Capítulo 4

Operaciones aritméticas básicas

4.1. Introducción

En este capítulo presentaremos los algoritmos paralelos para la realización de las operaciones aritméticas que son básicas en los principales métodos de criptografía de clave pública [66]. Consideraremos a las operaciones con números enteros utilizadas en sistemas basados en la exponenciación así como los basados en curvas elípticas sobre el campo de Galois $GF(p)$, donde p es un primo. También incluiremos a la multiplicación polinomial, operación básica para los sistemas basados en curvas elípticas definidas sobre $GF(2^n)$.

4.2. Adición paralela

4.2.1. Algoritmo de Brent

En esta sección presentaremos un algoritmo propuesto por Brent [14] para la adición de dos enteros. Sean $a = a_N a_{N-1} \cdots a_1$ y $b = b_N b_{N-1} \cdots b_1$ dos enteros binarios y sea $s = (a + b)\ mod\ 2^N$, donde $s = s_N s_{N-1} \cdots s_2 s_1$. Es bien conocido que

$$s_i = a_i \oplus b_i \oplus c_{i-1} \tag{4.1}$$

donde $c_0 = 0$ y para $i = 1$ a N

$$\begin{aligned} c_i &= p_i \wedge (g_i \vee c_{i-1}) \\ p_i &= a_i \vee b_i \\ g_i &= a_i \wedge b_i \end{aligned} \tag{4.2}$$

donde $\oplus$,$\vee$ y $\wedge$ son las operaciones OR exclusivo, OR y AND respectivamente. c_i es el *acarreo* del bit i-ésimo, y p_i y g_i son las condiciones de *generación* y *propagación* del acarreo, respectivamente. Como $c_0 = 0$, distribuyendo p_i en la ecuación anterior obtenemos

$$\begin{aligned} c_1 &= p_1 \wedge g_1 \\ c_2 &= p_2 \wedge (g_2 \vee (p_1 \wedge g_1) \\ &\vdots \\ c_i &= p_i \wedge (g_i \vee (p_{i-1} \wedge (g_{i-1} \vee \cdots \vee (p_1 \wedge g_1) \cdots))) \end{aligned} \tag{4.3}$$

Los bits de propagación p_i y de generación g_i, para $i = 1, \cdots, n$ pueden calcularse en paralelo en un solo paso utilizando un arreglo lineal de N compuertas AND y N compuertas OR. Una vez que todos los bits de acarreo son conocidos, pueden calcularse los s_i en exactamente dos pasos. Por lo tanto, si $T_A(N)$ es el tiempo requerido para sumar dos números binarios de N-bits y $T_C(N-1)$ es el tiempo necesario para calcular los $N-1$ bits de acarreo c_1 hasta c_N usando la ecuación anterior, entonces

$$T_A(N) = T_C(N-1) + 3 \tag{4.4}$$

El procedimiento para calcular los bits de acarreo es el siguiente: Sean $r \geq 1$ y $q \geq 1$ enteros tales que $N = rq$.

1. Utilizando $2N$ compuertas calcular $p_i = a_i \vee b_i$ y $g_i = a_i \wedge b_i$, para $i = 1, \cdots, N$.

2. Utilizando $s(q)$ compuertas, calcular

   $$E_i = p_{iq} \wedge (g_{iq} \vee \cdots \vee (p_{(i-1)q+1} \wedge g_{(i-1)q+1}) \cdots)$$

   para $i = 1, \cdots, r$.

3. Utilizando $r(q-1)$ compuertas, calcular

   $$P_i = p_{iq} \wedge \cdots \wedge p_{(i-1)q+1}$$

   para $i = 1, \cdots, r$.

4. Utilizando $(r/2)log\ r$ compuertas calcular

   $$D_i = P_r \wedge \cdots \wedge P_{i+1}$$

   donde $D_r = 1$ y $i = 1, \cdots, r$.

5. Utilizando r compuertas, calcular

$$F_i = D_i \wedge E_i$$

para $i = 1, \cdots, r$.

6. Calcular $C_N = F_r \vee F_{r-1} \vee \cdots \vee F_2 \vee F_1$ utilizando $(r-1)$ compuertas.

Obsérvese que el paso 2 puede realizarse en paralelo a los pasos 3 y 4. A continuación obtendremos una cota para el tiempo $T_C(N)$.

Lema 4.1 *(Ver [14]) Sea $N = rq$. Entonces*

$$T_C(N) \leq 1 + \lceil \log r \rceil + \text{máx}\{T_C(q), \lceil \log(r-1) \rceil + \lceil \log q \rceil\}$$

Demostración
Para demostrar este lema, obsérvese que, debido a la similaridad en la forma de E_i y c_N, se tiene que E_i puede calcularse en paralelo en $T_C(N)$ pasos. Simultaneamente, podemos calcular las P_i en paralelo en $\lceil \log q \rceil$ pasos [1] seguido del cálculo de las D_i en paralelo en $\lceil \log(r-1) \rceil$ pasos. Entonces puede calcularse $F_i = D_i \wedge E_i$ en un solo paso, y $C_N = F_r \vee F_{r-1} \vee \cdots \vee F_2 \vee F_1$ puede calcularse entonces en $\lceil \log r \rceil$ pasos. Debido a que las F_i no pueden calcularse sino hasta que E_i y D_i han sido calculados, de la combinacion de los tiempos mencionados se sigue el lema anterior [14]. □

Sea $N = 2^n$, y a y k dos enteros no negativos. Definimos

$$T(n) = T_C(2^n) - n \tag{4.5}$$

Lema 4.2 *(Ver [14])*

$$T(a + kT(a) + \frac{k(k-1)}{2}) \leq k + T(a) \tag{4.6}$$

Demostración
Sea $N = rq$, donde $r = 2^{n_1}$, $q = 2^{n_2}$ y $n_1 + n_2 = n$. Del lema 4.1 tenemos que

$$T_C(2^{n_1+n_2}) \leq 1 + n_1 + \text{máx}\{T_C(2^{n_2}), n_1 + n_2\}$$

de donde se sigue

$$\begin{aligned} T(n_1 + n_2) &\leq 1 + \text{máx}\{T_C(2^{n_2}) - n_2, n_1\} \\ &= 1 + \text{máx}\{T(n_2), n_1\} \end{aligned} \tag{4.7}$$

[1] A menos que se indique lo contrario, los logaritmos son en base 2.

Probaremos entonces el lema por inducción. Para $k = 0$ es obvio. Supongamos que es válido para alguna k y demostremos que se cumple para $k+1$. Sean $n_1 = k + T(a)$ y $n_2 = a + kT(a) + \frac{k(k-1)}{2}$, de la desigualdad previa, tenemos que

$$\begin{aligned} T(a + kT(a) &+ \frac{k(k-1)}{2}) \\ &\leq 1 + \text{máx}\{T[a + kT(a) + \frac{k(k-1)}{2}], k + T(a)\} \\ &\leq (1 + k) + T(a) \end{aligned}$$

□

Teorema 4.3 *(Ver [14])*

$$T_C(2^{k(k-1)/2}) \leq \frac{k(k+1)}{2} \tag{4.8}$$

Demostración
De la ecuación (4.5) tenemos que $T(0) = T_C(1) = 1$. Haciendo $a = 0$ en (4.6), tenemos

$$T(\frac{k(k+1)}{2}) \leq k + 1$$

Nuevamente, de (4.6),

$$T_C(2^{k(k+1)/2}) \leq (k+1) + \frac{k(k+1)}{2} = \frac{(k+1)(k+2)}{2}$$

Sustituyendo k por $(k-1)$ probamos el enunciado del teorema. □

Para extender este resultado en el caso en que N no es una potencia de 2, sea k tal que

$$2^{k(k-1)/2} \geq N \geq 2^{(k-1)(k-2)/2}$$

y, dado que $k \approx (\lceil 2 \log N \rceil)^{1/2}$, se tiene

$$\begin{aligned} T_C(N) &\leq \frac{k(k+1)}{2} \\ &= \lceil \log N \rceil + \frac{1}{2}(\lceil 2 \log N \rceil)^{1/2} + L \end{aligned} \tag{4.9}$$

donde L es una constante positiva pequeña. Por lo tanto,

$$\lceil \log 2N \rceil \leq T_C(N) \leq \lceil \log N \rceil + \frac{1}{2}(\lceil 2 \log N \rceil)^{1/2} + L \tag{4.10}$$

Podemos concluir que el algoritmo de Brent realiza la suma de dos enteros de N bits en $O(\log N)$ pasos, si bien requiere de un circuito de tamaño $O(N \log N)$.

4.3. Multiplicación paralela

4.3.1. Multiplicación de primaria en paralelo

Sean $a = a_N a_{N-1} \cdots a_2 a_1$ y $b = b_N b_{N-1} \cdots b_2 b_1$ dos enteros binarios y sea $s = a \times b = s_{2N} s_{2N-1} \cdots s_2 s_1$.

1. Encontrar los N productos parciales $x_1, x_2, \cdots, x_N$, cada uno de los cuales es de $2N-1$ bits de longitud.

$$\begin{array}{cccccccc} x_1 = 0 & & 0 & \cdots & 0 & a_N \wedge b_1 & \cdots & a_1 \wedge b_1 \\ x_2 = 0 & & 0 & \cdots & a_N \wedge b_2 & a_{N-1} \wedge b_2 & \cdots & 0 \\ \vdots & & & & & & & \\ x_N = a_N \wedge b_N & & a_{N-1} \wedge b_N & \cdots & a_1 \wedge b_N & 0 & \cdots & 0 \end{array}$$

   donde $a_i \wedge b_j$ es la operación lógica AND.

2. Sumar el anterior conjunto de N números $x_1, x_2, \cdots, x_N$, usando el *abanico asociativo* [54], donde cada suma es realizada mediante un algoritmo paralelo de tiempo $O(\log N)$ utilizando $O(N)$ elementos lógicos.

El paso 1 puede efectuarse en una unidad de tiempo utilizando N^2 elementos lógicos. El algoritmo de abanico asociativo puede realizarse en $\lceil \log N \rceil$ etapas de suma de enteros de $2N$ bits. Cada uno de estas últimas sumas puede hacerse en $O(\log N)$ pasos usando $O(N)$ elementos lógicos. Por lo tanto, el paso 2 toma $O((\log N)^2)$ unidades de tiempo usando $O(N^2)$ elementos. De lo anterior, obtenemos el siguiente teorema.

Teorema 4.4 *El producto de dos enteros de N bits puede obtenerse en paralelo mediante la multiplicación de primaria utilizando $O(N^2)$ elementos lógicos en $O((\log N)^2)$ pasos.*

4.3.2. Algoritmo de Karatsuba-Ofman

Para realizar la multiplicación de dos enteros a y b de N bits, donde $N = 2^n$, puede seguirse una estrategia de *divide y vénceras* [47]. Primero, reescribamos a y b como

$$\begin{aligned} a &= A_1 \times 2^{N/2} + A_0 \\ b &= B_1 \times 2^{N/2} + B_0 \end{aligned} \tag{4.11}$$

donde A_i y B_i, para $i = 0, 1$, son enteros de $N/2$ bits, A_0 y A_1 son el residuo y el cociente de la división de a entre $2^{N/2}$, respectivamente, al igual que B_0 y B_1 son cociente y residuo para b. Definamos entonces

$$\begin{aligned} r_0 &= A_0 \times B_0 \\ r_1 &= (A_1 + A_0) \times (B_1 + B_0) \\ r_2 &= A_1 \times B_1 \end{aligned} \tag{4.12}$$

Entonces

$$a \times b = r_2 \times 2^N + (r_1 - r_2 - r_0)2^{N/2} + r_0 \tag{4.13}$$

Para realizar la multiplicación se aplica recursivamente el método para obtener los valores de r_0,r_1 y r_2.

Sea $T(N)$ el tiempo requerido para realizar la multiplicación de a y b por este método. Obsérvese que las multiplicaciones para obtener r_0,r_1 y r_2 pueden efectuarse en paralelo. Más aún, las operaciones en (4.12) y (4.13) pueden realizarse en paralelo en $O(\log N)$ pasos. Por lo tanto

$$T(N) = T(\frac{N}{2}) + c_1 \log N \tag{4.14}$$

donde c_1 depende del número total de sumas. Tomando $T(1) = 1$ y resolviendo la relación de recurrencia (4.14) obtenemos

$$T(N) = O((\log N)^2) \tag{4.15}$$

Para mejorar el desempeño de este método, observando la recurrencia (4.14), una posibilidad es reducir el término $c_1 \log N$ a una constante. Esto se logra haciendo que el tiempo para sumar los enteros permanezca constante. El algoritmo de Mehlhorn-Preparata logra esto basandose en una representación modular en base 4 redundante (RR-4).

4.3.3. Multiplicación mediante convolución

El método de Karatsuba-Ofman puede ser extendido fácilmente a la separación de a y b en t segmentos, donde $t \geq 2$. Si cada uno de estos t segmentos es de m bits, entonces $N = tm$, y se tiene

$$\begin{aligned} a &= A_{t-1}2^{(t-1)m} + A_{t-2}2^{(t-2)m} + \cdots + A_1 2^m + A_0 \\ b &= B_{t-1}2^{(t-1)m} + B_{t-2}2^{(t-2)m} + \cdots + B_1 2^m + B_0 \end{aligned} \tag{4.16}$$

donde A_j, $0 \leq j \leq t-1$, son enteros de m bits correspondientes al bloque del bit $((j-1)m+1)$ al bit (jm) de a, y B_j es el análogo para b.

Definimos los polinomios

$$p(x) = A_{t-1}x^{(t-1)} + A_{t-2}x^{(t-2)} + \cdots + A_1x + A_0$$

$$q(x) = B_{t-1}x^{(t-1)} + B_{t-2}x^{(t-2)} + \cdots + B_1x + B_0$$

y

$$r(x) = r_{2t-1}x^{(2t-1)} + r_{2t-2}x^{(2t-2)} + \cdots + r_1x + r_0$$

donde $r(x) = p(x)q(x)$. Obsérvese que $a = p(2^m)$, $b = q(2^m)$ y en consecuencia $a \times b = r(2^m)$. Efectuando la multiplicación de $p(x)$ por $q(x)$ encontramos los coeficientes del polinomio $r(x)$, esto es

$$\begin{aligned}
r_0 &= A_0B_0 \\
r_1 &= A_0B_1 + A_1B_0 \\
r_2 &= A_0B_2 + A_1B_1 + A_2B_0 \\
&\vdots \\
r_{t-1} &= A_0B_{t-1} + A_1B_{t-2} + \cdots + A_{t-1}B_0 \\
r_t &= A_1B_{t-1} + A_2B_{t-2} + \cdots + A_{t-1}B_1 \\
&\vdots \\
r_{2t-2} &= A_{t-1}B_{t-1} \\
r_{2t-1} &= 0
\end{aligned}$$

Si definimos los vectores $\mathbf{A} = (A_0, A_1, \cdots, A_{t-1})^t$ y $\mathbf{B} = (B_0, B_1, \cdots, B_{t-1})^t$, entonces el vector $\mathbf{r} = (r_0, r_1, \cdots, r_{2t-1})^t$ es denominado la *convolución* de $\mathbf{A}$ y de $\mathbf{B}$. El cálculo directo de $\mathbf{r}$ implica la realización de t^2 multiplicaciones de enteros de m bits. Sin embargo, la convolución puede efectuarse en forma bastante eficiente usando la *transformada rápida de Fourier*.

4.4. Transformada rápida de Fourier modular

4.4.1. Raices de la unidad en aritmética modular

Sea $m \geq 2$ un entero. El conjunto $SRC = \{0, 1, 2, \cdots, m-1\}$ de todos los *residuos* módulo m es denominado el *sistema de residuos completo* (SRC), mientras que el conjunto

$$SRR = \{x \mid x \in SRC \text{ y } \text{MCD}(s, m) = 1\}$$

de todos los residuos que son primos relativos de m es denominado el *sistema reducido de residuos* (SRR). Sea $\phi(m)$ el número de enteros positivos menores que m y primos relativos a m, entonces es obvio que

$$\phi(m) = |SRR| < |SRC| = m$$

para toda $m > 1$. $\phi(m)$ es denominada la *función de Euler*.

Teorema 4.5 *(Teorema Pequeño de Fermat) Si p es un primo y $1 < a < p$ entonces*

$$1 \equiv a^{p-1} (\text{mod } p) \tag{4.17}$$

Demostración
Dado que $\text{MCD}(a, p) = 1$, $y \equiv ax \pmod{p} \in SRR$ si $x \in SRR$. Sea $x_i = 1, \cdots, p-1$ una enumeración de los elementos de SRR. Entonces,

$$\prod_{i=1}^{p-1} y_i \equiv (\prod_{i=1}^{p-1} (ax_i)) (\text{mod } p). \tag{4.18}$$

de donde,

$$\prod_{i=1}^{p-1} y_i \equiv (a^{p-1} \prod_{i=1}^{p-1} x_i) (\text{mod } p). \tag{4.19}$$

Y como,

$$\prod_{i=1}^{p-1} y_i = \prod_{i=1}^{p-1} x_i, \tag{4.20}$$

de (4.20) y (4.19) se sigue (4.17). □

El anterior resultado puede extenderse a módulos que no son primos, para dar origen al denominado *teorema de Euler*.

Lema 4.6 *Si* $\text{MCD}(a, m) = 1$ *y* $m > 1$, *entonces*

$$1 \equiv a^{\phi(m)} (\text{mod } m). \tag{4.21}$$

Del lema anterior puede deducirse que a es la raíz $\phi(m)$-ésima de la unidad módulo m.

A continuación presentaremos algunos resultados de teoría de números concernientes al estudio de raices de la unidad en aritmética modular.

Lema 4.7 *Sean a y m dos enteros positivos tales que $a < m$. Entonces para que exista un entero t tal que $a^t \equiv 1 (\text{mod } m)$, es condición necesaria y suficiente que* $\text{MCD}(a, m) = 1$. *Esto es, la t-ésima raíz de la unidad módulo m es siempre prima relativa a m.*

Demostración
Por el teorema de Euler $t = \phi(m)$ si a y m son primos relativos. Ahora supongamos que $\text{MCD}(a, m) = d > 1$ y $a^t \equiv 1(\text{mod } m)$ para alguna $t \geq 1$. Entonces d divide a a, a^t y m. Pero, como $a^t = mk + 1$ para alguna k, eso implica que d también divide a 1, lo cual es una contradicción. Entonces $d = 1$ y el lema se cumple. □

Si t es el menor entero positivo tal que $a^t \equiv 1(\text{mod } m)$ entonces t es denominado el *orden de* $a(\text{mod } m)$ y se denota por $\text{ord}_m(a)$.

Lema 4.8 *Si* $t = \text{ord}_m(a)$*, entonces las potencias* $a, a^2, \cdots, a^t$ *son todas distintas* (mod m)*.*

Demostración
Sean $1 \leq s < r \leq t$. Entonces

$$a^s \equiv a^r(\text{mod } m)$$

implicaría que m divide a $a^s(a^{r-s} - 1)$. Dado que $\text{MCD}(a, m) = 1$, se sigue que $\text{MCD}(a^s, m) = 1$. De lo anterior, tenemos que m divide a $a^{r-s} - 1$, esto es, $a^{r-s} \equiv 1(\text{mod } m)$ y $0 < r - s < t$. Esta es una contradicción con la definición de t, y por lo tanto se cumple el lema. □

Partiendo de lo anterior, puede llegarse a las siguientes conclusiones:

- Si $a^k \equiv 1(\text{mod } m)$ entonces $\text{ord}_m(a)$ divide a k.
- $\text{ord}_m(a)$ divide a $\phi(m)$.
- Para toda s, $a^s \equiv a^r(\text{mod } m)$, donde $s = q \cdot \text{ord}_m(a) + r$, $0 \leq r < \text{ord}_m(a)$, es decir, los *exponentes* de las potencias de a bajo el módulo m son calculadas módulo $t = \text{ord}_m(a)$.

Sea $m \geq 3$. Si g es un entero tal que $\text{MCD}(g, m) = 1$ y si $\text{ord}_m(g) = \phi(m)$ entonces g es denominada la *raíz primitiva de* m. Del lema 4.8, se tiene que g, g^2,$\cdots$, $g^{\phi(m)}$ son todas distintas módulo m, es decir, $\{g, g^2, g^3, \cdots, g^{\phi(m)}\}$ constituye el SRR para el módulo m.

Sea $1 \leq x < \phi(m)$. De

$$(g^x + (m-1))(1 + g^x + g^{2x} + g^{3x} + \cdots + g^{(\phi(m)-2)x} + g^{(\phi(m)-1)x})$$

$$= m(1 + g^x + g^{2x} + \cdots + g^{(\phi(m)-1)x}) \equiv 0(\text{mod } m)$$

y dado que $(g^x + (m-1)) \not\equiv 0(\text{mod } m)$, se tiene que

$$(1 + g^x + g^{2x} + \cdots + g^{(\phi(m)-1)x}) \equiv 0(\text{mod } m) \tag{4.22}$$

Hemos visto entonces que g tiene propiedades análogas a las de la raices de la unidad en el caso de variable compleja[2]. A continuación detallaremos como, dado un entero N, encontrar el correspondiente valor de m que nos facilite determinar las N raices de la unidad.

Sea $N = 2^n$, para alguna $n \geq 1$. Elijamos un entero $\alpha = 2^k$ para alguna $k \geq 1$. Definamos

$$m = \alpha^{N/2} + 1 = 2^{k2^{n-1}} + 1. \tag{4.23}$$

Dado que α es par y m es impar, $\text{MCD}(\alpha, m) = 1$, y por lo tanto $\alpha \in SRR \pmod m$. De la ecuación (4.23), dado que

$$-1 \equiv \alpha^{N/2} (\text{mod } m), \tag{4.24}$$

tenemos que

$$1 \equiv \alpha^{N} (\text{mod } m), \tag{4.25}$$

Por lo tanto, dado que N es el entero menor que cumple (4.25), por definición

$$\text{ord}_m(\alpha) = N \tag{4.26}$$

En consecuencia, $\alpha, \alpha^2, \cdots, \alpha^N$ son todas distintas y

$$\sum_{i=1}^{N} \alpha^{ix} \equiv \begin{cases} 0 \pmod m & \text{si } 1 \leq x < N, \\ N \pmod m & \text{si } x \text{ es un múltiplo de } N. \end{cases} \tag{4.27}$$

Más aun, dado que $N = 2^n$, de (4.25), se tiene que

$$N^{-1} \equiv 2^{-n} \equiv 2^{Nk-n} \pmod m. \tag{4.28}$$

En conclusión, el valor de m dado en (4.23) nos permite obtener las N raices de la unidad como potencias de α.

4.4.2. Transformada rápida de Fourier

Sea $N = 2^n$, $\alpha = 2^k$. Definamos

$$m = \alpha^{N/2} + 1.$$

Definamos las matrices de tamaño $N \times N$,

$$\mathbf{F} = [\alpha^{ij} (\text{mod } m)] \tag{4.29}$$

y

$$\overline{\mathbf{F}} = [\alpha^{-ij} (\text{mod } m)] \tag{4.30}$$

donde $0 \leq i, j \leq (N-1)$. Entonces se cumple lo siguiente

[2]Para más detalles, veáse [54], pp.186-193

Lema 4.9

$$\mathbf{F}^{-1} = (N^{-1}(\text{mod } m))\overline{\mathbf{F}} \tag{4.31}$$

Demostración
Sea

$$(N^{-1}(\text{mod } m))\mathbf{F}\overline{\mathbf{F}} = [I_{ip}],$$

donde

$$I_{ip} = N^{-1}\sum_{j=0}^{N-1}\alpha^{ij}\alpha^{-jp}(\text{mod } m) = N^{-1}\sum_{j=0}^{N-1}\alpha^{(i-p)j}(\text{mod } m).$$

Si $i = p$, entonces $I_{ii} = 1$ para toda $0 \leq i \leq N-1$. Para $i \neq p$, si $l = i - p$, entonces l no es múltiplo de N, y dado que $\alpha^l = \alpha^{l(\text{mod } N)}$ se tiene

$$I_{ip} = N^{-1}\sum_{j=0}^{N-1}\alpha^{lj} \equiv 0(\text{mod } m),$$

y por lo tanto $[I_{ip}]$ es la matriz identidad y se cumple el lema. □

La matriz inversa $\mathbf{F}^{-1}$, puede obtenerse a partir de $\mathbf{F}$ mediante cierta permutación de filas, como se establece en el siguiente lema.

Lema 4.10 *La primera fila de* $\overline{\mathbf{F}}$ *es igual a la primera fila de* $\mathbf{F}$*. La fila* $(N-i)$ *de* $\overline{\mathbf{F}}$ *es igual a la fila* i *de* $\mathbf{F}$ *para* $1 \leq i \leq N-1$.

Demostración
De la definición se sigue que la primera fila de $\mathbf{F}$ es igual a la primera fila de $\overline{\mathbf{F}}$. Un elemento de la fila i de $\overline{\mathbf{F}}$ es

$$\alpha^{-ij} \text{ para } j = 0, 1, \cdots, N-1.$$

Y dado que $\alpha^{N-i} = \alpha^{-i}$ se tiene que

$$\alpha^{-ij} = \alpha^{(N-i)j} \text{ para } j = 0, 1, \cdots, N-1.$$

Y por lo tanto se cumple el lema. □

Sea $\mathbf{a} = (a_0, a_1, \cdots, a_{N-1})^t$ un vector columna de N números enteros. La *transformada discreta de Fourier* del vector columna $\mathbf{a}$ es el vector columna $f(\mathbf{a})$ definido como

$$f(\mathbf{a}) = \mathbf{F} \centerdot \mathbf{a} \tag{4.32}$$

donde $\mathbf{F}$ es la matriz $N \times N$ definida en (4.29). De forma similar, la *transformada discreta inversa de Fourier* del vector columna $\mathbf{a}$ es el vector columna $f^{-1}(\mathbf{a})$ definido como

$$f^{-1}(\mathbf{a}) = \mathbf{F}^{-1} \centerdot \mathbf{a} \tag{4.33}$$

donde $\mathbf{F}^{-1}$ está definido por el lema 4.9.

Dado el vector columna $\mathbf{a} = (a_0, a_1, \cdots, a_{N-1})^t$ definamos el polinomio $p(x)$ que tiene como coeficientes a los elementos del vector (a), esto es,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1} \tag{4.34}$$

Si $f_j(\mathbf{a})$ es el j-ésimo elemento de $f(\mathbf{a})$, entonces para $0 \leq j \leq N-1$

$$f_j(\mathbf{a}) = \sum_{r=0}^{N-1} a_r \alpha^{jr} = p(\alpha^j), \tag{4.35}$$

es decir, calcular la transformada de Fourier del vector $\mathbf{a}$ es equivalente a evaluar el polinomio $p(x)$ en los puntos $1, \alpha, \alpha^2, \cdots, \alpha^{N-1}$. Debido a que, como es bastante conocido, un polinomio de grado $N-1$ puede ser representado de forma única por el conjunto de sus N coeficientes o por los valores que toma en N puntos diferentes, puede decirse que la transformada discreta de Fourier transforma al polinomio $p(x)$ de su representación de *coeficientes* a su representación de *valores*.

Reescribiendo $p(x)$ en (4.34) en la forma de Horner como

$$p(x) = (\cdots((a_{N-1}x + a_{N-2})x + a_{N-3})x + \cdots + a_1)x + a_0,$$

es fácil ver que se requieren $N-1$ multiplicaciones módulo m y $N-1$ sumas módulo m para evaluar $p(x)$ en un punto dado. Por lo tanto, dado el vector columna $\mathbf{a}$ y el conjunto de las N raices de la unidad, calcular $f(\mathbf{a})$ toma $N(N-1)$ multiplicaciones y $N(N-1)$ sumas de números enteros módulo m. Entonces, al cálculo de $f(\mathbf{a})$ requiere de $O(N^2)$ operaciones.

Dados N procesadores, como $p(\alpha^j)$ para $j = 0, \cdots, N-1$ puede ser calculado de manera independiente, asignando el cálculo de $p(\alpha^j)$ al procesador j, puede verse que $f(\mathbf{a})$ puede calcularse en un tiempo $O(N)$. De manera alterna, dado que el j-ésima componente de $f(\mathbf{a})$ es el producto interno de la j-ésima fila de $\mathbf{F}$ y el vector columna $\mathbf{a}$, se puede deducir fácilmente que $f(\mathbf{a})$ puede calcularse usando el algoritmo de abanico asociativo en $O(\log N)$ pasos, suponiendo que hay N^2 procesadores disponibles. Ambas estrategias requieren de un total de $O(N^2)$ operaciones.

A continuación veremos una técnica recursiva, denominada la *transformada rápida de Fourier*, la cual permite evaluar $f(\mathbf{a})$ en paralelo en $O(\log N)$ pasos utilizando únicamente $O(N)$ procesadores.

Sea $N = 2^n$ y $n \geq 0$. Sea

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$$

Ahora definimos recursivamente un árbol binario de polinomios con $p(x)$ como raíz. En el nivel 1, tendremos los polinomios $p_e(x)$ y $p_o(x)$ de grado $N/2 - 1$ definidos como

$$\begin{aligned} p_e(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{N-2} x^{N/2-1} \\ &= \sum_{j=0}^{N/2-1} a_{2j} x^j \end{aligned}$$

y

$$\begin{aligned} p_o(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{N-1} x^{N/2-1} \\ &= \sum_{j=0}^{N/2-1} a_{2j+1} x^j. \end{aligned}$$

Obsérvese que $p_e(x)$ contiene los coeficientes cuyos índices son múltiplos pares de 2^0, mientras que $p_o(x)$ contiene los de índices son múltiplos impares.

Definamos ahora dos polinomios $p_{ee}(x)$ y $p_{eo}(x)$ de grado $(N/4-1)$ como sigue.

$$\begin{aligned} p_{ee}(x) &= a_0 + a_4 x + a_8 x^2 + \cdots + a_{N-4} x^{N/4-1} \\ &= \sum_{j=0}^{N/4-1} a_{4j} x^j \end{aligned}$$

y

$$\begin{aligned} p_{eo}(x) &= a_2 + a_6 x + a_{10} x^2 + \cdots + a_{N-2} x^{N/4-1} \\ &= \sum_{j=0}^{N/4-1} a_{4j+1} x^j \end{aligned}$$

,es decir , $p_{ee}(x)$ contiene los coeficientes cuyos índices son múltiplo par de 2^1, mientras que $p_{eo}(x)$ los que son múltiplos impares. De forma similar, definimos

$$p_{oe}(x) = \sum_{j=0}^{N/4-1} a_{4j+1} x^j,$$

y

$$p_{oo}(x) = \sum_{j=0}^{N/4-1} a_{4j+3}x^j.$$

El nivel 2 del árbol está formado por los polinomios $p_{ee}(x)$, $p_{eo}(x)$, $p_{oe}(x)$ y $p_{oo}(x)$. En el nivel i, hay 2^i polinomios de grado $(N/2^i - 1)$. Las hojas en el nivel n contienen los polinomios de grado 0 que resultan de la permutación de los datos de entrada.

Para obtener la permutación de los datos en las hojas, definamos primero

$$\rho : \{0,1\}^n \to \{0,1\}^n,$$

donde

$$\rho(b_n b_{n-1} \cdots b_2 b_1) = b_1 b_2 \cdots b_{n-1} b_n,$$

es decir, ρ es una permutación de *inversión de bits*. Por ejemplo, si $x = (11010)_2 = (26)_{10}$, entonces $\rho(x) = (01011)_2 = (11)_{10}$.

Si etiquetamos las 2^n hojas del árbol de polinomios en el orden normal, comenzando con 0 en la hoja más a la izquierda, puede verificarse que la hoja i tiene el dato $a_{\rho(i)}$ para $i = 0, \cdots, N-1$.

De la definición de los polinomios tenemos que

$$\begin{aligned} p(x) &= p_e(x^2) + xp_o(x^2) \\ p(-x) &= p_e(x^2) - xp_o(x^2). \end{aligned} \tag{4.36}$$

A partir de esta relación, se puede determinar una estrategia para la evaluación de $p(x)$. Evaluamos primero los dos polinomios de grado $(N/2 - 1)$, $p_e(x)$ y $p_o(x)$ en el punto α^{2j} para $j = 0, \cdots, N/2$. Entonces, los valores del polinomio de grado $N-1$, $p(x)$ en los pares simétricos de puntos α^j y $\alpha^{N/2+j}$ pueden obtenerse con una suma, una multiplicación y una resta. Dado que

$$\begin{aligned} p_e(x) &= p_{ee}(x^2) + xp_{eo}(x^2) \\ p_e(-x) &= p_{ee}(x^2) - xp_{eo}(x^2). \end{aligned} \tag{4.37}$$

se tiene que los valores de $p_e(x)$ en los pares simétricos de raices, α^{2j} y $\alpha^{N/2+2j}$, puede obtenerse evaluando $p_{ee}(x)$ y $p_{eo}(x)$ en α^{4j} y realizando una multiplicación, una suma y una resta. De forma similar se aplica este proceso recursivo a $p_o(x)$ y $p_o(-x)$.

En general, los valores de los 2^k polinomios de grado $(N/2^k - 1)$ en el nivel k del árbol, evaluados en los puntos simétricos $\alpha^{2^k j}$ y $\alpha^{N/2+2^k j}$, pueden obtenerse evaluando 2^{k+1} polinomios de grado $(N/2^{k+1} - 1)$ en $\alpha^{2^{k+1} j}$, para $j = 0, 1, \cdots, N/2^{k+1}$.

Sea $M(N)$ el número total de multiplicaciones modulares y $S(N)$ el número total de sumas (y restas) modulares requeridas para calcular $f(\mathbf{a})$ con el algoritmo anterior. Como el algoritmo es recursivo, del análisis del nivel 0, tenemos que

$$M(N) = 2M(N/2) + N/2,$$

y

$$S(N) = 2S(N/2) + N,$$

donde $M(1) = S(1) = 0$. Resolviendo las recurrencias anteriores, se tiene

$$M(N) = \frac{1}{2} N \log N$$

y

$$S(N) = N \log N.$$

Si tenemos $N = 2^n$ procesadores, sin considerar tiempos de comunicación, los cálculos pueden arreglarse de forma tal que cada procesador realice uniformemente una multiplicación y una suma modulares, en paralelo, en cada nivel. En tal caso, todo el esquema requiere un tiempo total $O(\log N)$.

El cálculo de la transformada inversa puede hacerse de acuerdo al lema 4.10. Sea $\mathbf{a} = (a_0, a_1, \cdots, a_{N-1})^t$ un vector columna de números enteros. La transformada inversa de Fourier $f^{-1}(\mathbf{a})$ pude obtenerse de la siguiente manera. Calcular primero la transformada de Fourier $f(\mathbf{a})$ de $\mathbf{a}$. Entonces

$$f_0^{-1}(\mathbf{a}) = (N^{-1} (\text{mod } m)) f_0(\mathbf{a}) \tag{4.38}$$

y para $1 \leq k \leq N-1$

$$f_k^{-1}(\mathbf{a}) = (N^{-1} (\text{mod } m)) f_{N-k}(\mathbf{a}) \tag{4.39}$$

donde $f_j(\mathbf{a})$ y $f_j^{-1}(\mathbf{a})$ son los j-ésimos elementos de los vectores $f(\mathbf{a})$ y $f^{-1}(\mathbf{a})$, respectivamente. Entonces, calcular la transformada inversa requiere adicionalmente de N inversiones.

En conclusión, tenemos el siguiente resultado.

Teorema 4.11 *Sea $N = 2^n$ y $\mathbf{a} = (a_0, a_1, \cdots, a_{N-1})^t$ un vector de números enteros. Dados N procesadores, tanto la transformada de Fourier $f(\mathbf{a})$ como la transformada de Fourier inversa $f^{-1}(\mathbf{a})$ pueden ser calculadas cada una en $O(\log N)$ pasos paralelos requiriendo un total de $O(N \log N)$ operaciones modulares.*

4.4.3. Convolución de vectores

Sean

$$\begin{aligned} \mathbf{a} &= (a_0, a_1, \cdots, a_{N-1})^t \\ \mathbf{b} &= (b_0, b_1, \cdots, b_{N-1})^t \end{aligned} \tag{4.40}$$

dos vectores columnas. La convolución de $\mathbf{a}$ y $\mathbf{b}$, denotada por $\mathbf{CON}(\mathbf{a}, \mathbf{b})$ es el vector columna

$$\mathbf{CON}(\mathbf{a}, \mathbf{b}) = \mathbf{r} = (r_0, r_1, \cdots, r_{2N-1})^t$$

donde para $0 \leq i < N$

$$r_i = \sum_{j=0}^{i} a_j b_{i-j}, \tag{4.41}$$

siendo $a_h = b_h = 0$ para toda $h \geq N$.

Esto es,

$$\begin{aligned} r_0 &= a_0 b_0 \\ r_1 &= a_0 b_1 + a_1 b_0 \\ r_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\ &\vdots \\ r_{N-1} &= a_0 b_{N-1} + a_1 b_{N-2} + \cdots + a_{N-1} b_0 \\ r_N &= a_1 b_{N-1} + a_2 b_{N-2} + \cdots + a_{N-1} b_1 \\ &\vdots \\ r_{2N-3} &= a_{N-1} b_{N-2} + a_{N-2} b_{N-1} \\ r_{2N-2} &= a_{N-1} B_{N-1} \\ r_{2N-1} &= 0 \end{aligned}$$

Puede observarse que

$$r_i = \sum_{\substack{t+s=i \\ (0 \leq t, s \leq N-1)}} a_t b_s \tag{4.42}$$

La anterior definición de la convolución de dos vectores de N elementos, como un vector de $2N$ elementos está motivada por su aplicación a la multiplicación de dos polinomios de grado $(N-1)$. Sin embargo, en ocasiones se requerirá definir la convolución de dos vectores de N elementos como un vector de N elementos. Existen al menos dos posibles formas de definirla.

Sean $\mathbf{a}$ y $\mathbf{b}$ vectores de N elementos tal como se definieron en (4.40). La convolución *doblada positiva*, denotada por $\mathbf{PCON}(\mathbf{a}, \mathbf{b})$ es el vector columna

$$\mathbf{PCON}(\mathbf{a}, \mathbf{b}) = \mathbf{r} = (r_0, r_1, \cdots, r_{N-1})^t$$

donde, para $0 \leq i < N$,

$$r_i = \sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{N-1} a_j b_{N+i-j}.$$

Esto es,

$$\begin{aligned}
r_0 &= a_0b_0 + a_1b_{N-1} + a_2b_{N-2} + \cdots + a_{N-1}b_1 \\
r_1 &= a_0b_1 + a_1b_0 + a_2b_{N-1} + \cdots + a_{N-1}b_2 \\
r_2 &= a_0b_2 + a_1b_1 + a_2b_0 + a_3b_{N-1} + \cdots + a_{N-1}b_3 \\
&\vdots \\
r_{N-1} &= a_0b_{N-1} + a_1b_{N-2} + a_2b_{N-3} + \cdots + a_{N-1}b_0
\end{aligned}$$

Puede verificarse que

$$r_i = \sum_{\substack{t+s \equiv i \pmod N \\ 0 \leq t,s < N}} a_t b_s \tag{4.43}$$

De manera similar, definiremos a la convolución *doblada negativa*, denotada por $\mathbf{NCON}(\mathbf{a}, \mathbf{b})$, como el vector columna

$$\mathbf{NCON}(\mathbf{a}, \mathbf{b}) = \mathbf{r} = (r_0, r_1, \cdots, r_{2N-1})^t$$

donde, para $0 \leq i < N$,

$$r_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{N-1} a_j b_{N+i-j}.$$

4.4.4. Producto de polinomios

Sean $p(x)$ y $q(x)$ dos polinomios de grado $N-1$, donde

$$\begin{aligned}
p(x) &= \sum_{i=0}^{N-1} a_i x^i \\
q(x) &= \sum_{i=0}^{N-1} b_i x^i
\end{aligned} \tag{4.44}$$

Entonces, el producto polinomial $r(x)$ está dado por

$$r(x) = p(x) \cdot q(x) = \sum_{i=0}^{2N-1} r_i x^i, \tag{4.45}$$

donde $r_{2N-1} = 0$ y

$$r_i = \sum_{j=0}^{i} a_j b_{i-j} \tag{4.46}$$

Comparando (4.46) con (4.41), podemos observar que los coeficientes del producto polinomial pueden obtenerse como la convolución de los vectores $\mathbf{a}$ y $\mathbf{b}$ formados por los coeficientes de los polinomios $p(x)$ y $q(x)$, respectivamente. Sin embargo, el producto de componentes de $f(\mathbf{a})$ y $f(\mathbf{b})$ proporciona valores de $r(x)$ en solamente N puntos si bien $r(x)$ es un polinomio de grado $2N-2$. Este problema se resuelve extendiendo $p(x)$ y $q(x)$ al grado $(2N-1)$ añadiendo coeficientes *cero* a los términos con grado N al $2N-1$, y evaluandolos en las $2N$ raices de la unidad. Por lo tanto, definiendo

$$\begin{aligned} \underline{\mathbf{a}} &= (a_0, a_1, \cdots, a_{N-1}, 0, 0, \cdots, 0)^t \\ \underline{\mathbf{b}} &= (b_0, b_1, \cdots, b_{N-1}, 0, 0, \cdots, 0)^t. \end{aligned} \tag{4.47}$$

Sean $f(\underline{\mathbf{a}})$ y $f(\underline{\mathbf{b}})$ las transformadas de Fourier de $\underline{\mathbf{a}}$ y $\underline{\mathbf{b}}$, podemos establecer el siguiente teorema.

Teorema 4.12 *Si $\underline{\mathbf{a}}$ y $\underline{\mathbf{b}}$ son los vectores descritos en (4.47) entonces*

$$\mathbf{CON}(\underline{\mathbf{a}}, \underline{\mathbf{b}}) = f^{-1}(f(\underline{\mathbf{a}}) \times f(\underline{\mathbf{b}})), \tag{4.48}$$

donde $\times$ denota la multiplicación de vectores componente a componente.

Demostración
Como $a_j = b_j = 0$ para $N \leq j \leq 2N-1$, tenemos, por definición que el elemento $f_i(\mathbf{a})$ del vector $f(\mathbf{a})$ es

$$f_i(\mathbf{a}) = \sum_{t=0}^{N-1} a_t \alpha^{it}.$$

De manera similar,

$$f_i(\mathbf{b}) = \sum_{s=0}^{N-1} b_s \alpha^{is}.$$

Por lo tanto, de la definición de la transformada inversa de Fourier, tenemos

$$\begin{aligned} r_l &= \frac{1}{N}\sum_{i=0}^{2N-1} f_i(\underline{\mathbf{a}})f_i(\underline{\mathbf{b}})\alpha^{-li} \\ &= \frac{1}{N}\sum_{i=0}^{2N-1}\sum_{t=0}^{N-1}\sum_{s=0}^{N-1} a_t b_s \alpha^{(t+s-l)i} \\ &= \sum_{t=0}^{N-1}\sum_{s=0}^{N-1} a_t b_s \left\{\frac{1}{N}\sum_{i=0}^{2N-1}\alpha^{(t+s-l)i}\right\} \end{aligned} \tag{4.49}$$

donde $1/N$ denota a $N^{-1}(\text{mod } m)$. De (4.27), tenemos que el término entre llaves es *uno* cuando $t+s=l$ y *cero* cuando $t+s\neq l$. Por lo tanto,

$$r_l = \sum_{\substack{t+s=l \\ 0\leq t,s<N}} a_t b_s$$

y se cumple el teorema. □

El cálculo de la convolución doblada positiva $\mathbf{PCON}(\mathbf{a},\mathbf{b})$ puede realizarse de forma similar.

Teorema 4.13 *Sean* $\mathbf{a}$ *y* $\mathbf{b}$ *los vectores definidos en (4.40). Entonces*

$$\mathbf{PCON}(\mathbf{a},\mathbf{b}) = f^{-1}(f(\mathbf{a})\times f(\mathbf{b})),$$

donde $\times$ *denota la multiplicación de vectores componente a componente.*

Demostración
De la definición,

$$r_l = \sum_{t=0}^{N-1}\sum_{s=0}^{N-1} a_t b_s \left\{\frac{1}{N}\sum_{i=0}^{N-1}\alpha^{(t+s-l)i}\right\}$$

donde $1/N$ denota a $N^{-1}(\text{mod } m)$. El término entre llaves es igual a *uno* si $t+s\equiv l(\text{mod } N)$ y *cero* si $t+s\not\equiv l(\text{mod } N)$. Por lo tanto,

$$r_l = \sum_{\substack{t+s\equiv l(\text{mod } N) \\ 0\leq t,s<N}} a_t b_s$$

y el teorema queda demostrado. □

El cálculo de la convolución doblada negativa es un poco más complejo. Sean α y β las N y $2N$ raices de la unidad, respectivamente. [3] Entonces, $\beta^2 \equiv \alpha (\mathrm{mod}\ m)$ y $\beta^{N+k} \equiv -\beta^k (\mathrm{mod}\ m)$ para $0 \leq k < N$. Dados los vectores $\mathbf{a}$ y $\mathbf{b}$, definidos en (4.40), definamos

$$\begin{aligned} \underline{\mathbf{a}} &= (a_0, \beta a_1, \beta^2 a_2, \cdots, \beta^{N-1} a_{N-1})^t \\ \underline{\mathbf{b}} &= (b_0, \beta b_1, \beta^2 b_2, \cdots, \beta^{N-1} b_{N-1})^t \end{aligned} \tag{4.50}$$

y sea

$$\mathbf{NCON}(\underline{\mathbf{a}}, \underline{\mathbf{b}}) = \mathbf{r} = (r_0, \beta r_1, \beta^2 r_2, \cdots, \beta^{N-1} r_{N-1})^t \tag{4.51}$$

Teorema 4.14 *Si* $\underline{\mathbf{a}}$ *y* $\underline{\mathbf{b}}$ *son los vectores descritos en (4.50) entonces*

$$\mathbf{NCON}(\underline{\mathbf{a}}, \underline{\mathbf{b}}) = f^{-1}(f(\underline{\mathbf{a}}) \times f(\underline{\mathbf{b}})), \tag{4.52}$$

donde $\times$ *denota la multiplicación de vectores componente a componente.*

Demostración
La componente i del vector $f(\underline{\mathbf{a}})$ está dada por

$$f_i(\underline{\mathbf{a}}) = \sum_{t=0}^{N-1} \beta^t a_t \alpha^{it}.$$

De manera similar,

$$f_i(\underline{\mathbf{b}}) = \sum_{s=0}^{N-1} \beta^s b_s \alpha^{is}.$$

Entonces,

$$\begin{aligned} \beta^h r_h &= \sum_{t=0}^{N-1} \sum_{s=0}^{N-1} \beta^{t+s} a_t b_s \left\{ \frac{1}{N} \sum_{i=0}^{N-1} \alpha^{(t+s-h)i} \right\} \\ &= \sum_{\substack{t+s \equiv h (\mathrm{mod}\ N) \\ 0 \leq t,s < N}} \beta^{t+s} a_t b_s. \end{aligned}$$

Dado que $\beta^{N+k} \equiv -\beta^k (\mathrm{mod}\ m)$ para $0 \leq k < N$,

$$\beta^h r_h = \sum_{j=0}^{h} \beta^h a_j b_{h-j} - \sum_{j=h+1}^{N-1} \beta^h a_j b_{N+h-j}.$$

[3] En algunos casos, la β puede no estar bien definida. Por ejemplo, si $m = 2^{N/2-1} + 1$ es un primo. Para más detalles, veáse [54], pp. 201-202.

Cancelando en la expresión anterior a la β^h queda demostrado el teorema. □

A partir de los teoremas demostrados en esta sección y del análisis de complejidad para la transformada rápida de Fourier, podemos llegar a la siguiente conclusión:

Teorema 4.15 *Dados dos vectores compuestos de N números enteros, $\mathbf{a} = (a_0, a_1, \cdots, a_{N-1})^t$ y $\mathbf{b} = (b_0, b_1, \cdots, b_{N-1})^t$, se requiere de $O(\log N)$ operaciones aritméticas modulares para calcular* $\mathbf{CON}(\mathbf{a}, \mathbf{b})$, $\mathbf{PCON}(\mathbf{a}, \mathbf{b})$ *o* $\mathbf{NCON}(\mathbf{a}, \mathbf{b})$.

4.4.5. Algoritmo de Schonhage-Strassen

Dados dos enteros a y b de N bits, el algoritmo recursivo de Schonhage-Strassen [86] permite calcular $a \times b \pmod{(2^N + 1)}$ en $O(\log N)$ pasos (medido en unidades de tiempo t_b, el tiempo que requiere la ejecución de una operación booleana tal como OR y AND lógicos). El método de Schonhage-Strassen está basado en la convolución usando transformada rápida de Fourier y también en el *teorema chino del residuo.*

Sea $N = 2^n$ y $0 \leq a, b \leq 2^N$ dos enteros de $(N + 1)$ bits. Sea

$$s = (a \times b)(\mathrm{mod}(2^N + 1)) \tag{4.53}$$

Dado que

$$2^N \equiv -1(\mathrm{mod}(2^N + 1)),$$

cuando $a = 2^N$,

$$s \equiv (2^N + 1 - b)(\mathrm{mod}(2^N + 1)) \tag{4.54}$$

y de manera similar, cuando $b = 2^N$, s se obtiene de substituir b con a en (4.54). En vista de lo anterior, se supondrá en adelante que $0 \leq a, b < 2^N$. Sean

$$a = a_N a_{N-1} a_{N-2} \cdots a_2 a_1,$$

$$b = b_N b_{N-1} b_{N-2} \cdots b_2 b_1$$

y

$$N = tm$$

donde

$$t = 2^{\lfloor n/2 \rfloor} = O(N^{1/2}),$$

y

$$m = 2^{\lceil n/2 \rceil} = O(N^{1/2}).$$

Entonces, $m \geq t$ y t divide a m. El primer paso consiste en partir a y b en t grupos de m bits cada uno, esto es,

$$a = \sum_{k=0}^{t-1} A_k 2^{m_k},$$

y

$$b = \sum_{k=0}^{t-1} B_k 2^{m_k},$$

donde, para $0 \leq k \leq t-1$,

$$A_k = a_{(k+1)m} a_{(k+1)m-1} \cdots a_{km+1},$$
$$B_k = b_{(k+1)m} b_{(k+1)m-1} \cdots b_{km+1}.$$

Definiendo los polinomios

$$p(x) = \sum_{k=0}^{t-1} A_k x^k,$$

y

$$q(x) = \sum_{k=0}^{t-1} B_k x^k,$$

es claro que $a = p(2^m)$ y $b = q(2^m)$.
Definamos

$$r(x) = p(x)q(x) = \sum_{j=0}^{2t-1} r_j x^j,$$

donde

$$\begin{aligned}
r_0 &= A_0 B_0 \\
r_1 &= A_0 B_1 + A_1 B_0 \\
r_2 &= A_0 B_2 + A_1 B_1 + A_2 B_0 \\
&\vdots \\
r_{t-1} &= A_0 B_{t-1} + A_1 B_{t-2} + \cdots + A_{t-1} B_0 \\
r_t &= A_1 B_{t-1} + A_2 B_{t-2} + \cdots + A_{t-1} B_1 \\
&\vdots \\
r_{2t-3} &= A_{t-1} B_{t-2} + A_{t-2} B_{t-1} \\
r_{2t-2} &= A_{t-1} B_{t-1} \\
r_{2t-1} &= 0
\end{aligned}$$

Dado que $N = tm$, para $0 \leq \beta < t$,

$$2^{m(\alpha t+\beta)} \equiv \begin{cases} -1 \pmod{(2^N+1)} & \text{si } \alpha > 0, \\ 2^{m\beta} \pmod{(2^N+1)} & \text{si } \alpha = 0. \end{cases} \tag{4.55}$$

Entonces

$$\begin{aligned} r(2^m)(\mathrm{mod}(2^N+1)) &= D_{t-1}2^{(t-1)m} + D_{t-2}2^{(t-2)m} \\ &\quad + \cdots + D_2 2^{2m} + D_1 2^m + D_0, \end{aligned} \tag{4.56}$$

donde, para $0 \leq j \leq t-1$,

$$\begin{aligned} D_j &= r_j - r_{t+j} \\ &= \sum_{h=0}^{j} A_h B_{j-h} - \sum_{h=j+1}^{t-1} A_h B_{t+j-h}. \end{aligned}$$

De la igualdad anterior, se deduce que $r(2^m)(\mathrm{mod}(2^N+1))$ puede calcularse utilizando la convolución negativa. Sin embargo si en este punto se aplican directamente tenemos que hacer t multiplicaciones de enteros de $(N+1)$ bits. Por lo tanto, se requiere mayor análisis para hacer más eficiente el método.

Como A_i y B_j son enteros de de m bits, el producto $A_i B_j$ es un entero de $2m$ bits, es decir, $0 \leq A_i B_j < 2^{2m}$. Dado que r_j es la suma de $(j+1)$ de tales productos, obtenemos que

$$0 \leq r_j < (j+1)2^{2m}$$

y de manera similar,

$$0 \leq r_{t+j} < (t-j-1)2^{2m}.$$

Por lo tanto,

$$-(t-j-1)2^{2m} < D_j < (j+1)2^{2m},$$

y

$$0 \leq |D_j| < t2^{2m}$$

, es decir, D_j solo puede tomar $t2^m$ valores distintos. Y como

$$t2^{2m} = 2^{\lfloor n/2 \rfloor} 2^{2(\lceil n/2 \rceil + 1)} < 2^{2^n} < 2^N + 1,$$

se puede obtener $D_j(\mathrm{mod}\ (2^N+1))$ calculando $D_j(\mathrm{mod}\ t2^{2m})$.

Definiendo

$$\begin{aligned} E_j &\equiv D_j(\text{mod } (2^{2m}+1)) \\ F_j &\equiv D_j(\text{mod } t) \end{aligned} \tag{4.57}$$

Conociendo E_j y F_j se puede determinar D_j [4]como

$$D_j = E_j + (2^{2m}+1)[(F_j - E_j)(\text{mod } t)]. \tag{4.58}$$

Definamos ahora

$$\begin{aligned} \bar{A}_h &\equiv A_h(\text{mod } t) \\ \bar{B}_h &\equiv B_h(\text{mod } t). \end{aligned} \tag{4.59}$$

Sea $q = \lfloor n/2 \rfloor$. Entonces, como $t = 2^q$, $\bar{A}_h$ y $\bar{B}_h$ son enteros de q bits. Definamos $\bar{0}$ como una cadena de q ceros, y construyamos los enteros $\bar{A}$ y $\bar{B}$, cada uno de $3tq$ bits, de la siguiente forma

$$\bar{A} = \bar{0}\bar{0}\bar{A}_{t-1}\bar{0}\bar{0}\bar{A}_{t-2}\bar{0}\bar{0}\cdots\bar{0}\bar{0}\bar{A}_1\bar{0}\bar{0}\bar{A}_0,$$

y

$$\bar{B} = \bar{0}\bar{0}\bar{B}_{t-1}\bar{0}\bar{0}\bar{B}_{t-2}\bar{0}\bar{0}\cdots\bar{0}\bar{0}\bar{B}_1\bar{0}\bar{0}\bar{B}_0,$$

donde

$$\bar{A} = \sum_{h=0}^{t-1} \bar{A}_h 2^{(3q)h}$$

y

$$\bar{B} = \sum_{h=0}^{t-1} \bar{B}_h 2^{(3q)h}.$$

Dado que $\bar{A}_h\bar{B}_h$ es de $2q$ bits, en el producto $\bar{A}\bar{B}$ todos los términos producto $\bar{A}_k\bar{B}_h$ para toda $h, k = 0, 1, \cdots, t-1$, están confinados en un bloque de $3q$ bits. Entonces, si

$$\bar{r} = \bar{A}\bar{B} = \sum_{h=0}^{2t-1} \bar{r}_j 2^{(3q)h},$$

tenemos que

$$\bar{r}_j = \sum_{h=0}^{j} \bar{A}_h \bar{B}_{j-h}. \tag{4.60}$$

[4]Veáse la demostración en [54], pp. 203-205

Combinando las ecuaciones de (4.56) a (4.60) tenemos que

$$\begin{aligned} F_j &\equiv D_j(\text{mod } t) \\ &\equiv (\bar{r}_j - \bar{r}_{t+j})(\text{mod } t) \\ &\equiv \left[\sum_{h=0}^{j} \bar{A}_h \bar{B}_h - \sum_{h=j+1}^{t-1} \bar{A}_h \bar{B}_{t+j-h}\right](\text{mod } t) \end{aligned} \tag{4.61}$$

El producto $\bar{A}_h \bar{B}_h$ puede encontrarse, utilizando el algoritmo de Melhorn-Preparata, en $O(\log 3qt) = O(\log N)$ pasos usando a lo más $(3tq)^{1,59} = O(N \log N)$ operaciones a nivel de bit. Mientras que el cálculo de $E_j \equiv D_j(\text{mod } (2^{2m}+1))$ puede realizarse utilizando la convolución doblada negativa.

Sea $\alpha = 2^{4m/t}$ y $M = 2^{2m} + 1$. Dado $M = \alpha^{1/2} + 1$ se tiene que $\alpha, \alpha^2, \cdots, \alpha^t$ definen las t raices de la unidad módulo M. Definamos $\beta = 2^{2m/t}$, donde $\beta^2 = \alpha$, esto es, β es la raíz $2t$ de la unidad módulo M. Definamos

$$\begin{aligned} \bar{\mathbf{A}} &= (A_0, \beta A_1, \beta^2 A_2, \cdots, \beta^{N-1} A_{N-1})^t \\ \bar{\mathbf{B}} &= (B_0, \beta B_1, \beta^2 B_2, \cdots, \beta^{N-1} B_{N-1})^t \\ \bar{\mathbf{E}} &= (E_0, \beta E_1, \beta^2 E_2, \cdots, \beta^{N-1} E_{N-1})^t \end{aligned}$$

entonces

$$\bar{\mathbf{E}} = \mathbf{NCON}(\bar{\mathbf{A}}, \bar{\mathbf{B}}) = f^{-1}(f(\bar{\mathbf{A}} \times \bar{\mathbf{B}})).$$

De acuerdo al teorema 4.11, el cálculo de la transformada de Fourier de $\bar{A}$ y $\bar{B}$ y el de la transformada inversa requieren, cada una, $O(\log t) = O(\log N)$ pasos y $O(tm \log t) = O(N \log N)$ operaciones. Las t multiplicaciones involucradas en el cálculo de la convolución pueden realizarse en paralelo, con invocaciones recursivas. Después E_j puede recuperarse multiplicando $\beta^j E_j$ por $\beta^{-j}(\text{mod } (2^{2m}+1))$. Estas operaciones pueden hacerse en paralelo en $O(\log N)$ pasos y $O(N \log N)$ operaciones a nivel de bit.[5]

Una vez que se han calculado los valores de D_j de (4.58) se puede proceder al cálculo de $r(2^m)(\text{mod}(2^N+1))$ multiplicando D_j por 2^{jm}, sumando y calculando $(\text{mod}(2^N+1))$. Todos estos cálculos requieren de $O(\log N)$ pasos en $O(N \log N)$ operaciones a nivel de bit.

En resumen, al algoritmo es el siguiente:

Dados $a = a_N a_{N-1} \cdots a_2 a_1$ y $b = b_N b_{N-1} \cdots b_2 b_1$ encontrar $s = a \times b(\text{mod}(2^N+1))$.

[5]Dado que $x(\text{mod } m)$ puede ser calculado en $O(\log N)$ operaciones a nivel de bit si el número de dígitos de la base de x está determinado de antemano. Veáse la demostración en [54], pag. 192.

1. Hacer $t = 2^{\lfloor n/2 \rfloor}$ y $m = 2^{\lceil n/2 \rceil}$. Partir los bits de a en bloques de t bits $A_k = a_{(k+1)m} a_{(k+1)m-1} \cdots a_{km+1}$, para $k = 0, \cdots, t-1$. De modo similar definir B_k. Sea $A = (A_0, A_1, \cdots, A_{t-1})$ y $B = (B_0, B_1, \cdots, B_{t-1})$ se define $D = (D_0, D_1, \cdots, D_{t-1})$ como en (4.56).

2. Calcular $F_j \equiv D_j (\mathrm{mod}\ t)$ utilizando (4.61).

3. Calcular $E_j \equiv D_j (\mathrm{mod}(2^{2m} + 1))$ mediante la convolución doblada negativa y llamadas recursivas paralelas al algoritmo.

4. Utilizando (4.58), calcular D_j.

5. Calcular $a \times b (\mathrm{mod}(2^N + 1)) = r(2^m) (\mathrm{mod}(2^N + 1))$ utilizando (4.56).

Sean $T(N)$ el tiempo y $S(N)$ el número total de operaciones que requiere este algoritmo. Del análisis previo, tenemos que

$$T(N) = T(2m) + O(\log N) \tag{4.62}$$

y

$$S(N) = t \times S(2m) + O(N \log N). \tag{4.63}$$

Como $m \leq 2\sqrt{N}$, puede demostrarse por inducción que $T(N) = O(\log N)$. Para encontrar $S(N)$, definamos $S'(N) = S(N)/N$. Entonces, para una N suficientemente grande, la ecuación (4.63) se convierte en

$$S'(N) \leq 2S'(2m) + c \log N, \tag{4.64}$$

para alguna constante c. Puede verificarse que

$$S'(N) = O((\log N)(\log \log N))$$

de donde obtenemos que

$$S(N) = O(N(\log N) \log \log N).$$

En resumen, llegamos al siguiente resultado:

Teorema 4.16 *El producto* $(\mathrm{mod}\ (2^N + 1))$ *de dos enteros de* N *bits puede obtenerse mediante el algoritmo de Schonhage-Strassen en* $O(\log N)$ *pasos utilizando* $O(N(\log N) \log \log N)$ *elementos lógicos.*