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.

1 Introduction

Multiplication of two polynomials in a finite field is an important step in many algorithms, such as point addition in elliptic curve cryptography. For classical computers a wealth of variations exist, often based around Karatsuba's multiplication method [6].

In the classical setting, temporary results for the steps of Karatsuba calculations have traditionally been stored separately. In 1993 Maeder [8] used around $2n$ additional space for multiplying degree-n polynomials. This was improved by Thomé in 2002 to n temporary space which at the time was believed to be optimal: "it does not seem likely that anything better than this result can be obtained." [13] However, in 2009 Roche did obtain a better result: $O(\log n)$ space Karatsuba multiplication of polynomials without additional time by doing many in-place operations [11]. This was expanded by Cheng [4] to also work for integers. Despite the advantages these variants offer, these methods are still relatively unknown.

This bound of $O(\log n)$ temporary storage is still higher than the bound presented in this paper, which is reduced to 0 by partly overwriting the input polynomial and restoring it before the end. With this advantage we can modify the algorithms presented by Roche [11] for the quantum setting. The algorithms in this paper have an exponential speedup over other quantum algorithms that do not use extra space [10]. Other variants that reach the same speedup as

classical Karatsuba multiplication in the quantum setting so far have have done so at the cost of space [7].

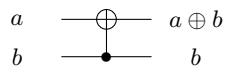
1.1 Overview

We introduce our notation for quantum computing by giving the elementary quantum gates in section 2. Our new multiplication algorithm needs several subroutines, specifically modular shifts and multiplication by a constant polynomial, introduced in section 3. We introduce a Quantum Karatsuba algorithm for multiplication without reduction in section 4 and in binary finite fields in section 5. Both algorithms run without ancillary qubits and have a sub-quadratic Toffoli gate count. We implemented the algorithm in a simulated quantum computer and present the gate counts for specific finite fields in section 6.

2 Quantum background

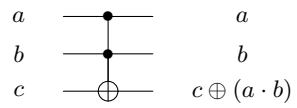
Quantum computing uses reversible gates, which unlike classical gates can be run in reverse and require an equal number of input and output quantum bits (qubits). In this paper we will not make use of the quantum properties of qubits, but the gates we use can be applied to superpositions of qubits in states 1 and 0. For the purpose of multiplication we need two gates to do reversible addition and multiplication:

- The CNOT, or Feynman, gate serves as the equivalent of XOR or $\mathbb{F}_2$-addition. This gate takes 2 qubits as inputs and adds one input to the other qubit and outputs the other qubit as itself: $(a, b) \to (a \oplus b, b)$. It is reversible and its own inverse: applying it twice would result in $(a \oplus b \oplus b, b) = (a, b)$. In Circuit 1 an example has been drawn. In algorithms we write this as $a \leftarrow \text{CNOT}(a, b)$.
- The Toffoli (TOF) gate serves as the equivalent of AND or $\mathbb{F}_2$-multiplication in our case. This gate takes 3 qubits as inputs and adds the result of mulitplication of the frist two qubits to the third qubit and outputs the other qubits as themselves: $(a, b, c) \to (a, b, c \oplus (a \cdot b))$. It is also its own inverse. In circuit 2 an example has been drawn. In algorithms we write this as $c \leftarrow \text{TOF}(a, b, c)$

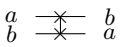


Circuit 1: The CNOT gate

In addition to these operations, we will also need to swap some qubits. Unlike the previous gates we do not build these in physical circuits. Rather, we change the index on some qubits: if we were to swap qubits 1 and 2 we would simply refer to qubit 1 as "2" and qubit 2 as "1" from that point on without counting



Circuit 2: The TOF gate



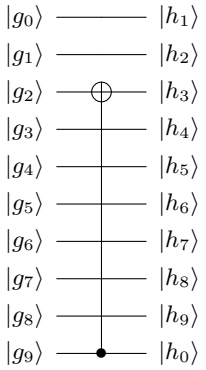
Circuit 3: The swap

any quantum gates. In Circuit 3 an example has been drawn.
These 3 actions are the only essential ones we use in this paper. Although none of these are explicit quantum actions, the quantum dimension comes from optimizing for low Toffoli gate count. Currently no large quantum computer exists but current estimates put the cost of one Toffoli gate at many times that of a CNOT gate.

3 Basic Arithmetic

In this section we discuss reversible in-place algorithms for the basic arithmetic of binary polynomials.

3.1 Addition and binary shift



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

The first operation we consider, addition, can easily be implemented for binary polynomials. Individual additions can be done with a CNOT gate, the addition of two polynomials of degree at most n takes $n + 1$ CNOT gates with depth 1. This operation uses ancillary qubits and the result of the addition replaces

either of the inputs. Since addition is component-wise, addition for polynomials over $\mathbb{F}_2$ is the same as addition for elements of the field $\mathbb{F}_{2^n}$.

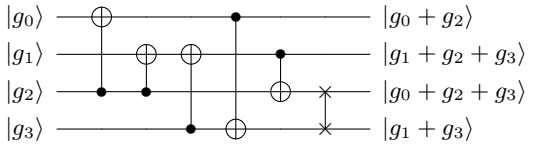
Binary shifts are straightforward: they correspond to multiplying or dividing by x. This requires no quantum computation by doing a series of swaps.

Finally, if we have a fixed n, a polynomial $g(x)$ of degree at most $n-1$ and want to do a multiplication by x followed by a modular reduction by a fixed weight-ω and degree-n polynomial $m(x)$ that has coefficient 1 for x^0, we can do this in 2 steps. We represent $m(x)$ as M where M is an ordered list of length ω that contains the degrees of the nonzero terms in descending order, for example if $m(x) = 1 + x^3 + x^{10}$ we get $M = [10, 3, 0]$. Let $g(x) = \sum_{i=0}^{n-1} g_i x^i$:

- Step 1: For every qubit g_i change its index so that it represents the coefficient of $x^{i+1 \bmod n}$. Let h_i be the coefficients of the relabeled polynomial, i.e. $h_{i+1 \bmod n} = g_i$.
- Step 2: Apply CNOT controlled by the x^0 term h_0 (g_{n-1} before Step 1) to h_j, with $j = M_1, \ldots, M_{\omega-2}$. In the example of $1 + x^3 + x^{10}$ we would apply 1 CNOT to h_3 controlled by h_0.

See Circuit 4 for an example. After a multiplication by x the coefficient of x^0 is always 0. Since $m(x)$ always has coefficient 1 for x^0, after a reduction by $m(x)$ that qubit will be 1 and if no reduction takes place that qubit is 0, which means our modular shift algorithm is always reversible. This results in a total of $\omega - 2$ CNOT gates for a modular reduction, with depth $\omega - 2$ and we do not use ancillary qubits. Since we use reversible gates, running this circuit in reverse corresponds to dividing by x modulo $m(x)$.

3.2 Multiplication by a constant polynomial



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

Multiplication by a constant non-zero polynomial in a fixed binary field is $\mathbb{F}_2$-linear: as the field polynomial is irreducible, every input corresponds to exactly one output. We can see that any such multiplication can be represented as a matrix, which we can turn into a circuit using an LUP-decomposition, an algorithm also used by Amento, Rötteler and Steinwandt [1]. For example, multiplication by $1 + x^2$ modulo $1 + x + x^4$ can be represented by a matrix Γ. Using the decomposition $\Gamma = P^{-1}LU$ we get an upper and lower triangular matrix which we can translate into a circuit. Any 1 not on the diagonal in U and L is a CNOT

controlled by the column number on the row number. In cases of conflict[1], for U CNOT gates should be performed top row first, second row second and so on and for L CNOT gates from the bottom row up. P represents a series of swaps, and can be represented either as a permutation matrix or an ordered list with all elements from 0 to $n-1$.

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

Circuit 5 shows how we translate Γ. According to [1] this costs up to n^2+n CNOT gates with depth up to $2n$. We can improve this count by noting L and U are each size n by n and can have up to $(n^2-n)/2$ non-diagonal non-zero entries, giving us up to n^2-n CNOT gates. Note that the LUP-decomposition is precomputed and for any fixed polynomial and field we can give an exact CNOT gate count and depth.

Since this algorithm is introduced in [1] without correctness proof and we will use it later for a bigger algorithm, we will write an explicit implementation and go over the correctness of this algorithm. Note that since we are working with reversible algorithms, multiplying by constant $f(x)$ is the same as doing the reverse of multiplying by constant $f(x)^{-1}$.

Theorem 1. *Algorithm 1 correctly describes multiplication by a non-zero constant polynomial in a fixed binary field.*

Proof. Since multiplication by a non-zero constant in a finite field is a linear map, an invertible matrix Γ to represent this linear map must exist. Since Γ is invertible, its decomposition L, U, P^{-1} must also consist of linear maps. Since we are working in a binary field and U is an invertible upper-triangular matrix, the diagonal of U is all-one. If we look at lines 1 through 4 of the algorithm, we can see it corresponds to applying linear map U to g, as it results in $g_i = \sum_{j=0}^{n-1} u_{i,j} g_j$ for $i = 0, .., n-1$. Analogously the same is true for L in lines 5 through 8. We can also see that if P^{-1} is a row-permutation of the identity matrix, lines 9 through 13 will apply it correctly. Since $P^{-1}LU = \Gamma$ we have correctly applied the linear map Γ. □

Note that the algorithm is not optimized for depth, for example in circuit 5 the first and second CNOT could be swapped so the depth would be 3 rather than 4.

[1] Conflicts exist if according to the triangular matrix a CNOT would both have to be applied on and controlled by a qubit. By doing the controlled operation first and applying the operation on it afterwards, we ensure that the matrix multiplication is correctly translated.

Algorithm 1: $\text{MULT}_{f(x)}$, 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 $\mathbb{F}_2/m(x)$.

```
1  for i = 0..n − 1                                          // U · G
2   do
3       for j = i + 1..n − 1 do
4           if U[i, j] = 1 then
5               G[i] ← CNOT(G[i], G[j])
6  for i = n − 1..0                                          // L · UG
7   do
8       for j = i − 1..0 do
9           if L[i, j] = 1 then
10              G[i] ← CNOT(G[i], G[j])
11 for i = 0..n                                              // P^{-1} · LUG
12  do
13      for j = i + 1..n − 1 do
14          if P^{-1}[i, j] = 1 then
15              SWAP(G[i], G[j])
16              SWAP column i and j of P^{-1}
```

Choice of field polynomials

When doing operations in a finite binary field we can choose what representation we use, as long as the polynomial $m(x)$ is irreducible. Our goal is to make the matrices L and U as sparse as possible. For this purpose we also want our Γ to be as sparse as possible, which can be achieved in two steps: pick irreducible polynomials with as few non-zero coefficients as possible, i.e. trinomials when available and pentanomials otherwise, and pick irreducible polynomials where the second highest non-constant term has the lowest possible degree. For example, the pentanomial $1 + x^3 + x^4 + x^{19} + x^{20}$ would require 108 CNOT gates, the pentanomial $1 + x^3 + x^5 + x^9 + x^{20}$ would require 55 CNOT gates, while the trinomial $1 + x^3 + x^{20}$ would require only 27. All 3 polynomials are irreducible. In Table 1 we can see some examples of gate counts for various choices of n. The depth count is an upper bound without accounting for swapping gates.

Degree	Irreducible polynomial	Source	CNOT gates	Depth upper bound
4	$[4, 1, 0]$	[2]	5	4
8	$[10, 4, 3, 1, 0]$	[2]	20	14
16	$[16, 5, 3, 1, 0]$	[2]	47	30
32	$[32, 7, 3, 2, 0]$	[2]	133	93
64	$[64, 4, 3, 1, 0]$	[2]	264	182
127	$[127, 1, 0]$	[2]	396	293
128	$[128, 7, 2, 1, 0]$	[2]	626	443
163	$[163, 7, 6, 3, 0]$	[5]	740	975
163	$[163, 89, 74, 15, 0]$	[3]	1885	1646
233	$[233, 74, 0]$	[5]	3319	2976
256	$[256, 10, 5, 2, 0]$	[2]	1401	1030
283	$[283, 12, 7, 5, 0]$	[5]	2117	1700
283	$[283, 160, 123, 37, 0]$	[3]	6785	6368
571	$[571, 10, 5, 2, 0]$	[5]	4027	3177
571	$[571, 353, 218, 135, 0]$	[3]	33182	32331
1024	$[1024, 19, 6, 1, 0]$	[12]	8147	6624

Table 1: Comparison of the CNOT gates required for various instances of Algorithm 1. Source is the source of the polynomial.

4 Quantum Multiplication for binary polynomials

This section details schoolbook multiplication and we present our new Karatsuba algorithm.

4.1 Quantum Schoolbook Multiplication

The simplest way to multiply is schoolbook multiplication. For two polynomials of degree at most $n-1$ that takes n^2 Toffoli gates, the number of pairs of qubits from the first and second polynomial. While the computation does not use ancillary qubits, the result needs to be stored separately from input in $2n-1$ qubits; unlike the previous circuits we cannot replace either of the inputs with the result since the Toffoli gate requires a separate output. If we want to apply modular reduction steps by a weight-k and degree-n odd polynomial, this adds $(n-1)\cdot(k-2)$ CNOT gates and uses no ancillary qubits (by using the modular shift algorithm after every n multiplications). The result is stored in n qubits.

4.2 Classic Karatsuba multiplication in binary polynomial rings

Rather than using schoolbook multiplication, methods like Karatsuba multiplication [6] can speed up multiplication of large numbers. We can look at in-place multiplication in the classical case for ideas [11]. As input we take two polynomials of size up to n, $f(x)$ and $g(x)$ as well as a polynomial of size $2n$: $h(x)$. As output we desire $h + f \cdot g$. For some k such that $\frac{n}{2} \leq k < n$ (we will always use

$k = \lceil \frac{n}{2} \rceil$) we can split each polynomial as follows: $f = f_0 + f_1 x^k$, $g = g_0 + g_1 x^k$ and $h = h_0 + h_1 x^k + h_2 x^{2k} + h_3 x^{3k}$.

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:

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

4.3 Reversible Karatsuba multiplication in binary polynomial rings

Based on these equations we can split our multiplication algorithm into 2 parts: given $f(x), g(x), h(x)$ calculate $h + f \cdot g$ and given $k, f(x), g(x), h(x)$ with $k > \max(\deg(f), \deg(g))$ calculate $h + (1 + x^k)f \cdot g$. We will look at our algorithms for the 2 parts, which can then be used recursively to provide a significant improvement to the schoolbook algorithm in terms of Toffoli gate count.

Algorithm 2: 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**
- $C[k..k + \ell - 1] \leftarrow \text{CNOT}(C[k..k + \ell - 1], C[2k..2k + \ell - 1])$
- $C[0..k - 1] \leftarrow \text{CNOT}(C[0..k - 1], C[k..2k - 1])$
- $C[k..2k + \ell - 1] \leftarrow \text{KMULT}(A[0..n - 1], B[0..n - 1], C[k..2k + \ell - 1])$
- $C[0..k - 1] \leftarrow \text{CNOT}(C[0..k - 1], C[k..2k - 1])$
- $C[k..k + \ell - 1] \leftarrow \text{CNOT}(C[k..k + \ell - 1], C[2k..2k + \ell - 1])$

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

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.

Lemma 1. *Given polynomials f, g of degree up to $n-1$ with $n>1$, polynomial h of degree up to $k+2n-2$ with some $k \geq n$ and assuming Algorithm 3 correctly calculates $h+fg$ with degrees of f, g and h bounded as above, Algorithm 2 correctly calculates $h+(1+x^k)fg$ in $\mathbb{F}_2[x]$ without altering the values of f and g.*

Proof. Let $\ell = \max(0, 2n-1-k)$. Table 2 gives the result of each step on array C, split into 3 parts of size k, k and $\ell-1$ respectively: $h = h_0+h_1x^k+h_2x^{2k}$. The final result corresponds to $h_0+(fg)_0+(h_1+(fg)_0+(fg)_1)x^k+(h_2+(fg)_1)x^{2k} = h_0+h_1x^k+h_2x^{2k}+fg+fgx^k = h+(1+x^k)fg$, where $(fg)_0$ is the first k terms of $f \cdot g$ and $(fg)_1$ is the last up to ℓ terms.

f and g do not have their values altered because arrays A and B remain unchanged. □

Algorithm 2 computes $h+(1+x^k)fg$ with at most $2k+2\ell \geq 2k+2(2n-1-k) = 4n-2$ CNOT gates, at a depth of 4 per layer and 1 call to Algorithm 3 for an n-by-n multiplication. For $n=1$ both the depth and number of gates is 2 CNOT and 1 TOF gates.

Lemma 2. *Let $k = \lceil \frac{n}{2} \rceil$. Given polynomials f, g of degree up to $n-1$ with $n>1$ and h of degree up to $2n-2$. Assuming Algorithm 2 correctly calculates $h'+(1+x^k)f'g'$ for f', g' up to degree $k-1$ and h' up to degree $3k-2$, and Algorithm 3 correctly calculates $h''+f''g''$ with f'', g'' of degree $k-1$ and h'' of degree $2k-2$ without altering the values of f'' and g''. Then Algorithm 3 correctly calculates $h+fg$ in $\mathbb{F}_2[x]$. The values of f and g are the same after the algorithm as they were before.*

Proof. Table 3 gives the result of each line on array C, split into 4 parts of size k, k, k and $2n-1-3k$ respectively: $h = h_0+h_1x^k+h_2x^{2k}+h_3x^{3k}$. As can be seen in the table, the final result corresponds to $(h_0+\alpha_0)+(h_1+\alpha_0+\alpha_1+\beta_0+\gamma_0)x^k+(h_2+\alpha_1+\beta_0+\beta_1+\gamma_1)x^{2k}+(h_3+\beta_1)x^{3k} = h+f\cdot g$ as discussed in Section 4.2. Lines 7 and 8 are the inverses of lines 4 and 5 so return A and B to their original states. □

Algorithm 3 computes $h+fg$ with $4(n-k)$ CNOT gates, at a depth of 4, 1 call to itself for a k-by-k multiplication, 1 call to Algorithm 2 for a k-by-k multiplication

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 $\quad C[0..3k-2] \leftarrow \text{MULT1x}_k(A[0..k-1], B[0..k-1], C[0..3k-2])$
3 $\quad C[k..2n-2] \leftarrow \text{MULT1x}_k(A[k..n-1], B[k..n-1], C[k..2n-2])$
4 $\quad A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$
5 $\quad B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$
6 $\quad C[k..3k-2] \leftarrow \text{KMULT}(A[0..k-1], B[0..k-1], C[k..3k-2])$
7 $\quad B[0..n-k-1] \leftarrow \text{CNOT}(B[0..n-k-1], B[k..n-1])$
8 $\quad A[0..n-k-1] \leftarrow \text{CNOT}(A[0..n-k-1], A[k..n-1])$
9 **else**
10 $\quad C[0] \leftarrow \text{TOF}(A[0], B[0], C[0])$

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.

and 1 call to Algorithm 2 for an $(n-k)$-by-$(n-k)$ multiplication. For $n = 1$ we just have a single TOF gate.

Theorem 2. *Given polynomials f, g of degree up to $n-1$ and h of degree up to $2n-2$, Algorithm 3 correctly calculates $h + fg$. The values of f and g are the same after the algorithm as they were before.*

Proof. We use proof by induction. For $n = 1$ line 10 of Algorithm 3 correctly calculates $h + fg$ without altering f or g.

For $n = 2$ two calls are made to Algorithm 2 and one call to Algorithm 3 with $n' = 1$ and $k' = 1$. Lines 7-9 of Algorithm 2 correctly calculate $h' + (1 + x^k)f'g'$.

For $n > 2$ we use lemmas 1 and 2 as our inductive steps. Every time Algorithm 3 is called recursively to calculate $h' + f'g'$ with f', g' of degree $n'-1$, it is with either $n' = \lceil \frac{n}{2} \rceil$ or $n' = n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$.

The series $\lceil \frac{n}{2} \rceil, \lceil \frac{\lceil \frac{n}{2} \rceil}{2} \rceil, \lceil \frac{\lceil \frac{\lceil \frac{n}{2} \rceil}{2} \rceil}{2} \rceil, \ldots$ reaches 1 in $O(\log n)$ steps and $\lfloor \frac{n}{2} \rfloor \leq \lceil \frac{n}{2} \rceil$. From this we can see that we reach $n' = 1$ or 2 in finite steps. By induction

Algorithm 3 correctly calculates $h + fg$ and returns f and g to their original values. □

5 Reversible Karatsuba multiplication in binary finite fields

With this basis, we can move on to the modular multiplication. We will need Algorithm 1, which we will also run in reverse for multiplication by an inverse, and the binary shifts from Section 3.1, which we will refer to as MODSHIFT, as well as the previous Karatsuba algorithms. Unlike before, we will assume we start with an all-zero input as it saves a significant number of additions, although it would cost no Toffoli gates: lines 2 and 8-9 would have to be run in reverse on that input. We can see in Algorithm 4 the number of operations we use:

- 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.

Note that Algorithm 3 can multiply two polynomials f and g of degree at most $\lceil \frac{n}{2} \rceil - 1$ while needing n space for the output polynomial h, which has degree $n-1$ at most in the case that n is odd. We make recursive calls to Algorithm 3 rather than Algorithm 4 because it uses significantly fewer CNOT operations and fits in the required space.

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.

Theorem 3. *Algorithm 4 correctly calculates fg in a field $\mathbb{F}_2[x]/m(x)$ and the values of f and g are the same after the algorithm as they were before.*

Proof. Table 4 gives the result of each line on array C. As can be seen in the table, the final result corresponds to $(1+x^k)\alpha + x^k\gamma + x^k(1+x^k)\beta \mod m$. Lines 6 and 7 are the inverses of lines 3 and 4 so return A and B to their original states. □

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

6 Results

Algorithm 4 uses the same number of Toffoli gates as regular Karatsuba multiplication: 3 half-sized multiplications. This means the asymptotic number of Toffoli gates is the same as for regular Karatsuba: $O(n^{\log(3)}) \approx O(n^{1.58})$. This is a significant improvement over the n^2 Toffoli gates required for schoolbook multiplication. The number of CNOT gates is less clear as the number of CNOT gates required for the multiplications with constant polynomials is strongly dependent on our choice of field polynomial. It is not within the scope of this paper to find a stronger bound than $O(n^2)$ for the number of CNOT gates, which is currently used for the constant multiplication. In a strict comparison of these CNOT gates, this is worse than the $O(n)$ CNOT gates used by modular schoolbook multiplication, even if we can find a better estimate, but our primary goal is minimizing the number of Toffoli gates without introducing ancillary qubits. In our implementation, even the sum of CNOT and Toffoli gates ends up lower after some degree than the number of Toffoli gates for schoolbook multiplication.

We implemented Algorithm 4 in Java to simulate the execution. Code can be found in [14]. We used the program to automatically count the number of gates and give an estimate of the depth, see Table 5 for the results. Depth count is done by maintaining a set of gates and checking every gate: if they overlap with the previous gate(s) the depth is increased by 1 and if they are not overlapping the gate is added to the set of gates to check against. The set of gates is cleared

Degree	schoolbook TOF gates	Algorithm 4 TOF gates	CNOT gates	Depth upper bound
2	4	3	11	10
4	16	9	49	36
8	64	27	220	139
16	256	81	725	396
32	1,024	243	2,371	1204
64	4,096	729	7,160	3,312
127	16,129	2,185	21,028	9,063
128	16,384	2,187	21,898	9,586
163	26,569	4,387	38,143	18,647
233	54,289	6,323	66,974	32,505
256	65,536	6,561	66,107	27,756
283	80,089	10,273	91,737	43,249
571	326,041	31,171	274,967	124,999
1024	1,048,576	59,049	600,089	240,678

Table 5: CNOT and TOF gate count and depth upper bounds for various instances of Algorithm 4 as well as TOF gate count for schoolbook multiplication. Field polynomials used are the same as in Table 1, with the irreducible polynomial chosen that has the lowest CNOT count.

and replaced with the last gate whenever the depth is increased. The author is aware of methods to improve the depth but leaves this to future work.

When doing classical Karatsuba multiplication, the recursive Karatsuba multiplication is often substituted for schoolbook multiplication starting at a cutoff. For example, if multiplication is at most 7 times as expensive as addition, multiplication of two polynomials of degree at most 2 might be replaced by schoolbook multiplication to get 4 TOF gates instead of 3 TOF and 8 CNOT gates. However, the author is unaware of any realistic estimates of cost difference between CNOT and Toffoli gates where the difference is this small.

6.1 Comparison to other instances of binary finite field multiplication

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.

We compare our algorithm to two previous instances of multiplication: a variant by Kepley and Steinwandt [7] that optimizes TOF gate count and a variant by Maslov, Mathew, Cheung and Pradhan [9] that does not use Karatsuba. Other variants exist, such as a Karatsuba variant by Parent, Roetteler and Mosca [10], that are worse in terms of space or Toffoli gate count. Since Kepley and Steinwandt use Clifford and T-gates rather than CNOT and Toffoli, we translate 7 of their T-gates and 8 Clifford gates to 1 Toffoli gate, and translate any remaining Clifford gates to CNOT. The resulting comparison is in Table 6. We can see that although Algorithm 4 does not compare favorably in every regard, both the number of Toffoli gates and the number of qubits are best compared to the alternatives.

7 Conclusion

Algorithm 4 provides a multiplication algorithm for binary polynomials in finite fields without using ancillary qubits and which has sub-quadratic Toffoli gate count. The CNOT gate count is high and the depth is not optimized, which is left open for future work: multiplication by a constant polynomial in $\mathbb{F}_{2^n}$ can likely be done in approximately linear time, which would bring down the theoretical CNOT gate count to the same order as classical Karatsuba. The saving in Toffoli gate count is the same as for Karatsuba on classical computers: for cryptographic field sizes the savings in Toffoli gates ranges from 80 to over 90 percent. This provides a basis for future work on elliptic curve problems on quantum computers as well as potential other work.

Acknowledgements The author thanks Tanja Lange for her insights into quantum algorithms and classical finite field operations, Tanja Lange and Gustavo Banegas for their advice and supervision both on this paper and the master thesis this paper originates from, and to Daniel J. Bernstein for his insights into both quantum computing and classical multiplication algorithms.

References

1. B. Amento, M. Rötteler, and R. Steinwandt, *Efficient quantum circuits for binary elliptic curve arithmetic: reducing T-gate complexity*, Quantum Information & Computation, 13 (2013), pp. 631–644.
2. R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren, eds., *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman and Hall/CRC, 2005.
3. G. Banegas, R. Custódio, and D. Panario, *A new class of irreducible pentanomials for polynomial-based multipliers in binary fields*, Journal of Cryptographic Engineering, (2018), pp. 1–15.
4. Y. Cheng, *Space-Efficient Karatsuba Multiplication for Multi-Precision Integers*, CoRR, abs/1605.06760 (2016).

5. FIPS, *PUB 186-4: Federal information processing standards publication. digital signature standard (DSS)*, Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD, (2013), pp. 20899–8900.
6. A. A. Karatsuba and Y. P. Ofman, *Multiplication of many-digital numbers by automatic computers*, in Doklady Akademii Nauk, vol. 145, Russian Academy of Sciences, 1962, pp. 293–294.
7. S. Kepley and R. Steinwandt, *Quantum circuits for $\mathbb{F}_{2^n}$-multiplication with subquadratic gate count*, Quantum Information Processing, 14 (2015), pp. 2373–2386.
8. R. Maeder, *Storage Allocation for the Karatsuba Integer Multipliation Algorithm*, in Design and Implementation of Symbolic Computation Systems, International Symposium, DISCO '93, Gmunden, Austria, September 15-17, 1993, Proceedings, 1993, pp. 59–65.
9. D. Maslov, J. Mathew, D. Cheung, and D. K. Pradhan, *An $O(m^2)$-depth quantum algorithm for the elliptic curve discrete logarithm problem over $GF(2^m)^a$*, Quantum Information & Computation, 9 (2009), pp. 610–621.
10. A. Parent, M. Roetteler, and M. Mosca, *Improved reversible and quantum circuits for Karatsuba-based integer multiplication*, in 12th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2017, June 14-16, 2017, Paris, France, 2017, pp. 7:1–7:15.
11. D. S. Roche, *Space- and Time-Efficient Polynomial Multiplication*, in Proceedings of the 2009 international symposium on Symbolic and algebraic computation, ACM, 2009, pp. 295–302.
12. G. Seroussi, *Table of low-weight binary irreducible polynomials*, Hewlett-Packard Laboratories, 1998.
13. E. Thomé, *Karatsuba multiplication with temporary space of size$\leq$ n*, Online, September, (2002). `https://members.loria.fr/EThome/files/kara.pdf`.
14. I. van Hoof, *QMKMBP: Quantum modulo Karatsuba multiplier for binary polynomials*. Github, 2019. `https://github.com/ikbenbeter/QMKMBP`.