6

Division and Inversion Over Finite Fields

Abdulah Abdulah Zadeh
Memorial University of Newfoundland, Canada

1. Introduction

Arithmetic operation such as addition, multiplication, division and inversion are widely used in data communication systems, coding and cryptography particularly public key cryptography.

Since 1976, when the principles of public key cryptography were introduced (by Whitfield Diffie and Martin Hellman) (Diffie & Hellman 1976), RSA was the most well-known public key cryptographic system. Rivest, Shamir and Adleman (RSA) algorithm composes a public key considered sufficiently long enough to be recognized as secure. The security of RSA is based on difficulty of factoring large numbers to its prime components. For many years, RSA was the leading method for industrial encryption. RSA cryptographic algorithm includes addition, squaring and multiplication operations. Addition and squaring are two simple operations over finite fields; hence, the most important arithmetic operation for RSA based cryptographic systems is multiplication.

With the advances of computer computational power, RSA is becoming more and more vulnerable. In 1985, Victor S. Miller (Miller 1985) and Neal Koblitz (Koblitz 1987) proposed Elliptic Curve Cryptography (ECC), independently. ECC offer higher security in compare with RSA.

The security of ECC relies on the difficulty of solving Elliptic Curve Discrete Logarithm Problem or ECDLP. So far not any efficient method has been offered to solve ECDLP and its complexity is higher than factoring large numbers to its prime components (where the security of RSA relies on that). Hence, ECC can offer higher security with smaller key size and designers can use it to save storage space, consumed power in the circuit and increase the bandwidth.

Elliptic Curve Cryptographic algorithm includes addition, squaring, multiplication and division (or inversion). Many research and studies have been done on multiplication. However, division and inversion research are becoming more relevant to cryptographic systems. In the terms of implementation area, complexity and executing time; division (or inversion) is the most costly operation in public key cryptography. For many years hardware implementations of division or inversion were an ambitious goal. However, recent advances in technology of ASIC circuits and the ability to provide high capacity FPGAs, let circuit designers to achieve this goal.

In this chapter we study two main classes of proposed algorithms for division (and inversion). The first class of dividers is based on Fermat's little theorem. This class of dividers also called as multiplicative based dividers. In the next chapter we introduce the principles of these algorithms and the proposed methods to improve their efficiency.

Chapter three is about the other class of dividers, called Euclidian based dividers. We review the principles and all proposed algorithms based on Euclidian algorithm.

2. Dividers based on Fermat's little theorem

The most simple and primary dividers were based on Fermat's little theorem. These kinds of dividers are also known as multiplicative based dividers, because in these algorithms, division is performed by sequence of multiplication operations (and squaring). Squaring in finite fields are simple operations, which are usually perform in a simple clock cycle. However multiplication is more complicated operation and in terms of time and implementation area is more costly.

Based on Fermat's little theorem, if P is a prime number for any integer a, we can write:

$$a^p \equiv a \ (mod\ P)$$

Dividing two side to a, we get

$$a^{P-1} \equiv 1 \ (mod\ P) \quad \text{or} \quad a \times a^{P-2} \equiv 1 \ (mod\ P)$$

Hence we can conclude the inversion of any integer a over $GF(P)$ is a^{P-2}.

Example.1: For example inversion of 4 over $GF(7)$ is $4^{-1} \equiv 4^5 \equiv 2 \ (mod\ 7)$.

$$2 \times 4 \equiv 8 \equiv 1 \ (mod\ 7)$$

Expanding this technique to $GF(2^m)$, we can write

$$a^{2^m-1} = a \times a^{2^m-2} = 1 \ (over\ GF(2^m)).$$

Hence, $a^{-1} = a^{2^m-2}$, in which $a \in GF(2^m)$.

To compute a^{2^m-2}, the most primary method is "square and multiplication" algorithm. In square and multiplication algorithm instead of $2^m - 3$ multiplications, we calculate a^{2^m-2}, with at most $m - 1$ squaring and $m - 2$ multiplications.

Alg.1: *Square and Multiplication Algorithm*

Input $a \in GF(2^m)$

Output $A = a^{2^m-2}$

1. b=$2^m - 2$
2. $A = a$
3. while $b \neq 1$
   - 3.1. if (b is even)
     - 3.1.1. $b = b/2$
     - 3.1.2. $A = A \times A$
   - 3.2. else
     - 3.2.1. $b = b - 1$
     - 3.2.2. $A = A \times a$
4. Return A

To better understand of square and multiplication algorithms, we review the following equations. As we know, we can decompose 2^{m-2} in the following form.

$$2^m - 2 = 2(2^{m-1} - 1)$$

$$= 2(2^{m-1} - 2 + 1)$$

$$= 2(1 + 2(2^{m-2} - 1))$$

$$\vdots$$

$$= 2(1 + 2(1 + 2(1+..)))$$

Hence, we can use the above equations to decompose a^{2^m-2} to:

$$a^{2^m-2} = a^{2(2^{m-1}-1)} = (a^{(2^{m-1}-1)})^2$$

$$= (a^{(2^{m-1}-2+1)})^2 = (a \times a^{(2^{m-1}-2)})^2$$

$$(a \times a^{2(2^{m-2}-1)})^2 = (a \times (a^{(2^{m-2}-1)})^2)^2$$

$$\vdots$$

$$= (a(a(\dots a(aa^2)^2 \dots)^2)^2)^2$$

The square and multiplication algorithm use the same principle to calculate a^{2^m-2}.

2.1 Itoh and Tsujii algorithm

Itoh and Tsujii (Itoh & Tsujii 1988) offered a more efficient algorithm over normal basis; however it is applicable over polynomial and other basis. Their algorithm was based on multiplication which can be applied on some values of m. In their algorithm, they reduced the number of multiplications, significantly. Many efforts have been done to improve Itoh and Tsujii algorithm and make it more general for all values of m (Guajardo & C. Paar 2002; Henrıquez, et. al. 2007). Here we review the general form of this algorithm.

To describe Itoh and Tsujii algorithm, we introduce a new term, called addition chain.

Definition *addition chain*: Addition chain for an integer value such as $m-1$, is a series of integers with t elements such that, $u_0 = 1$ and $u_t = m - 1$, and $u_i = u_{ki} + u_{ji}$.

Where ki and ji are two integer values between 0 and i.

Example.2: If $m = 193$, then the addition chain could be

1, 2, 3, 6, 12, 24, 48, 96, 192

In this addition chain for all elements of sequence we have $u_i = u_{i-1} + u_{i-1}$ except for u_2, which $u_2 = u_1 + u_0$.

u_0	u_1	u_2	u_3	u_4
	$u_0 + u_0$	$u_1 + u_0$	$u_2 + u_2$	$u_3 + u_3$
1	2	3	6	12

u_5	u_6	u_7	u_8
$u_4 + u_4$	$u_5 + u_5$	$u_6 + u_6$	$u_7 + u_7$
24	48	96	192

Let's define a function $\beta_k(a) = a^{2^k-1}$, which $a \in GF(2^m)$. We know that $\beta_m(a) = a^{2^m-1} = a^{-1}$. The other characteristic of this function is enlisted as follow:

$$\beta_{j+k} = \beta_k^{2^j} \times \beta_j$$

$$\beta_{2k} = \beta_k^{2^k+1} \text{ or } \beta_k^{2^k} \times \beta_k$$

Hence, to compute a^{-1}, we should use the equations above and using addition chaining to achieve $\beta_m(a) = a^{2^m-1}$.

Example.3: for $m = 193$, and above addition chain, we can write the following calculations

$u_0 = 1$	$\beta_1 = a^{2^1-1}$
$u_1 = 2$	$\beta_2 = (\beta_1)^{2^2-1} = a^{2^2-1}$
$u_2 = 3$	$\beta_3 = (\beta_2)^{2^1-1} \times \beta_1$
$u_3 = 6$	$\beta_6 = (\beta_3)^{2^3} \times \beta_3$
$u_3 = 12$	$\beta_{12} = (\beta_6)^{2^6} \times \beta_6$
$u_3 = 24$	$\beta_{24} = (\beta_{12})^{2^{12}} \times \beta_{12}$
$u_4 = 48$	$\beta_{48} = (\beta_{24})^{2^{24}} \times \beta_{24}$
$u_4 = 96$	$\beta_{96} = (\beta_{48})^{2^{48}} \times \beta_{48}$
$u_4 = 192$	$\beta_{192} = (\beta_{96})^{2^{96}} \times \beta_{96}$

It has been shown that the maximum number of multiplication in this method is t and the required number of square operation is $m-1$. The size of addition chain or t is estimated as $\lfloor log_2(m-1) \rfloor + HW(m-1) + 1$, where $HW(m-1)$ is the hamming weight of $m-1$.

For more information and more details, the readers may refer to (Guajardo & C. Paar 2002; Henrıquez, et. al. 2007).

Itoh and Tsujii algorithm is presented in Alg.2.

After calculating inversion, division simply becomes a multiplication operation.

The advantage of Fermat's little theorem based inversion algorithm is that, it can be implemented just by using multiplication and square arithmetic operators. This eliminates the need to add any extra components, such as dividers. When ECC was proposed, the

dividers were not as advanced as they are now; hence, multiplicative based dividers were the best candidates for hardware implementation of ECC, particularly over FPGAs. Also it is possible to use these dividers for reconfigurable cryptosystems, which are designed to perform both RSA and ECC algorithms. Since the sizes of these cryptosystems are becoming larger, dropping a big component such as divider is a huge saving on implemented area for designers. The main drawback of the cipher cores without dividers is the longer computational time.

Alg.2: *Itoh and Tsujii Algorithm to compute inversion*

Input $a \in GF(2^m)$

Output a^{-1}

1. $\beta_{u0}(a) = a$
2. For $i = 1$ to t do
   2.1. $\beta_{ui}(a) = \left(\beta_{uki}(a)\right)^{2^{uji}} \times \beta_{uji}(a)$
3. Return $\beta_{ut}^2(a)$

3. Euclidian based dividers

Euclid's algorithm is an old algorithm to calculate the greatest common divider (GCD) of two integers. The basic principle of Euclid's algorithm is that, the greatest common divider of a and b, $GCD(a,b)$, is equal to the greatest common divider of a and $a \pm b$ or in other word $GCD(a,b) = GCD(a, a \pm b) = GCD(a \pm b, b)$.

Example.4: $GCD(18,30) = 6$,

$$GCD(18,30) =$$

$$GCD(30 - 18,18) = GCD(12,18) =$$

$$GCD(18 - 12,12) = GCD(6,12) =$$

$$GCD(12 - 6,6) = GCD(6,6) = 6$$

We can apply the above principle more than once and rewrite this theorem as $GCD(a,b) = GCD(a, n \times a \pm m \times b) = GCD(\acute{n} \times a \pm \acute{m} \times b, b)$.

Example.5: GCD(90,525)=15

$$GCD(90,525 - 5 \times 90) = GCD(90,525 - 450) =$$

$$GCD(90,75) = GCD(75,90 - 75) =$$

$$GCD(75,15) = GCD(15,75 - 3 \times 15) =$$

$$GCD(15,0) = 15$$

To reduce the calculation time, we can offer the Alg.3.

Alg.3: *Euclidian algorithm to calculate Greatest Common Divider (GCD)*

Input a, b

Output $GCD(a, b)$

1. While $(b \neq 0)$
   1.1. $t = b$
   1.2. $b = a \bmod b$
   1.3. $a = t$
2. Return (a)

The above algorithm can be made more compact using a recursive approach. Alg. 4 presents the recursive and more compact version of Alg. 3.

Alg.4: *Euclidian algorithm to calculate Greatest Common Divider (Recursive Approach)*

Input a, b

Output $GCD(a, b)$

1. if $(b = 0)$
   1.1. Return (a)
2. else
   2.1. Return $(GCD(a, a \bmod b))$

We provide a useful theorem below which will be used this section, to make the Euclidian algorithm more general for our purpose.

Theorem: let's assume $b = a \times q + r$. Then $GCD(a, b) = GCD(a, r)$

$$GCD(a, b) = GCD(a, b - a \times q)$$

$$= GCD(a, r)$$

The simple proof for this theorem is by applying Euclid's theorem ($GCD(a, b) = GCD(a, b - a)$) for q times, to give the same relationship.

In order to use Euclid's theorem for division or inversion, assume two values such as a and b. We have already seen how to compute $d = GCD(a, b)$. We know that there are two variables, x and y, which satisfies the following equation

$$a \times x + b \times y = d$$

If we can design an algorithm which accepts a and b, and produces x and y; we can use that algorithm to find inversion. Assume P is a prime value and a is an integer where $0 < a < P - 1$. We know $v = GCD(a, P) = 1$. Hence, applying the above algorithm, we can find x and y which $a \times x + P \times y = 1$.

If we use that algorithm over the finite field, $GF(P)$, we can calculate the inverse of a which is x (i.e. $a^{-1} = x$). Using the algorithm above, it gives us x and y such that it satisfy the equation: $a \times x + P \times y = 1$. Over the finite field, $GF(P)$, $P \times y = 0$. Then $a \times x + P \times y = 1$ over $GF(P)$ could be simplified to $a \times x = 1$. Then x is the inversion of a over $GF(P)$.

Let's $GCD(a_i, b_i) = d$. We know there are two integer values, x_i and y_i such that (where one of the values is smaller than zero):

$$a_i \times x_i + b_i \times y_i = d.$$

Based on Euclid's theorem, we can write $GCD(a_i, b_i - a_i q_i) = d$. Hence, the equation above can be rewritten as:

$$a_i \times x_{i+1} + (b_i - a_i q_i) \times y_{i+1} = d.$$

By rearranging this equation, we can write:

$$a_i \times x_{i+1} - a_i q_i \times y_{i+1} + b_i \times y_{i+1} =$$

$$a_i \times (x_{i+1} - q_i \times y_{i+1}) + b_i \times y_{i+1} = d$$

Then we can conclude:

$$\begin{aligned} x_i &= x_{i+1} - q_i \times y_{i+1} \\ y_i &= y_{i+1}. \end{aligned} \tag{1}$$

Similarly, for $GCD(a_i - b_i q_i, b_i) = d$, we can write the same equations and conclude

$$\begin{aligned} y_i &= y_{i+1} - q_i \times x_{i+1} \\ x_i &= x_{i+1}. \end{aligned} \tag{2}$$

If we perform the Euclidian algorithm to calculate d, at the final step or loop $GCD(a_n, b_n) = GCD(a_n, a_n q_n) = a_n = d$. The above relationship for this step will be

$$a_n \times x_n + b_n \times y_n =$$

$$a_n \times x_n + a_n q_n \times y_n = a_n = d$$

So $x_n = 1$ and $y_n = 0$.

Example.6: Let's $a = 37$ and $b = 17$

$$37x_0 + 17y_0 = 1$$

$$(37 - 2 \times 17)x_0 + 17y_0 = 1 \qquad q_0 = 2$$

$$3x_1 + 17y_1 = 1$$

$$3x_1 + (17 - 5 \times 3)y_1 = 1 \qquad q_1 = 5$$

$$3x_2 + 2y_2 = 1$$

$$(3 - 1 \times 2)x_2 + 2y_2 = 1 \qquad q_2 = 1$$

$$x_3 + 2y_3 = 1$$

$$x_3 + (2 - 2 \times 1)y_3 = 1 \qquad q_3 = 2$$

$$x_4 = 1$$

Using (1) and (2) for the above relation in backward (start from x_4, y_4 and q_3), we can calculate x_0 and y_0.

$$y_3 = y_4 = 0 \qquad x_3 = x_4 - q_3 y_4 = 1$$

$$x_2 = x_3 = 1 \qquad y_2 = y_3 - q_2 x_3 = -1$$

$$y_1 = y_2 = -1 \qquad x_1 = x_2 - q_1 y_2 = 6$$

$$x_0 = x_1 = 6 \qquad y_0 = y_1 - q_0 x_1 = -13$$

Then finally:

$$37 \times 6 + 17 \times (-13) = 1$$

Hence, one way of finding x and y is to execute Euclidian algorithm. Then calculate x_i and y_i based on the equations above. Alg.5 is based on this idea.

Alg.5: *Algorithm of Finding x and y*

Input: $a, b \quad (b \geq a)$

Output: $GCD(a, b), x, y$

1. $y_1 = 1$
2. $y_2 = 0$
3. $x_1 = 1$
4. $x_2 = 0$
5. While $(a \neq 1)$
   5.1. $q = \left\lfloor \frac{b}{a} \right\rfloor;\ r = b - qa;\ x = x_2 - qx_1;\ y = y_2 - qy_1;$
   5.2. $b = a;\ a = r;\ x_2 = x_1;\ x_1 = x;\ y_2 = y_1;\ y_1 = y;$
6. $d = b;$
7. $x = x_2;$
8. $y = y_2;$
9. Return (d, x, y)

In order to get better impression about the role of x_1, x_2, y_1 and y_2 in Alg.5 (and Alg.6) we recommend to extend the last two equations of example.6 (i.e. y_0 and x_0) and rewrite them with q_i, y_4 and x_4.

All the substitutions at step 5.1 and 5.2 of Alg.5 should be executed at the same time.

We can simplify this algorithm for a and P (where $0 \leq a < P$, and P is a prime number) to calculate a^{-1} over $GF(P)$ (Alg.6).

Alg.6: *Algorithm of Computing Inversion Over $GF(P)$*

Input: $P, a \in GF(P)$

Output: a^{-1}

1. $y_1 = 1$
2. $y_2 = 0$
3. While $(a \neq 1)$
   3.1. $q = \left\lfloor \frac{P}{a} \right\rfloor$
   3.2. $a = P - qa;\ P = a;\ y_2 = y_1; y_1 = y_2 - qy_1$
4. Return (y_1)

All the operations on Alg.6 are performs over $GF(P)$. All the substitutions at step 3.2 of Alg.6 should be done simultaneously.

In the algorithm above, we should perform a division at each loop (step 3.1.). To avoid division, we can assume if $P \geq a$ then q =1 and if $P < a$ then $q = 0$ or swap a and P and y_1 and y_2 values. Then we can compute $GCD(a, b - a)$, instead of computing $GCD(a, b) = GCD(a, b - aq)$. This technique increases the number of iterations.

Modifying the above algorithms for polynomial basis, we have Alg.7. All operations in Alg.7 should be done over $GF(2^m)$. In Alg.7, P represents the irreducible polynomial of $GF(2^m)$.

Alg.7: *Algorithm of Computing Inversion Over $GF(2^m)$*

Input: $a \in GF(2^m)$

Output: a^{-1}

1. $y_1 = 1$
2. $y_2 = 0$
3. While $(a \neq 1)$
   3.1. $a = P + a;\ P = a;\ y_2 = y_1; y_1 = y_2 + y_1$
4. Return (y_1)

Example.7: let's assume we want to calculate 1/7 over $GF(17)$

$$y_1 = 1 \quad y2 = 0 \quad a = 7 \quad P = 17$$

$$y_1 = -1 \quad y2 = 1 \quad a = 10 \quad P = 7$$

$$y_1 = 1 \quad y2 = -1 \quad a = 7 \quad P = 10$$

$$y_1 = -2 \quad y2 = 1 \quad a = 3 \quad P = 7$$

$$y_1 = 3 \quad y2 = -2 \quad a = 4 \quad P = 3$$

$$y_1 = -2 \quad y2 = 3 \quad a = 3 \quad P = 4$$

$$y_1 = 5 \quad y2 = -2 \quad a = 1 \quad P = 3$$

Then $7^{-1} = 5$ over $GF(17)$.

The reviewed algorithm, so far, calculates inversion. After an inversion is calculated, simply multiply y_1 to create a division. In (Takagi 1998), N. Takagi offered an algorithm which directly calculates division.

This algorithm is based on two concepts:

(1) If a is even and P is odd, then $GCD(a, P) = GCD(a/2, P)$;

(2) If both a and P are odd, then $GCD(a, P) = GCD((a - P)/2, a) = GCD(\frac{a-P}{2}, P)$; Where in the proposed algorithm, we choose the minimum of a and P (i.e. $GCD(a, P) = GCD((a - P)/2, \min\{a, P\})$).

The proposed algorithm over $GF(P)$ is presented as Alg.8. In Alg.8, b_0 represents the least significant bit (LSB) of b. Also all operation are performed over $GF(P)$.

Alg.8: *Algorithm of Computing Division Over $GF(P)$*

Input: $P, a \in GF(P), b \in GF(P)$

Output: a/b

1. $v = 0$
2. While $(b > 0)$
   - 2.1. While $(b_0 = 0)$
     - 2.1.1. $b = b/2$; $a = a/2$;
   - 2.2. If $(b \geq P)$
     - 2.2.1. $b = b - P$; $a = a - v$;
   - 2.3. else
     - 2.3.1. $b = P - b$; $P = b$;
     - 2.3.2. $a = v - a$; $v = a$;
3. Return (v)

P values decrease at each step. At the final step, b and P are zero and one, respectively. This algorithm will finish at most after $2m-1$ iterations, where $2^{m-1} < P < 2^m$.

Alg.9: *Algorithm of Computing Division Over* $GF(2^m)$

Input: $P(x), a \in GF(2^m), b \in GF(2^m)$

Output: a/b

1. $v = 0$;
2. While $((a \neq 0)\ and\ (P \neq 1))$
   2.1. If $(b_0 = 1)$
      2.1.1. If $(b \geq P)$
         2.1.1.1. $b = b + P;\ a = a + v$;
      2.1.2. else
         2.1.2.1. $b = P + b;\ P = b$;
         2.1.2.2. $a = v + a;\ v = a$;
   2.2. $b = b/2$;
   2.3. $a = a/2$;
3. Return (v)

To extend this algorithm to be applicable over $GF(2^m)$, the following changes should be applied; Assume $P(x)$ as irreducible polynomial (It is known that P_0 is always 1) and substitute $P(x)$ with P. The degrees of the most significant nonzero bit of $b(x)$ and $P(x)$ will distinguish which variable is larger (in step 2.2). Hence, the algorithm will be as Alg.9.

Alg.10: *Modified Algorithm of Computing Division Over* $GF(2^m)$

Input: $P(x), a \in GF(2^m), b \in GF(2^m)$

Output: a/b

1. $v = 0$;
2. While $((a \neq 0)\ and\ (P \neq 1))$
   2.1. If $(b_0 = 1)$
      2.1.1. If $(\delta < 0)$
         2.1.1.1. $b = b + P;\ P = b$;
         2.1.1.2. $a = a + v;\ v = a$;
         2.1.1.3. $\delta = -\delta$;
      2.1.2. else
         2.1.2.1. $b = P + b$;
         2.1.2.2. $a = v + a$;
   2.2. $b = b/2$;
   2.3. $a = a/2$;
   2.4. $\delta = \delta - 1$;
3. Return (v)

This algorithm, takes at most $2m-1$ iterations to finish. Checking the degree of b and P, is a costly operation in hardware implementation. In (Brent & Kung 1983), Brent and Kung reduced this complexity by adopting a new idea. They used a new variable, δ, to represent the difference of upper bounds of degree b and $P(x)$. In (Brent & Kung 1983) they use this method to calculate the Greatest Common Divisor of two variables. However this method can be used to calculate division.

At the initialization step, δ should be equal to -1. Then the above algorithm has to be changed as Alg.10.

Example.8: Let's $a = 1101$, $b = 0111$ and the irreducible polynomial is $P(x) = x^4 + x + 1$.

$$\delta = -1 \quad b = 0111 \quad P = 1\,0011 \quad a = 1101 \quad v = 0$$

$$\delta = 0 \quad b = 1010 \quad P = 0\,0111 \quad a = 1111 \quad v = 1101$$

$$\delta = -1 \quad b = 0101 \quad P = 0\,0111 \quad a = 1110 \quad v = 1101$$

$$\delta = 0 \quad b = 0001 \quad P = 0\,0101 \quad a = 1000 \quad v = 1110$$

$$\delta = -1 \quad b = 0010 \quad P = 0\,0101 \quad a = 0011 \quad v = 1110$$

$$\delta = -2 \quad b = 0001 \quad P = 0\,0101 \quad a = 1000 \quad v = 1110$$

$$\delta = -1 \quad b = 0010 \quad P = 0\,0001 \quad a = 0011 \quad v = 1000$$

The final step to improve the algorithm above is applied within the loop. Hardware implementation of "$while$" statement is difficult. This is because the number of iterations is an unknown variable, making it inappropriate for cryptographic cores and particularly systolic implementations. We know that this algorithm takes at most $2m-1$ iterations. Hence, instead of a "$while$" loop, we implement a "for" loop. This modification can be done by a simple change in Alg.10. In step.2, instead of "While $((a \neq 0)\ and\ (P \neq 1))$" we should write "For $i = 1$ to $2m-1$".

So far we have presented very general forms of divider algorithms. We reviewed all the proposed algorithms because each one has a unique characteristic that makes it more efficient for a specific design of a core. Many research papers have been done to improve the above algorithms and make them more efficient for hardware implementations. For example, in (Wu, Shieh & Hwang 2001), the designers proposed a new algorithm. In their algorithms, they eliminate δ and use two other variables to Instead of comparing δ relationship to zero, they only check two bits of their new adopted variables in their algorithm; thus making the new algorithms more efficient for hardware (by eliminating step 2.1.1 in Alg.10). Another example can be seen in (Zadeh 2007), where the number of iterations is reduced from $2m-1$ to m by combining two loop iterations. The paper explores how a number of modifications can reduce the number of conditional statements.

Other similar classes of dividers have been proposed such as Dual Field Modular dividers or Unified Modular Division (UMD). These classes perform division on two finite field (over $GF(P)$ and $GF(2^m)$). Unified Modular Dividers have been applied in some applications such as network servers (Wolkerstorfer 2002; Tenca & Tawalbeh 2004).

Euclidian algorithm is the most efficient algorithm for division in terms of area and time. Until now, not many hardware platforms were able to implement this algorithm. Advances in technology of ASIC offer many high capacity reconfigurable platforms such as FPGA. It gives hardware designers the ability of using these dividers in real applications. It is foreseeable that Euclidian dividers will be more widely implemented in the future.

4. Conclusion

In this chapter, we have reviewed two common classes of dividers which are widely used for cryptographic purpose. The most common dividers to be implemented in Elliptic Curve Cryptography and other cryptographic cores are multiplicative based dividers (based on Fermat's little theorem) and Euclidian based dividers.

To perform division over finite field, some other dividers have been proposed such as "Wiener-Hopf equation" based dividers. In Wiener-Hopf based dividers, the divisor (b) should expand to an $m \times m$ matrix, B, then the linear equation $B \times v = a$ should be solved to get v. v can be calculated using Gaussian elimination algorithm (Morii, Kasahara & Whiting 1989; Hasan & Bhargava 1992). The hardware efficiency of these dividers are not comparable with multiplicative and Euclidian based dividers.

In terms of implementation area multiplicative based dividers are very efficient. Since they don't need any extra component on the circuit and they can perform division using embedded components of the cipher cores. Also in term of speed, Euclidian based dividers are very fast.

5. References

Brent R. P., Kung H. T., (Aug. 1983), "Systolic VLSI arrays for linear time GCD computation", in VLSI-83, pp: 145—154, Amsterdam.

Chen C., Qin Z., (June 2011), "Efficient algorithm and systolic architecture for modular division", International Journal of Electronics, vol. 98, No. 6, pp: 813—823.

Diffie W., Hellman M. E., (Nov. 1976), "New directions in cryptography", IEEE Transactions on Information Theory, vol. IT-22, pp: 644–654.

Dormale G. M. D., Quisquater J. , (2006), "Iterative modular division over $GF(2^m)$: novel algorithm and implementations on FPGA", Applied Reconfigurable Computing – ARC 2006, pp: 370—382.

Guajardo Jorge, Paar Christof, (2002), "Itoh-Tsujii inversion in standard basis and its application in cryptography and codes", Designs, Codes and Cryptography, vol. 25, pp: 207—216.

Hankerson, Darrel, Menezes, Alfred J., Vanstone, Scott, (2004), "Guide to elliptic curve cryptography", Springer-Verlag, ISBN: 978 0 387 95273 4.

Hasan M.A., Bhargava V.K., (Aug. 1992), "Bit-serial systolic divider and multiplier for finite fields $GF(2^m)$", IEEE Transaction on Computers, vol. 41, No. 8, pp: 972—980.

Itoh T., Tsujii S., (1988), "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal basis", Information and computing, vol. 78, pp: 171-177.

Kim Chang Hoon, Hong Chun Pyo, (July 2002), "High speed division architecture for $GF(2^m)$", Electronics Letters, vol. 38, No.15, pp: 835—836.

Koblitz N., (1987), "Elliptic curve cryptosystems", Mathematics of Computation, vol. 48, pp: 203–209.

Miller V. S., (1985), "Use of elliptic curves in cryptography", H.C. Wiliams, Ed., Advances in Cryptology, CRYPTO 85, LNCS, vol. 218, pp: 417–426.

Morii M., Kasahara M., Whiting D. L., (Nov. 1989), "Efficient bit serial multiplication and the discrete time Wiener Hopf equation over finite fields", IEEE Transaction on Information Theory, vol. 35, pp:1177—1183.

Rodrıguez-Henrıquez Francisco, Morales-Luna Guillermo, Saqib Nazar A., Cruz-Cortes Nareli, (2007), "Parallel Itoh-Tsujii multiplicative inversion algorithm for a special class of trinomials", Des. Codes Cryptography, pp: 19—37.

Takagi N., (May 1998), "a vlsi algorithm for modular division based on the binary GCD algorithm", IEICE Transaction on Fundamentals, vol. E81-A, No.5, pp: 724—728.

Takagi N., Yoshika J., Takagi K., (May 2001), "A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis", IEEE Transaction on Computers, vol. 50, No. 5, pp: 394—398.

Tawalbeh L. A., Tenca A. F., (Sep. 2004), "An algorithm and hardware architecture for integrated modular division and multiplication in $GF(P)$ and $GF(2^n)$", Application Specific Systems, Architectures and Processors 2004, IEEE, pp: 247—257.

Tenca A. F., Tawalbeh L.A., (March 2004), "Algorithm for unified modular division in $GF(P)$ and $GF(2^m)$ suitable for cryptographic hardware", Electronics Letters, vol. 40, No. 5, pp: 304—306.

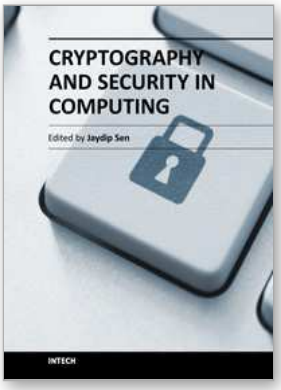
Wolkerstorfer Johannes, (2002), "Dual-field arithmetic unit for $GF(P)$ and $GF(2^m)$", International Workshop on Cryptographic Hardware and Embedded Systems CHES 2002, LNCS, vol. 2523, pp: 500—514.

Wu C., Wu C., Shieh M., Hwang Y., (2001), "Systolic VLSI realization of a novel iterative division algorithm over $GF(2^m)$: a high-speed, low-complexity design", ISCAS, pp: 33—36.

Wu C., Wu C., Shieh M., Hwang Y., (2004), "High speed, low complexity systolic designs of novel iterative division algorithms in $GF(2^m)$", IEEE Transaction on Computers, pp: 375—380.

Wu C. H., Wu C. M., Shieh M. D., Hwanng Y. T. , (Aug 2000), "Novel iterative division algorithm over $GF(2^m)$ and its systolic VLSI realization", Circuits and Systems, pp: 280—283.

Zadeh Abdulah Abdulah, (2007), "High speed modular divider based on GCD algorithm", Information and Communications Security, ICICS, LNCS, pp: 189—200.



Cryptography and Security in Computing

Edited by Dr. Jaydip Sen

ISBN 978-953-51-0179-6
Hard cover, 242 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

The purpose of this book is to present some of the critical security challenges in today's computing world and to discuss mechanisms for defending against those attacks by using classical and modern approaches of cryptography and other defence mechanisms. It contains eleven chapters which are divided into two parts. The chapters in Part 1 of the book mostly deal with theoretical and fundamental aspects of cryptography. The chapters in Part 2, on the other hand, discuss various applications of cryptographic protocols and techniques in designing computing and network security solutions. The book will be useful for researchers, engineers, graduate and doctoral students working in cryptography and security related areas. It will also be useful for faculty members of graduate schools and universities.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821