Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic Toffoli gate count

https://youtu.be/c0sWL97W3wE

장경배

한성대학교
HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Paper

Quantum Inf Process (2015) 14:2373–2386
DOI 10.1007/s11128-015-0993-1

CrossMark

Quantum circuits for $\mathbb{F}_{2^n}$-multiplication with subquadratic gate count

Shane Kepley[1] · **Rainer Steinwandt**[1]

Received: 10 December 2014 / Accepted: 8 April 2015 / Published online: 12 May 2015

Abstract One of the most cost-critical operations when applying Shor's algorithm to binary elliptic curves is the underlying field arithmetic. Here, we consider binary fields $\mathbb{F}_{2^n}$ in polynomial basis representation, targeting especially field sizes as used in elliptic curve cryptography. Building on Karatsuba's algorithm, our software implementation automatically synthesizes a multiplication circuit with the number of T-gates being bounded by $7 \cdot n^{\log_2(3)}$ for any given reduction polynomial of degree $n = 2^N$. If an irreducible trinomial of degree n exists, then a multiplication circuit with a total gate count of $\mathcal{O}(n^{\log_2(3)})$ is available.

2015

Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic Toffoli gate count

Iggy van Hoof

Technische Universiteit Eindhoven
i.v.hoof@student.tue.nl

Abstract. Multiplication is an essential step in a lot of calculations. In this paper we look at multiplication of 2 binary polynomials of degree at most $n-1$, modulo an irreducible polynomial of degree n with $2n$ input and n output qubits, without ancillary qubits, assuming no errors. With straightforward schoolbook methods this would result in a quadratic number of Toffoli gates and a linear number of CNOT gates. This paper introduces a new algorithm that uses the same space, but by utilizing space-efficient variants of Karatsuba multiplication methods it requires only $O(n^{\log_2(3)})$ Toffoli gates at the cost of a higher CNOT gate count: theoretically up to $O(n^2)$ but in examples the CNOT gate count looks a lot better.

2019

Performance

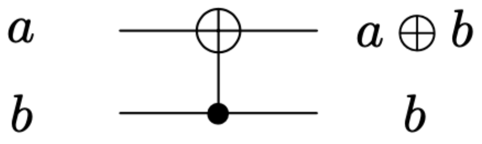
CNOT

Qubit

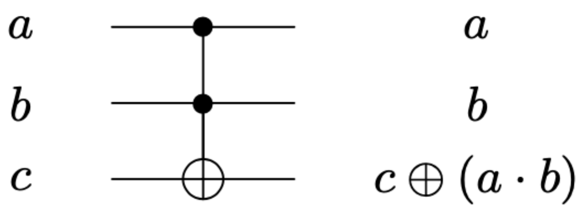
Field size 2^n	Toffoli gates			CNOT gates			qubits		
$n =$	Here	[7]	[9]	Here	[7]	[9]	Here	[7]	[9]
4	9	9	16	49	22	3	12	17	12
16	81	81	256	725	376	45	48	113	48
127	2185	2185	16129	21028	13046	126	381	2433	381
256	6561	6561	65536	66107	57008	765	768	7073	768
n	$O(n^{\log_2 3})$	$O(n^{\log_2 3})$	n^2	$O(n^2)$	$O(n^{\log_2 3})$	$O(n)$	$3n$	$O(n^{\log_2 3})$	$3n$

Table 6: Comparison of this work with the works of Kepley and Steinwandt [7] and Maslov et al. [9] in terms of Toffoli and CNOT gates as well as qubit count.

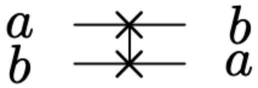
Quantum Background



Circuit 1: The CNOT gate



Circuit 2: The TOF gate



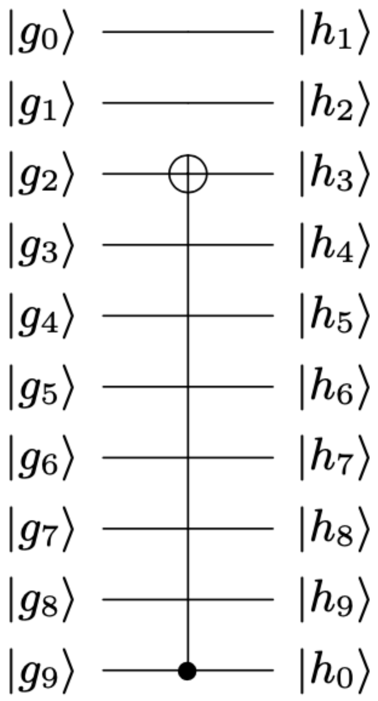
Circuit 3: The swap

$\mathrm{CNOT}(a, b)$ → (a + b, b)

$\mathrm{TOF}(a, b, c)$ → (a, b, c + a*b)

SWAP(a,b) → (b, a)

Basic Arithmetic (1) : MODSHIFT



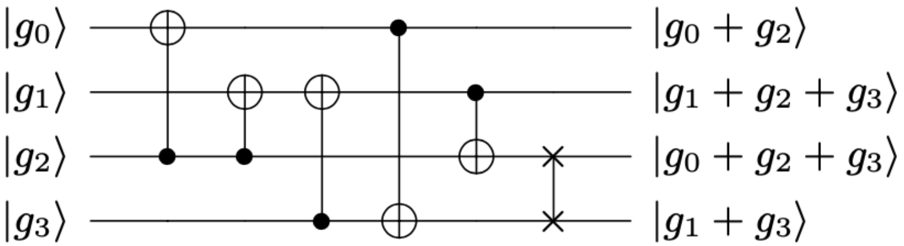
Circuit 4: Binary shift circuit for $\mathbb{F}_{2^{10}}$ with $g_0 + \cdots + g_9x^9$ as the input and $h_0 + \cdots + h_9x^9 = g_9 + g_0x + g_1x^2 + (g_2 + g_9)x^3 + g_3x^4 + \cdots + g_8x^9$ as the output.

modular m(x)

if $m(x) = 1 + x^3 + x^{10}$

즉 $x^{10} = x^3 + 1$

Basic Arithmetic (2) : Algorithm 1



Circuit 5: Multiplication of g by $1 + x^2$ modulo $1 + x + x^4$. Depth 4 and 5 CNOT gates.

multiplication by $1 + x^2$ modulo $1 + x + x^4$ 는 LUP decomposition 을 통해 행렬 Γ 로 표현가능

$$\Gamma = P^{-1} L U$$

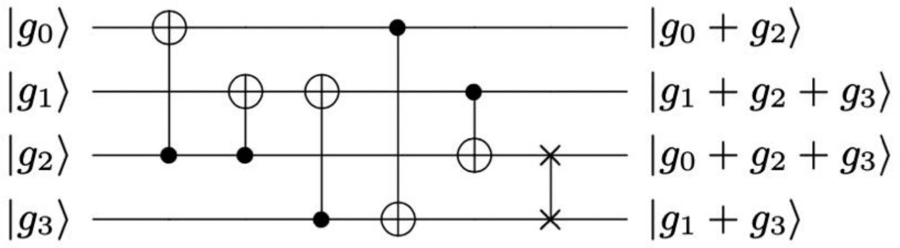
< LUP decomposition >

Basic Arithmetic (2) : Algorithm 1

multiplication by $1 + x^2$ modulo $1 + x + x^4$

LUP decomposition

$$\Gamma = \begin{pmatrix} 1&0&1&0 \\ 0&1&1&1 \\ 1&0&1&1 \\ 0&1&0&1 \end{pmatrix} = P^{-1}LU = \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{pmatrix} \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&1&1&0 \\ 1&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0&1&0 \\ 0&1&1&1 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}$$



Algorithm 1: MULT$_{f(x)}$, from [1]. Reversible algorithm for in-place multiplication by a nonzero constant polynomial $f(x)$ in $\mathbb{F}_2[x]/m(x)$ with $m(x)$ an irreducible polynomial.

Fixed input : A binary LUP-decomposition L, U, P^{-1} for a binary n by n matrix that corresponds to multiplication by the constant polynomial $f(x)$ in the field $\mathbb{F}_2[x]/m(x)$.

Quantum input: A binary polynomial $g(x)$ of degree up to $n-1$ stored in an array G.

Result: G as $f \cdot g$ in the field $\mathbb{F}_2/m(x)$.

```
for i = 0..n − 1                                  // U · G
do
    for j = i + 1..n − 1 do
        if U[i, j] = 1 then
            G[i] ← CNOT(G[i], G[j])
for i = n − 1..0                                  // L · UG
do
    for j = i − 1..0 do
        if L[i, j] = 1 then
            G[i] ← CNOT(G[i], G[j])
for i = 0..n                                      // P^{-1} · LUG
do
    for j = i + 1..n − 1 do
        if P^{-1}[i, j] = 1 then
            SWAP(G[i], G[j])
            SWAP column i and j of P^{-1}
```

Choice of Polynomials

$1 + x^3 + x^4 + x^{19} + x^{20}$ → 108 CNOT gates

$1 + x^3 + x^5 + x^9 + x^{20}$ → 55 CNOT gates

$1 + x^3 + x^{20}$ → 27 CNOT gates

1. coefficient 가 1인 개수가 적어야 함
2. 두번째 높은 지수가 작아야 함

3.1 Parameter set `kem/mceliece348864`

KEM with $m = 12$, $n = 3488$, $t = 64$, $\ell = 256$. Field polynomial $f(z) = z^{12} + z^3 + 1$. Hash function: SHAKE256 with 32-byte output. This parameter set is **proposed and implemented** in this submission.

3.9 Parameter set `kem/mceliece8192128`

KEM with $m = 13$, $n = 8192$, $t = 128$, $\ell = 256$. Field polynomial $f(z) = z^{13} + z^4 + z^3 + z + 1$. Hash function: SHAKE256 with 32-byte output. This parameter set is **proposed and implemented** in this submission.

Quantum Multiplication for binary polynomials

1. Quantum Schoolbook Multiplication

2. Classic Karatsuba multiplication in binary polynomial rings

- input

  $f(x)$, $g(x)$ → size n 의 polynomial

  $h(x)$ → size 2n 의 polynomial

- output

  $h+f\cdot g$

Classic Karatsuba multiplication in binary polynomial rings

각 polynomial 을 다음과 같이 나눈다.

$$\frac{n}{2} \leq k < n$$

$$k = \lceil \frac{n}{2} \rceil$$

$$f = f_0 + f_1 x^k,\ g = g_0 + g_1 x^k$$

$$h = h_0 + h_1 x^k + h_2 x^{2k} + h_3 x^{3k}$$

$$\alpha = f_0 \cdot g_0,\ \beta = f_1 \cdot g_1 \text{ and } \gamma = (f_0 + f_1) \cdot (g_0 + g_1).$$

Karatsuba multiplication

* α, β, γ 또한 f, g 처럼 나눌 수 있음

$$h + f \cdot g = h + \alpha + (\gamma + \alpha + \beta) x^k + \beta x^{2k}$$

$$h+f\cdot g = (h_0+\alpha_0)+(h_1+\alpha_0+\alpha_1+\beta_0+\gamma_0)x^k+(h_2+\alpha_1+\beta_0+\beta_1+\gamma_1)x^{2k}+(h_3+\beta_1)x^{3k}$$

$$h + f \cdot g = h + (1 + x^k)\alpha + x^k \gamma + x^k (1 + x^k)\beta.$$

Algorithm 2 : $\mathrm{MULT1x}_k$.

Algorithm 2: $\mathrm{MULT1x}_k$. Reversible algorithm for multiplication by the polynomial $1 + x^k$.

Fixed input : A constant integer $k > 0$ to indicate part size as well as an integer $n \leq k$ to indicate polynomial size. $\ell = \max(0, 2n - 1 - k)$ is the size of h_2 and $(fg)_1$. In the case of Karatsuba we will have either $n = k$ or $n = k - 1$.

Quantum input: Two binary polynomials $f(x), g(x)$ of degree up to $n - 1$ stored in arrays A and B respectively of size n. A binary polynomial $h(x)$ of degree up to $k + 2n - 2$ stored in array C of size $2k + \ell$.

Result: A and B as input, C as $h + (1 + x^k)fg$

```
1  if n > 1 then
2      C[k..k+ℓ−1] ← CNOT(C[k..k+ℓ−1], C[2k..2k+ℓ−1])
3      C[0..k−1] ← CNOT(C[0..k−1], C[k..2k−1])
4      C[k..2k+ℓ−1] ← KMULT(A[0..n−1], B[0..n−1], C[k..2k+ℓ−1])
5      C[0..k−1] ← CNOT(C[0..k−1], C[k..2k−1])
6      C[k..k+ℓ−1] ← CNOT(C[k..k+ℓ−1], C[2k..2k+ℓ−1])
7  else
8      C[0] ← CNOT(C[0], C[k])
9      C[k] ← TOF(A[0], B[0], C[k])
10     C[0] ← CNOT(C[0], C[k])
```

given $f(x), g(x), h(x)$ calculate $h + f \cdot g$

given $k, f(x), g(x), h(x)$ with $k > \max(\deg(f), \deg(g))$

Line	C in $\mathrm{MULT1x}_k$		
	$C[0..k-1]$	$C[k..2k-1]$	$C[2k..2k+\ell-1]$
1	h_0	h_1	h_2
2	h_0	$h_1 + h_2$	h_2
3	$h_0 + h_1 + h_2$	$h_1 + h_2$	h_2
4	$h_0 + h_1 + h_2$	$h_1 + h_2 + (fg)_0$	$h_2 + (fg)_1$
5	$h_0 + (fg)_0$	$h_1 + h_2 + (fg)_0$	$h_2 + (fg)_1$
6	$h_0 + (fg)_0$	$h_1 + (fg)_0 + (fg)_1$	$h_2 + (fg)_1$

Table 2: Step by step calculation of Algorithm 2.

Algorithm 2 : $\mathrm{MULT1x}_k$

Line	C in $\mathrm{MULT1x}_k$		
	$C[0..k-1]$	$C[k..2k-1]$	$C[2k..2k+\ell-1]$
1	h_0	h_1	h_2
2	h_0	h_1+h_2	h_2
3	$h_0+h_1+h_2$	h_1+h_2	h_2
4	$h_0+h_1+h_2$	$h_1+h_2+(fg)_0$	$h_2+(fg)_1$
5	$h_0+(fg)_0$	$h_1+h_2+(fg)_0$	$h_2+(fg)_1$
6	$h_0+(fg)_0$	$h_1+(fg)_0+(fg)_1$	$h_2+(fg)_1$

Table 2: Step by step calculation of Algorithm 2.

$$(fg)_0 + \{(fg)_0 + (fg)_1\}x^k + (fg)_1 x^{2k}$$

$$\underbrace{f_0g_0 + f_1g_1x^k}_{fg} + \underbrace{f_0g_0x^k + f_1g_1x^{2k}}_{fg\cdot x^k}$$

$$fg = f_0g_0 + f_1g_1x^k.$$

$$= fg + fg\cdot x^k$$

$$= (1+x^k)\cdot fg$$

$$h = h_0 + h_1x^k + h_2x^{2k}$$

$$h_0 + (fg)_0 + (h_1 + (fg)_0 + (fg)_1)x^k + (h_2 + (fg)_1)x^{2k} = h_0 + h_1x^k + h_2x^{2k} + fg + fgx^k$$

$$\rightarrow h + (1+x^k)fg;$$

Algorithm 2 : $\mathrm{MULT1x}_k$.

Algorithm 2: $\mathrm{MULT1x}_k$. Reversible algorithm for multiplication by the polynomial $1 + x^k$.

Fixed input : A constant integer $k > 0$ to indicate part size as well as an integer $n \leq k$ to indicate polynomial size. $\ell = \max(0, 2n - 1 - k)$ is the size of h_2 and $(fg)_1$. In the case of Karatsuba we will have either $n = k$ or $n = k - 1$.

Quantum input: Two binary polynomials $f(x), g(x)$ of degree up to $n - 1$ stored in arrays A and B respectively of size n. A binary polynomial $h(x)$ of degree up to $k + 2n - 2$ stored in array C of size $2k + \ell$.

Result: A and B as input, C as $h + (1 + x^k)fg$

if $n > 1$ **then**
- $C[k..k+\ell-1] \leftarrow \mathrm{CNOT}(C[k..k+\ell-1], C[2k..2k+\ell-1])$
- $C[0..k-1] \leftarrow \mathrm{CNOT}(C[0..k-1], C[k..2k-1])$
- $C[k..2k+\ell-1] \leftarrow \mathrm{KMULT}(A[0..n-1], B[0..n-1], C[k..2k+\ell-1])$
- $C[0..k-1] \leftarrow \mathrm{CNOT}(C[0..k-1], C[k..2k-1])$
- $C[k..k+\ell-1] \leftarrow \mathrm{CNOT}(C[k..k+\ell-1], C[2k..2k+\ell-1])$

else
- $C[0] \leftarrow \mathrm{CNOT}(C[0], C[k])$
- $C[k] \leftarrow \mathrm{TOF}(A[0], B[0], C[k])$
- $C[0] \leftarrow \mathrm{CNOT}(C[0], C[k])$

Result

1. 4n-2 의 CNOT gate
2. depth – 4 당 한번의 algorithm 3 호출

⇩

(n x n) multi

Algorithm 3 : KMULT

Algorithm 3: KMULT. Reversible algorithm for multiplication of 2 polynomials.

Fixed input : A constant integer n to indicate polynomial size and an integer $k < n \leq 2k$ with $k = \lceil \frac{n}{2} \rceil$ for $n > 1$ and $k = 0$ for $n = 1$, to indicate upper and lower half.

Quantum input: Two binary polynomial f, g of degree up to $n-1$ stored in arrays A and B respectively of size n. A binary polynomial h of degree up to $2n-2$ stored in array C of size $2n-1$.

Result: A and B as input, C as $h + fg$

```
1  if n > 1 then
2      C[0..3k−2] ← MULT1x_k(A[0..k−1], B[0..k−1], C[0..3k−2])
3      C[k..2n−2] ← MULT1x_k(A[k..n−1], B[k..n−1], C[k..2n−2])
4      A[0..n−k−1] ← CNOT(A[0..n−k−1], A[k..n−1])
5      B[0..n−k−1] ← CNOT(B[0..n−k−1], B[k..n−1])
6      C[k..3k−2] ← KMULT(A[0..k−1], B[0..k−1], C[k..3k−2])
7      B[0..n−k−1] ← CNOT(B[0..n−k−1], B[k..n−1])
8      A[0..n−k−1] ← CNOT(A[0..n−k−1], A[k..n−1])
9  else
10     C[0] ← TOF(A[0], B[0], C[0])
```

Line	C in MULT1x$_k$		
	$C[0..k-1]$	$C[k..2k-1]$	$C[2k..2k+\ell-1]$
1	h_0	h_1	h_2
2	h_0	$h_1 + h_2$	h_2
3	$h_0 + h_1 + h_2$	$h_1 + h_2$	h_2
4	$h_0 + h_1 + h_2$	$h_1 + h_2 + (fg)_0$	$h_2 + (fg)_1$
5	$h_0 + (fg)_0$	$h_1 + h_2 + (fg)_0$	$h_2 + (fg)_1$
6	$h_0 + (fg)_0$	$h_1 + (fg)_0 + (fg)_1$	$h_2 + (fg)_1$

Table 2: Step by step calculation of Algorithm 2.

$$\alpha = f_0 \cdot g_0,\ \beta = f_1 \cdot g_1 \text{ and } \gamma = (f_0 + f_1) \cdot (g_0 + g_1).$$

Line	C in KMULT			
	$C[0..k-1]$	$C[k..2k-1]$	$C[2k..3k-1]$	$C[3k..2n-2]$
1	h_0	h_1	h_2	h_3
2	$h_0 + \alpha_0$	$h_1 + \alpha_0 + \alpha_1$	$h_2 + \alpha_1$	h_3
3-5	$h_0 + \alpha_0$	$h_1 + \alpha_0 + \alpha_1 + \beta_0$	$h_2 + \alpha_1 + \beta_0 + \beta_1$	$h_3 + \beta_1$
6-8	$h_0 + \alpha_0$	$h_1 + \alpha_0 + \alpha_1 + \beta_0 + \gamma_0$	$h_2 + \alpha_1 + \beta_0 + \beta_1 + \gamma_1$	$h_3 + \beta_1$

Table 3: Step by step calculation of Algorithm 3.

Algorithm 3 : KMULT

Algorithm 3: KMULT. Reversible algorithm for multiplication of 2 polynomials.

Fixed input : A constant integer n to indicate polynomial size and an integer $k < n \leq 2k$ with $k = \lceil \frac{n}{2} \rceil$ for $n > 1$ and $k = 0$ for $n = 1$, to indicate upper and lower half.

Quantum input: Two binary polynomial f, g of degree up to $n - 1$ stored in arrays A and B respectively of size n. A binary polynomial h of degree up to $2n - 2$ stored in array C of size $2n - 1$.

Result: A and B as input, C as $h + fg$

if $n > 1$ **then**

  $C[0..3k-2] \leftarrow \text{MULT1x}_k(A[0..k-1], B[0..k-1], C[0..3k-2])$

  $C[k..2n-2] \leftarrow \text{MULT1x}_k(A[k..n-1], B[k..n-1], C[k..2n-2])$

  $A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$

  $B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$

  $C[k..3k-2] \leftarrow \text{KMULT}(A[0..k-1], B[0..k-1], C[k..3k-2])$

  $B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$

  $A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$

else

  $C[0] \leftarrow \text{TOF}(A[0], B[0], C[0])$

Result

1. 4(n– k) CNOT gate,
2. k x k multi 위한 Algorithm 2 호출 1번
3. (n-k) x (n-k) multi 위한 Algorithm 2 호출 1번
4. k x k multi 하는 자기자신 호출 1번

$n' = \frac{n}{2}$

Algorithm 4 : $\mathrm{MODMULT}$

앞선 algorithm 들을 이용하여 Modular multiplication 을 수행할 수 있음

Algorithm 3 : $\overline{\mathrm{KMULT}}$ → h + f*g

Algorithm 1 : $\mathrm{MULT}_{f(x)}$→ G as $f \cdot g$ in the field $\mathbb{F}_2/m(x)$, input : g

MODSHIFT → k 번 shift 연산 그런데 , x^k 곱해주고 $modular$ 수행

Algorithm 4 : MODMULT

Algorithm 4: MODMULT. Reversible algorithm for multiplication of 2 polynomials in $\mathbb{F}_2[x]/m(x)$ with $m(x)$ an irreducible polynomial.

Fixed input : A constant integer n to indicate field size, $k = \lceil \frac{n}{2} \rceil$. $m(x)$ of degree n as the field polynomial. The LUP-decomposition precomputed for multiplication by $1 + x^k$ modulo $m(x)$.

Quantum input: Two binary polynomials $f(x), g(x)$ of degree up to $n-1$ stored in arrays A and B respectively of size n. An all-zero array C of size n

Result: A and B as input, C as $f \cdot g \mod m$.

1 $C[0..n-1] \leftarrow \text{KMULT}(A[k..n-1], B[k..n-1], C[0..n-1])$
2 $C[0..n-1] \leftarrow \text{MULT}_{1+x^k}(C[0..n-1])$
3 $A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$
4 $B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$
5 $C[0..n-1] \leftarrow \text{KMULT}(A[0..k-1], B[0..k-1], C[0..n-1])$
6 $B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$
7 $A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$
8 **for** $i = 0..k-1$ **do**
9 $\quad C[0..n-1] \leftarrow \text{MODSHIFT}(C[0..n-1])$
10 $C[0..n-1] \leftarrow \text{MULT}^{-1}_{1+x^k}(C[0..n-1])$
11 $C[0..n-1] \leftarrow \text{KMULT}(A[0..k-1], B[0..k-1], C[0..n-1])$
12 $C[0..n-1] \leftarrow \text{MULT}_{1+x^k}(C[0..n-1])$

$$\alpha = f_0 \cdot g_0, \beta = f_1 \cdot g_1$$

Line	C in MODMULT
1	β
2-4	$(1+x^k)\beta \mod m$
5-7	$\gamma + (1+x^k)\beta \mod m$
8,9	$x^k\gamma + x^k(1+x^k)\beta \mod m$
10	$(1+x^k)^{-1}(x^k\gamma + x^k(1+x^k)\beta) \mod m$
11	$\alpha + (1+x^k)^{-1}(x^k\gamma + x^k(1+x^k)\beta) \mod m$
12	$(1+x^k)\alpha + x^k\gamma + x^k(1+x^k)\beta \mod m$

Table 4: Step-by-step calculation of Algorithm 4.

Algorithm 4 : MODMULT

Line	C in MODMULT
1	β
2-4	$(1+x^k)\beta \mod m$
5-7	$\gamma + (1+x^k)\beta \mod m$
8,9	$x^k\gamma + x^k(1+x^k)\beta \mod m$
10	$(1+x^k)^{-1}(x^k\gamma + x^k(1+x^k)\beta) \mod m$
11	$\alpha + (1+x^k)^{-1}(x^k\gamma + x^k(1+x^k)\beta) \mod m$
12	$(1+x^k)\alpha + x^k\gamma + x^k(1+x^k)\beta \mod m$

Table 4: Step-by-step calculation of Algorithm 4.

$k = \lceil \frac{n}{2} \rceil$) we can split each polynomial as follows: $f = f_0 + f_1x^k$, $g = g_0 + g_1x^k$ and $h = h_0 + h_1x^k + h_2x^{2k} + h_3x^{3k}$.

We compute intermediate products $\alpha = f_0 \cdot g_0$, $\beta = f_1 \cdot g_1$ and $\gamma = (f_0 + f_1) \cdot (g_0 + g_1)$. Finally, we add these in the right way for Karatsuba multiplication:

$$h + f \cdot g = h + \alpha + (\gamma + \alpha + \beta)x^k + \beta x^{2k}.$$

For cleanliness, we can split up our α, β, γ in the same way as f and g to get a result with no overlap, which is useful for checking correctness:

$$h+f\cdot g = (h_0+\alpha_0)+(h_1+\alpha_0+\alpha_1+\beta_0+\gamma_0)x^k+(h_2+\alpha_1+\beta_0+\beta_1+\gamma_1)x^{2k}+(h_3+\beta_1)x^{3k}$$

Alternatively, we can rewrite this another way that will prove useful:

$$\boxed{h + f \cdot g = h + (1+x^k)\alpha + x^k\gamma + x^k(1+x^k)\beta.}$$

Algorithm 4 : MODMULT

Algorithm 4: MODMULT. Reversible algorithm for multiplication of 2 polynomials in $\mathbb{F}_2[x]/m(x)$ with $m(x)$ an irreducible polynomial.

Fixed input : A constant integer n to indicate field size, $k = \lceil \frac{n}{2} \rceil$. $m(x)$ of degree n as the field polynomial. The LUP-decomposition precomputed for multiplication by $1 + x^k$ modulo $m(x)$.

Quantum input: Two binary polynomials $f(x), g(x)$ of degree up to $n-1$ stored in arrays A and B respectively of size n. An all-zero array C of size n

Result: A and B as input, C as $f \cdot g \mod m$.

$C[0..n-1] \leftarrow \text{KMULT}(A[k..n-1], B[k..n-1], C[0..n-1])$
$C[0..n-1] \leftarrow \text{MULT}_{1+x^k}(C[0..n-1])$
$A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$
$B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$
$C[0..n-1] \leftarrow \text{KMULT}(A[0..k-1], B[0..k-1], C[0..n-1])$
$B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$
$A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$
for $i = 0..k-1$ **do**
    $C[0..n-1] \leftarrow \text{MODSHIFT}(C[0..n-1])$
$C[0..n-1] \leftarrow \text{MULT}^{-1}_{1+x^k}(C[0..n-1])$
$C[0..n-1] \leftarrow \text{KMULT}(A[0..k-1], B[0..k-1], C[0..n-1])$
$C[0..n-1] \leftarrow \text{MULT}_{1+x^k}(C[0..n-1])$

Result

- 3 calls to Algorithm 3: twice for k-by-k multiplication and once for $(n-k)$-by-$(n-k)$ multiplication.
- 3 calls to Algorithm 1 (once in reverse), each time for multiplication by the same polynomial $1 + x^k$.
- k calls to MODSHIFT.
- 4 times $(n-k)$ CNOT gates, half of which can be performed at the same time.

Performance

Field size 2^n	Toffoli gates			CNOT gates			qubits		
$n =$	Here	[7]	[9]	Here	[7]	[9]	Here	[7]	[9]
4	9	9	16	49	22	3	12	17	12
16	81	81	256	725	376	45	48	113	48
127	2185	2185	16129	21028	13046	126	381	2433	381
256	6561	6561	65536	66107	57008	765	768	7073	768
n	$O(n^{\log_2 3})$	$O(n^{\log_2 3})$	n^2	$O(n^2)$	$O(n^{\log_2 3})$	$O(n)$	$3n$	$O(n^{\log_2 3})$	$3n$

Table 6: Comparison of this work with the works of Kepley and Steinwandt [7] and Maslov et al. [9] in terms of Toffoli and CNOT gates as well as qubit count.

감사합니다