent x. You are also given the timing values of the 10 exponentiations with different values of the base y. Use the following information to determine whether the $(b+1)^{th}$ bit is 0 or 1.

Table 1: Timing Information

y	t_{actual}	t_{guess0}	t_{guess1}
9098	416	227	237 ✓
28609	440	249	261 ✓
18976	416	261	227
16425	446	273	267
28994	649	243	395 ✓
27744	412	301	331 ✓
28744	446	331	289
19096	483	331	257
13353	516	264	349 ✓
15767	398	328	352 ✓

Note: In Table 1, y represents 10 different values for exponentiation, t_{actual} represents the actual time to calculate the complete exponentiation, t_{guess0} and t_{guess1} represent the time to calculate the partial exponentiations for $(b+1)$ bits with $(b+1)^{th}$ bit to be 0 and 1 respectively. (10 marks)