Inversion modular en Campos Finitos Binarios

Marco Antonio Negrete Cervantes [*mnegrete@computacion.cs.cinvestav.mx*]
Karla Paulina Gómez Avila [*kpgomez@computacion.cs.cinvestav.mx*]
Dr. Francisco Rodríguez-Henríquez [*francisco@cs.cinvestav.mx*]

Mayo, 2006

Centro de Investigación y Estudios Avanzados del IPN
Departamento de Ingeniería Eléctrica Sección de Computación

Resumen

En esta investigación fueron implementados en hardware dos algoritmos para encontrar los inversos de un elemento en un campo finito binario $a \in F_{2^m}$. Los algoritmos estudiados fueron el algoritmo extendido binario de Euclides y el algoritmo de Itoh-Tsujii, haciendo una comparación entre los resultados obtenidos de velocidad con respecto al área.

El inverso de un elemento no cero $a \in F_{2^m}$ es el único elemento $g \in F_{2^m}$ para el cual se cumple que $ag = 1 \in F_{2^m}$, esto es $ag \equiv 1 mod\,(f)$. Este inverso es denotado como $a^{-1} mod\,(f)$ o simplemente a^{-1} si la reducción polinomial f es comprendida en el contexto. La aritmética en campos binarios encuentra una aplicación relevante en el desarrollo de curvas elípticas.

1. Introducción

*U*n campo finito es un sistema algebraico que consiste de un conjunto finito F junto con 2 operaciones binarias, + y *, sobre los elementos de F y que satisface los siguientes axiomas:

1. Los elementos 0 y 1 se encuentran en F.
2. F es un grupo abeliano[1] respecto a la operación +.
3. F $\{0\}$ es un grupo abeliano respecto a la operación $*$.
4. Para todo x, y y z en F, $x * (y + z) = (x * y) + (x * z)$ y $x + (y * z) = (x + y) * (x + z)$.

*L*a aritmética en campos binarios es principalmente aplicada en el desarrollo de curvas elípticas. Una curva elíptica es un conjunto de pares ordenados que cumplen una relación para propósitos de la criptografía. Esta relación se define sobre elementos de un campo finito. En la práctica, los campos finitos sobre los que se definen las curvas elípticas son dos: el campo de los números primos GF (p) y el campo de las cadenas de m bits GF (2^m). En ambos casos, las operaciones de adición y multiplicación terminan siempre con reducción módulo p (F_p) o $m\,(F_{2^m})$.

*L*a característica importante de las curvas elípticas es que cuando se definen sobre los campos GF(p) o $GF\,(2^m)$, el conjunto de puntos que conforman a tal curva elíptica forman un grupo abeliano sobre la operación de suma. Con esto, cualquier operación bien definida que se realice entre los puntos de la curva, siempre resultará en otro punto de la curva elíptica. Otra característica importante del grupo abeliano es como se define el problema del logaritmo discreto sobre los puntos de la curva elíptica es intratable, es decir, el mejor algoritmo conocido para resolver el problema del logaritmo discreto de curva elíptica es de complejidad exponencial.

*E*l problema del logaritmo discreto en curvas elípticas consiste en encontrar el valor del número entero k tal que dados dos puntos cualesquiera P, Q de la curva elíptica, la relación $P = kQ$ se cumpla (k y el orden de Q son números suficientemente grandes). En este problema se basa la seguridad de los criptosistemas de curva elíptica.

*E*l mérito que tienen los criptosistemas de curvas elípticas (Elliptic Curve Cryptosystems denotados ECC) es que, comparado con el criptosistema de llave pública más usado, el RSA, ECC puede proveer el mismo nivel de seguridad con una longitud de llave más pequeña, con una reducción en el tamaño de la llave de 7:1. Con una llave pequeña, las operaciones criptográficas, como la firma digital, puede realizarse más rápido. El uso de llaves pequeñas implica mayor velocidad, y menores necesidades de memoria y de poder de cómputo para la implementación de los algoritmos correspondientes a este esquema. El tiempo para la transmisión y el espacio para almacenamiento de las llaves también se reducen en un factor de 7. Estas características son atractivas especialmente

[1] Un grupo abeliano es un grupo $(G, +)$ tal que $a + b = b + a$ para todos los elementos $a, b, \in G$, es decir, es un grupo conmutativo.

para aplicaciones en dispositivos móviles, los cuales cuentas con modestos recursos computacionales. Un algoritmo criptográfico que opere sobre puntos de una curva elíptica (ECDSA) puede tener el mismo nivel de seguridad que un algoritmo criptográfico que opere sobre enteros grandes (RSA) usando mucho menos bits.

2. Desarrollo

2.1. Algoritmo Extendido Binario de Euclides

*S*e tiene dos polinomios binarios a y b, ambos diferentes de cero. El maximo común divisor (gcd, greatest common divisor) de a y b, denotado por gcd (a, b), es el polinomio binario d del grado más alto que divide ambos polinomios a y b. Los algoritmos eficientes para el cálculo del gcd(a, b) se basan en el siguiente teorema.

Teorema 1. Se tiene dos polinomios binarios a y b. Entonces $gcd = (a, b) = gcd(b - ca, a)$ para cualquier polinomio binario c.

En el Algoritmo de Euclides clásico para el cálculo del gcd de dos polinomios binarios a y b, cuando el $grado(b) \geq grado(a)$, b es divido por a para obtener un cociente q y un residuo r, que satisfacen la ecuación $b = qa + r$ y $grado(r) \prec grado(a)$. Por el Teorema 1, el $gcd(a, b) = gcd(r, a)$. Así, el problema de determinar el $gcd(a, b)$ es reducido a calcular $gcd(r, a)$ donde los argumentos (r, a) tienen un grado menor que el grado de los argumentos originales (a, b). Este proceso es repetido hasta que uno de los argumentos es cero, el resultado obtenido inmediatamente de $gcd(0, d) = d$. El algoritmo debe terminar como el grado de los residuos va disminuyendo paulatinamente. Por otro lado, este algoritmo es eficiente porque el número de divisiones largas es a lo más k, donde $k = grado(a)$.

En una variante del Algoritmo clásico de Euclides, solamente un paso en cada división larga es modificado. Esto es, si el $grado(b) \geq grado(a)$ y $j = grado(b) - grado(a)$ entonces se calcula Por el Teorema 1, $gcd(a, b) = gcd(r, a)$. Este proceso es repetido hasta encontrar un residuo igual a cero. Si el $grado(r) \prec grado(b)$, el número de divisiones parciales es a lo más k, en donde $k = max\{grado(a), grado(b)\}$.

El algoritmo de euclides puede ser extendido para encontrar polinomios binarios g y h que satisfagan la ecuación $ag + bh = d$ donde $d = gcd(a, b)$. En el algoritmo original se mantienen fijas

$$ag_1 + bh_1 = u$$
$$ag_2 + bh_2 = v$$

El algoritmo termina cuando $u = 0$, con lo cual $v = gcd(a, b)$ y $ag_2 + bh_2 = d$.

Supóngase ahora que f es un polinomio binario irreducible de grado m y un polinomio no-cero a tiene grado $m - 1$ (por lo tanto $gcd(a, f) = 1$). Si el Algoritmo original es ejecutado con las entradas a y f, el último cociente u no-cero es encontrado despues de realizar una substracción entre de los grados de los polinomios, y es $u = 1$. Después de que esto ocurra, los polinomios g_1 y g_2 son actualizados, y satisfacen la ecuación $ag_1 + fh_1 = 1$. Por lo tanto $ag_1 \equiv 1(modf)$ y $a^{-1} = g_1$ Nótese que h_1 y h_2 no se necesitan para determinar g_1. Esta observación es implementada en el siguiente Algoritmo para realizar inversos en un campo F_{2^m}.

Algorithm 1. Binary algorithm for inversion in F_{2^m}

Input: A nonzero binary polynomial a of degree at most $m - 1$.

Output: $a^{-1} mod\ f$.

1.$u \leftarrow a, v \leftarrow b$.

1.$g_1 \leftarrow 1, g_2 \leftarrow 0$.

3.$While\ (u \neq 1, v \neq 1) do$

$3 \cdot 1.\ While\ z\ divides\ u do$

$u \leftarrow u/z$.

$if z\ divides\ g_1\ then\ g_1 \leftarrow g_1/z; else\ g_1 \leftarrow (g_1 + f)/z$.

$3 \cdot 1.\ While\ z\ divides\ v do$

$v \leftarrow v/z$.

$if z\ divides\ g_2\ then\ g_2 \leftarrow g_2/z; else\ g_2 \leftarrow (g_2 + f)/z$.

$3 \cdot 3.\ if\ deg(u) > deg(v)\ then:\ u \leftarrow u + v, g_1 \leftarrow g_1 + g_2;$

$Else:\ v \leftarrow v + u, g_2 \leftarrow g_2 + g_1$.

4.$If\ u = 1\ then\ return\ (g_1); else\ return\ (g_2)$.

Este algoritmo fue implementado para un FPGA debido a que los cálculos involucrados con el grado de los polinomios en el paso 3.3 se reducen a una comparación simple de la representación binaria de los polinomios, esto difiere en el algoritmo 1 en donde sí se requiere un cálculo explícito de los grados de los polinomios en el paso 3.1.

2.2. Algoritmo Itoh-Tsujii

El algoritmo usado para la inversión esta basado en la identidad del Teorema Petit de Fermat:

$$a^{-1} = a^{2^m - 2} = \left[a^{2^{m-1}-1}\right]^2$$

Itoh y Tsujii propusieron un método que minimiza el número de multiplicadores para calcular las inversiones el cual esta basado en las siguientes identidades:

$$a^{2^{m-1}-1} = \left\{(a^{2^{m-1/2}-1})^{2^{m-1/2}} \bullet a^{2^{m-1/2}-1}\right\}, impar$$

$$a^{2^{m-1}-1} = \left\{a(a^{2^{m-1/2}-1})^2\right\}, par$$

El algoritmo Itoh-Tsujii calcula los inversos

$$\left(a^{2^{m-1}-1}\right)$$

de los elementos en el campo usando arreglos recursivos de operaciones en campos finitos.

Para el desarrollo del algoritmo se tiene una α que es un elemento no-cero arbitrario en el campo $GF(2^m)$. Entonces, se define $\beta \in GF(2^m)$, donde k es un entero positivo, y se obtiene:

$$\beta_k = \alpha^{2^k-1}$$

Los $k-bit$ de α serán representados en su expansión binaria, y se considera como el número positivo $m-1$, la expansión es representada de la siguiente manera:

$$m-1 = 2^{l_1} + 2^{l_2} + \ldots + 2^{l_{t-1}} + 2^l_t$$

Donde $l_1 \prec l_2 \prec \ldots \prec l_{t-1} \prec l_t = k-1$.

Entonces el inverso multiplicativo de α, representado por $\alpha^{-1} \in GF(2^m)$, se puede representar por:

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

Aplicando las identidades antes propuestas se obtiene:

$$\beta^2 = (\alpha^{2^{m-1}-1})^2 = (\alpha^{2^m} - 2) = \alpha^{-1}$$

De estas definiciones se obtiene el algoritmo propuesto por Itoh-Tsujii

Algorithm 2. Algorithm Itoh-Tsujii

Input: An element $a \in GF(2^M)$, an addition chain U of length t for $m-1$ and its associated sequence V.

Output: $a^{-1} mod\ f$.

Procedure $MultiplicativeIversion(\alpha,U)$

1. $\beta_0 = a$
2. $for\ i\ from\ 1\ to\ t\ do:$
3. $\beta_i = (\beta_{i_1})^{2^{u_{i_2}}} \cdot \beta_{i_2}$
4. $return\ (\beta_t{}^2)$;

Este algoritmo fue implementado en un FPGA para el cálculo de inversos en polinomios de grado 193, la generación de coeficientes para el polinomio irreducible $x^{193}+x^{15}+1$ con el algoritmo Itoh-Tsujii es desarrollado por medio de la cadena de adición obtenida para m-1=192.

$$U = \{1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 192\}$$

.

2.3. Implementación en Hardware Reconfigurable

*L*a implementación en Hardware es económica debido a que el lenguaje VHDL es un lenguaje descriptor de hardware, con esta característica y el trabajo de aritmética binaria el uso de los recursos de la tarjeta se minimiza produciendo un diseño simple y que puede ser transferido a cualquier tarjeta.

Al ser un diseño simple, debido al trabajo de aritmética binaria y al nulo empleo de componentes adicionales (Componentes Core Generator), es posible implementarlo en casi cualquier tarjeta, aunque es deseable hacer un uso eficiente de los recursos, por lo que la selección para el diseño es utilizar la Tarjeta Virtex 2 xc2v4000-6bf957 debido a la cantidad de entradas y salidas.

El lenguaje y los métodos que existen en VHDL están pensados en aritmética entera, en la que los acarreos entre dígitos están presentes y son considerados en las operaciones, mientras que en aritmética binaria estos no existen. Con esto es necesario implementar funciones o modificar las existentes para realizar las operaciones básicas, tales como el paquete de corrimientos implementado para solucionar la aritmética binaria en VHDL, este paquete realiza la multiplicación y división por 2 que en esta aritmética se reducen simplemente a corrimientos a la izquierda y a la derecha respectivamente.

Los ciclos en este lenguaje descriptor de hardware no se encuentran totalmente soportados, los únicos ciclos que están bien definidos en VHDL son aquellos que tienen un número de iteraciones bien determinado, es decir que no dependan de una variable externa. En los algoritmos implementados existen varios ciclos que dependen del número del cual se quiera encontrar su inverso, por lo que no fue posible implementar para síntesis estos ciclos en la forma tradicional secuencial. Para resolver esto se cambiaron los ciclos por una máquina de estados que emule de una forma eficiente en hardware estos procedimientos.

Para el algoritmo de Itoh-Tsujii además de implementar el paquete de algebra binaria, se usaron dos módulos más, uno para crear las multiplicaciones y otro para elevar al cuadrado una serie de números. La multiplicación fue realizada por el método de Karatsuba-Ofman para árboles desbalanceados.

El diseño, también, requiere de elevar un polinomio a distintas potencias, dependiendo del paso de la cadena de adición que se esté realizando. Estas potencias son todas cuadrados. En este módulo se obtienen los valores de los cuadrados del parámetro de entrada en una tira. En cada paso de la tira es posible obtener valores intermedios, cuando no sea requerido encontrar todos los valores. Adicionalmente todos los valores calculados son reducidos al mismo tiempo que son calculados.

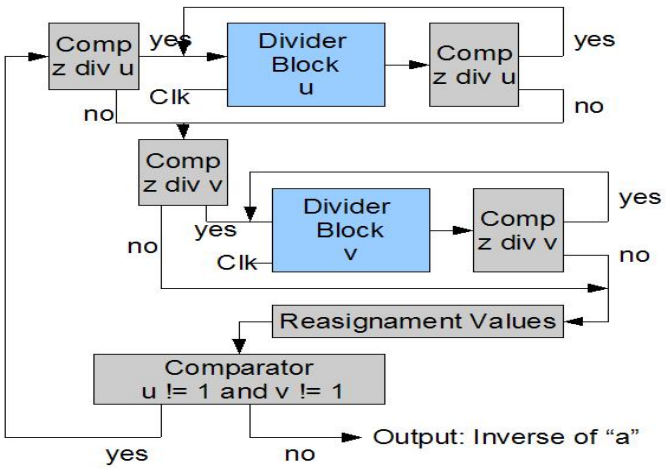

Fig 1. Diagrama a Bloques para el Algoritmo Extendido de Euclides

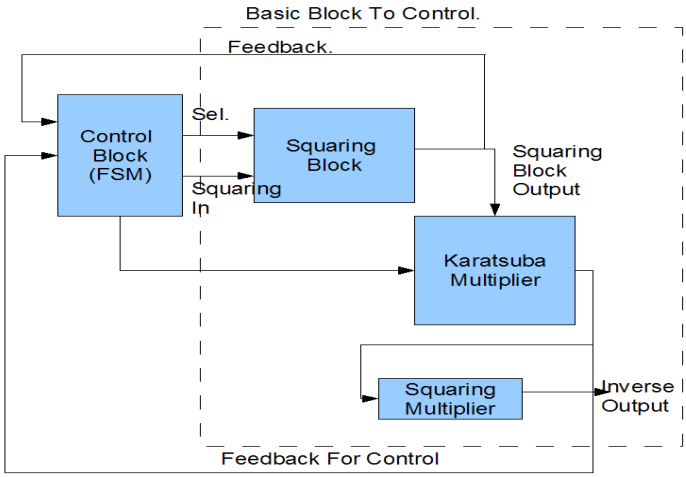

Fig 2. Diagrama a Bloques para el Algoritmo Itoh-Tsujii

3. Resultados

Los resultados obtenidos para cada algoritmo son presentados en la siguiente tabla comparativa

Parámetro	Ext. Euclides	Itoh-Tsujii	Itoh-Tsujii sin Mult. Karatsuba
Slices Usados	1195	9945	2345
Periodo Mínimo	13.140 (ns)	18.098 (ns)	18.098 (ns)
Frecuencia máxima	76.103 MHz	55.254 MHz	55.254 MHz
Número de ciclos promedio	191	40	40
Tiempo total	2.509 μs	0.760 μs	0.760 μs
Relación Tiempo-área $(\mu s/slices)$	333.53	132.30	561.1

4. Conclusiones

La mayoría de los productos y estándares que usan criptografía de llave asimétrica para cifrado y firmas digitales usan RSA. Actualmente se esta realizando investigación alrededor de ECC, principalmente orientándose a la implementación de distintas aplicaciones en dispositivos de hardware reconfigurable, ya que la ganancia obtenida en recursos sobre velocidad es mayor en comparación con la obtenida en software.

El algoritmo binario extendido de Euclides resulta una implementación en hardware muy sencilla. Las operaciones y comparaciones que se tienen que realizar se reducen a operaciones booleanas y corrimientos, con esto la implementación en un FPGA es de un costo en área muy bajo debido a la escasa lógica combinatoria y secuencial. Así la implementación es posible en casi en cualquier tarjeta a pesar de que tenga recursos limitados, la limitante de este diseño tan simple es que en tiempo no es tan óptima.

El algoritmo de Itoh-Tsujii es un algoritmo que en recursos de hardware resulta costoso, debido a la serie de potencias que se realizan, hasta que en determinado punto se obtiene el inverso, pero su costo en tiempo resulta bueno, esto es debido a que son pocos los pasos que se tienen que hacer para encontrar la solución, esto depende de la cadena de adición que se seleccione.

La implementación de ambos algoritmos en un lenguaje descriptor de hardware resulto un poco complicada debido a la naturaleza no secuencial del lenguaje, la optimización de los diseños depende de la forma en la estos se conciban, un diseño creado en una forma secuencial no será un diseño óptimo en hardware mientras que uno que se piense y diseñe para su ejecución concurrente y donde se aprovechen las ventajas del paralelismo resultará un buen diseño tanto en área como en tiempo.

De acuerdo a la tabla comparativa se observa que el algoritmo de Euclides representa una mejor opción en cuanto a área, ocupa sólo un 12.02 % del área que ocupa el segundo, por lo que es factible implementarlo en cualquier tarjeta, mientras que el segundo diseño no es posible. En cuestión de tiempo, en el caso de Euclides presenta tiempos promedios que son 3.3 veces más lentos que el diseño basado en Fermat. Con todo esto es posible afirmar que no vale la pena el costo en área para hacer un algoritmo 3.3 veces más rápido pagando 8.3 veces más el área, aunque si consideramos el multiplicador como un bloque ajeno, entonces el Itoh-Tsujii se vuelve la opción conveniente ya que la relación Tiempo-Área mejora muchísimo y pasa de 1:8.3 a 1:1.96, mientras que la relación entre tiempos permanece igual, por lo que aquí si es una opción viable sacrificar el doble de área para obtener el triple de velocidad.

Referencias

[1] Wade Trappe and Lawrence C. Washington: *Introduction to Cryptography with Coding Theory*, (Prentice Hall; 1st edition (January 15, 2002)).

[2] Darrel Hankerson, Julio López-Hernández and Alfred Meneses: *Software Implementation or Elliptic Curve. Cryptography over Binary Fields.*

[3] Francisco Rodríguez-henríquez, Guillermo Morales-Luna, Nazar A. Saqib and Nareli Cruz-Cortés: *Parallel Itoh-Tsujii Multiplicative Inversion. Algorithm for a Special Class of Trinomials*

[4] $http://www.springer.com/sgw/cda/pageitems/document/cda_downloaddocument/0,11996,0-0-45-110359-0,00.pdf?referer=www.springeronline.com$