Binary Field Montgomery Multiplication on Quantum Computers

Kyoungbae Jang[1], Gyeong Ju Song[1], Hyunji Kim[1], Hyeokdong Kwon[1], Wai-Kong Lee[2], Zhi Hu[3], and Hwajeong Seo[1][0000−0003−0069−9061]

[1]IT Department, Hansung University, Seoul (02876), South Korea,
{starj1023, thdrudwn98, khj1594012, korlethean, hwajeong84}@gmail.com
[2]Department of Computer Engineering,
Gachon University, Seongnam, Incheon (13120), Korea,
waikonglee@gachon.ac.kr [3]Central South University, China,
huzhi_math@csu.edu.cn

Abstract. Optimizing arithmetic operations into quantum circuits to utilize quantum algorithms, such as the Shor algorithm and Grover search algorithm for cryptanalysis, is an active research field in cryptography implementation. In particular, reducing quantum resources is important for efficient implementation. In this paper, binary field ($GF(2^n)$) Montgomery multiplication in quantum circuits is presented. We utilize the bit-level Montgomery algorithm to efficiently compute the Montgomery product $C = A \cdot B \cdot r^{-1}$ in the binary field $GF(2^n)$. Additionally, we also present an efficient Montgomery multiplication quantum circuit in the case where the modulus of $GF(2^n)$ is specified.

Keywords: Quantum Computers · Montgomery Multiplication · Binary Field.

1 Introduction

International companies, such as Google and IBM, are advancing the development of large-scale quantum computers. Quantum computers have more computational power than classical computers in certain areas, such as deep learning, chemistry, and cryptography. If a large-scale quantum computer capable of operating quantum algorithms is developed, the safety of currently widely used cryptographic algorithms may be lowered or broken. It has been proven that Shor's algorithm can break the safety of RSA and Elliptic Curve Cryptography (ECC). How long RSA and ECC can be used depends on the development of quantum computers and optimization of Shor's algorithm [1].

In [2], authors estimated that Shor's algorithm can be applied using $2n + 2$ qubits for RSA of n-bit key. Gidney estimated the number of improved $2n + 1$ qubits [3]. Shor's algorithm can also be applied to discrete logarithms in elliptic curves (i.e. ECC). In [4], the authors presented that ECC is more vulnerable to attack by quantum computers than RSA by estimating quantum resources required to solve the elliptic curve discrete logarithms. In [5], it was shown that

Shor's algorithm can be applied to ECC with fewer resources than the result of [4]. Both works are the results of studies targeting prime curves [4] and [5]. In 2020 [6], quantum resources were estimated on binary curves, and fewer quantum resources were required than prime curves.

Arithmetic operations performed in quantum algorithms must be implemented as quantum circuits, and it is important to optimize required quantum resources. Due to this motivation, implementing arithmetic operations in the prime field $GF(p)$ or the binary field $GF(2^n)$ as a quantum circuit is an active research field. Among them, many studies have been presented to optimize modular multiplication that requires many qubits and quantum gates [7–12]. In [7, 10, 11], the Karatsuba algorithm, which is widely used in practice, was used. The Karatsuba algorithm divides an n-bit multiplication into three $\frac{n}{2}$-bit multiplications. Although, Karatsuba multiplication needs several extra additions, the method significantly reduces the complexity of multiplication. By applying the Karatsuba algorithm to quantum computing, the gate complexity in quantum circuits is also optimized. In [12], the prime field multiplication applying the Montgomery reduction algorithm is presented.

In this paper, we focus on binary field multiplication using the Montgomery reduction algorithm. By utilizing the bit-level Montgomery algorithm [13], it is optimized for quantum computing in which qubit unit operations are performed. The proposed binary field Montgomery multiplication quantum circuit compute the Montgomery product for any modulus, and we also propose a more efficient quantum circuit when the modulus is specified. The proposed method was implemented by utilizing ProjectQ [14], a quantum programming tool provided by IBM. ProjectQ can implement quantum circuits using quantum gates and qubits, and can analyze the quantum resources required for the circuit.

2 Related Work

2.1 Binary Field Multiplication

Multiplication in $GF(2^n)$ performs n-bit polynomial multiplication and reduction by modulus N. The multiplication of $A(x)$ and $B(x)$ in $GF(2^n)$ is as follows.

$$C = A \cdot B \bmod N \tag{1}$$

For the product of A and B, a reduction using modulus N is performed over n bits in length. As a result, C, the product of A and B, is an element of $GF(2^n)$.

2.2 Montgomery Multiplication

In 1987, Montgomery proposed a new type of modular multiplication [15]. In the Montgomery algorithm, operands A and B are converted to the Montgomery domain as follows. The choice of the simple exponentiation $r = x^n$ simplifies Montgomery multiplication.

$$\begin{aligned} A' &= A \cdot r \bmod N \\ B' &= B \cdot r \bmod N \end{aligned} \tag{2}$$

General multiplication in $GF(2^n)$ computes $A \cdot B \bmod N$, but Montgomery multiplication computes $A \cdot B \cdot r^{-1} \bmod N$ which is the form multiplied by the inverse of r.

$$(A \cdot B)' = (A \cdot r) \cdot (B \cdot r) \cdot r^{-1} \bmod N = A' \cdot B' \cdot r^{-1} \bmod N \tag{3}$$

The inverse operation in $GF(2^n)$ is complicated, but $N' = -N^{-1} \bmod r$ can be pre-computed. The process for Montgomery multiplication is shown in Algorithm 1.

Algorithm 1 Montgomery multiplication.

```
Input: Modulus $N$, operands $A$ and $B$, exponentiation $r = x^n$, pre-computed $N' = -N^{-1} \bmod r$.
Output: $C = A \cdot B \cdot r^{-1} \bmod N$.
1: $T = A \cdot B$
2: $Q = T \cdot N' \bmod r$
3: $C = (T + Q \cdot N)/r$
4: if $C \geq N$ then
5:     $C = C - N$
6: end if
7: return $C$
```

2.3 Quantum Gates

Quantum gates used in quantum computers are reversible gates that can return to their original state during computation. Quantum computing using reversible quantum gates is different from classical computing, but there are quantum gates that can replace logical operations performed in classical computers. There are X, CNOT, Toffoli, and Swap gates, which are shown in Figure 1.

- **X gate**: This quantum gate can replace the NOT operation, inverting the state of the input qubit. X $(x) = \sim x$.
- **CNOT gate**: This quantum gate can replace the XOR operation, inverting the state of y only if x is 1. CNOT $(x, y) = (x, x \oplus y)$.
- **Toffoli gate**: This quantum gate can replace the AND operation, inverting the state of z only if x and y is 1. Toffoli $(x, y, z) = (x, y, z \oplus x \cdot y)$.
- **Swap gate**: This quantum gate swaps the states of x and y. Swap $(x, y) = (y, x)$.

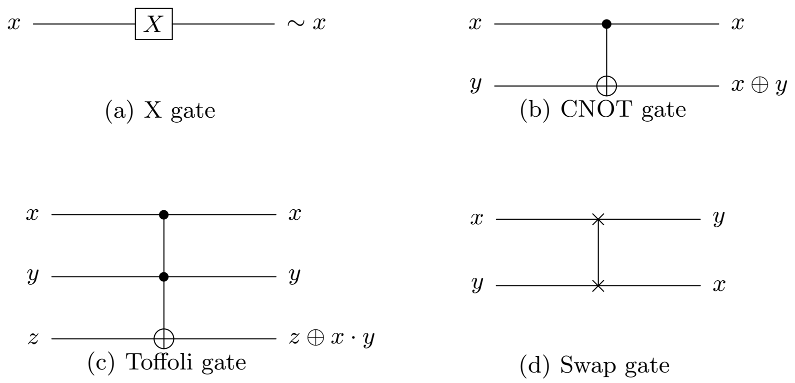

Fig. 1: Quantum gates.

3 Proposed Method

3.1 Qubit-level Montogomery Multiplication

We propose a qubit-level Montgomery multiplication quantum circuit by implementing the bit-level Montgomery multiplication as a quantum circuit. Since the operations between qubits are performed in quantum circuits, we utilized a bit-level Montgomery multiplication [13] as a quantum circuit. Finally, the proposed Montgomery multiplication in quantum circuits computes $A \cdot B \cdot r^{-1}$ mod N in $GF(2^n)$ instead of generic multiplication ($A \cdot B$ mod N). The detailed process for qubit-level Montgomery multiplication is shown in Algorithm 2.

Algorithm 2 Qubit-level Montgomery multiplication quantum circuit.

```
Input: $(n-1)$-qubit of modulus $N$, $n$-qubit operands $A$ and $B$, $n$-qubit result $C$.
Output: $C = A \cdot B \cdot r^{-1}$ mod $N$.
1: for $i = 0$ to $n-1$ do
2:     $C$ = OperandMul $(a_i, B, C)$
3:     $C$ = ModulusMul $(c_0, N, C)$
4:     RotateRight $(C, 1)$
5: end for
6: return $C$
```

In qubit-level Montgomery multiplication, the multiplication of operands A and B is performed by scanning each qubit. Therefore, n iterative multiplications are performed in $GF(2^n)$, which are the OperandMul in Algorithm 2. It is a schoolbook method, but only one qubit of operand A is checked (i.e. a_i), and the product with B is stored in C using the Toffoli gate. The detailed process for OperandMul is shown in Algorithm 3.

Algorithm 3 OperandMul.

```
Input: Single qubit operand $a_i$ and $n$-qubit operand $B$, $n$-qubit result $C$.
Output: $C = a_i \cdot B$.
1: for $j = 0$ to $n - 1$ do
2:     $c_j$ = Toffoli $(a_i, b_j, c_j)$
3: end for
4: return $C$
```

In Montgomery multiplication of Algorithm 1, $N' = -N^{-1} \bmod r$ is computed, which is a part that can be pre-computed. Although the pre-computation is possible, the cost of computing the inverse in $GF(2^n)$ is high. However, we do not compute N' in qubit-level Montgomery multiplication by selecting the bit-level Montgomery multiplication that checks each bit [13]. By scanning each single qubit a_i of A, step 2 of Algorithm 1 can be omitted. Therefore, after OperandMul, ModulusMul of Algorithm 4 is performed.

Algorithm 4 ModulusMul.

```
Input: $(n - 1)$-qubit of modulus $N$ and $n$-qubit result $C$
Output: $C = C + c_0 \cdot N$
1: for $i = 0$ to $n - 2$ do
2:     $c_{i+1}$ = Toffoli $(c_0, n_{i+1}, c_{i+1})$ (need revision)
3: end for
4: return $C$
```

The modulus N of $GF(2^n)$ is $n_n x^n + n_{n-1} x^{n-1} + ... + n_1 x + n_0$. In Algorithm 4, only $(n-1)$-qubit modulus is input to compute the product of modulus and c_0 (i.e. $c_o \cdot N$). This is an advantage considering that n_n is always 1 in the modulus and the division of r in step 3 of Algorithm 1. In n-bit modulus N, n_0 is always 1, and due to r division, c_0 updated by the addition of the product of n_0 and c_0 is discarded due to right-shift operation. For this reason, we ignore the constant part and reconstruct the $(n-1)$-qubit modulus $n_{n-1}x^{n-1} + n_{n-2}x^{n-2} + ... + n_2 x^2 + n_1 x$. In Algorithm 4, ModulusMul, the product of c_0 and n_{i+1} is stored in c_{i+1} by Toffoli gates. Additionally, for the highest coefficient $n_n x^n$, it can be effectively replaced by step 4 (i.e. RotateRight) of Algorithm 2. The detailed process for this (ModulusMul + RotateRight) is shown in Figure 2.

The upper lines of ModulusMul in Figure 2 represent the multiplication part of N and c_0. In Figure 2, we avoid the product of n_n, n_0 by considering the division of r performed next. As a result, c'_0, the product of n_0 and c_0, and c'_n the product of n_n and c_0, are not computed. We compute the products from n_1 to n_{n-1} and add them from c_1 to c_{n-1}. c_0 remains at position c'_0. After ModulusMul is finished, perform RotateRight. Due to the simple exponentiation of $r = x^n$ and the bit-level Montgomery multiplication, division x equals 1 right shift.

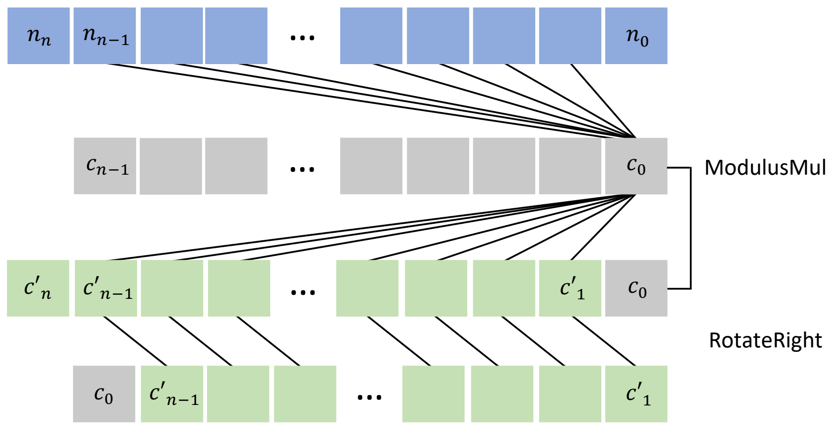

Fig. 2: ModulusMul + RotateRight.

We perform 1 right rotation instead of 1 right shift on the n-qubit resulting from ModulusMul. c'_1 to c'_{n-1} are right shifted by one qubit, and c_0 is positioned at the highest qubit due to rotation operation. In the highest qubit, right-shifted c'_n must be located, but c_0 is located. This is possible because n_n is always 1, so $c'_n = c_0 \cdot n_n = c_0 \cdot 1 = c_0$. If steps 3 and 4 of Algorithm 2 are finished, it is repeated until the last qubit a_{n-1} is checked. As shown in Figure 2, with the combination of ModulusMul and RotateRight, we allocate only $n-1$ qubits (i.e. $n_{n-1}, ..., n_1$) for modulus N ($n_n, ..., n_0$). Finally, the quantum resources required for each step are summarized as follows.

- **OperandMul**: n^2 Toffoli gates for schoolbook multiplication.
- **ModulusMul**: $n \cdot (n-1)$ Toffoli gates for modulus multiplication.
- **RotateRight**: $(n-1)$ Swap gates for right rotation.

Rotation right by 1 in Figure 2 can be performed with $(n-1)$ Swap gates. As shown in Figure 3, Swap (c_0, c'_1), Swap (c'_1, c'_2), ... Swap (c'_{n-2}, c'_{n-1}) ard done in order. However, Swap gates are not measured as quantum resources because they can be replaced by relabeling the qubits [16].

Finally, quantum resources required to implement Montgomery multiplication in $GF(2^n)$ as a quantum circuit are as follows.

- **Total**: $(4n-1)$ qubits and $(2n^2-n)$ Toffoli gates for Montgomery multiplication and the circuit depth is n^2+n.

3.2 Qubit-level Montgomery Multiplication with Specified Modulus

In the previous section, we presented Montgomery multiplication when the modulus is not specified, but if the modulus is specified, more efficient quantum circuit design between classic and quantum is possible. This section presents qubit-level Montgomery multiplication with specified modulus.

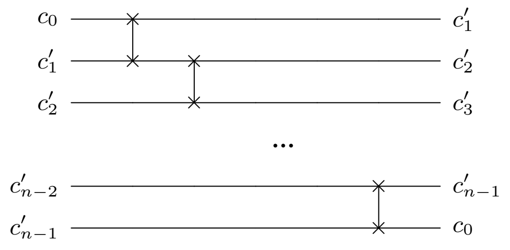

Fig. 3: RotateRight.

If the modulus is specified, qubits can be saved, since there is no need to allocate $(n-1)$-qubits for the modulus N and ModularMul is replaced by Algorithm 6, FixedModulusMul. In ModularMul, $(n-1)$-qubit modulus N, n-qubit C and Toffoli gates are used, but in FixedModulusMul, only n-qubit C and CNOT gates are used. Toffoli gate is actually composed of several CNOT gates, so the gate cost is reduced. The detailed process for qubit-level Montgomery multiplication with specified modulus is shown in Algorithm 5.

Algorithm 5 Qubit-level Montgomery multiplication quantum circuit with specified modulus.

```
Input: Specified modulus $N$, $n$-qubit operands $A$ and $B$, $n$-qubit result $C$.
Output: $C = A \cdot B \cdot r^{-1} \bmod N$.
1: for $i = 0$ to $n-1$ do
2:     $C$ = OperandMul $(a_i, B, C)$
3:     $C$ = FixedModulusMul $(c_0, N, C)$
4:     RotateRight $(C, 1)$
5: end for
6: return $C$
```

Algorithm 5 is not significantly different from Algorithm 2. The difference is step 2 FixedModulusMul which adds the product of modulus N and coefficient c_0 to C. In FixedModulusMul, the use of CNOT gates is determined according to the value of n_i, which is the coefficient of modulus N. Since we already know the modulus N, we can add the product of c_0 and N with only the CNOT gates. Also, there is no need to allocate qubits for modulus N. The process for FixedModulusMul is shown in Algorithm 6.

Algorithm 6 FixedModulusMul.

```
Input: Modulus $N$ and $n$-qubit result $C$.
Output: $C = C + c_0 \cdot N$.
1: for $i = 0$ to $n-2$ do
2:     if $n_{i+1} = 1$ then
3:         $c_{i+1}$ = CNOT $(c_0, c_{i+1})$
4:     end if
5: end for
6: return $C$
```

The advantages obtained in combination with step 4 (i.e. RotateRight) are the same as in the previous section. Details are shown in Figure 4.

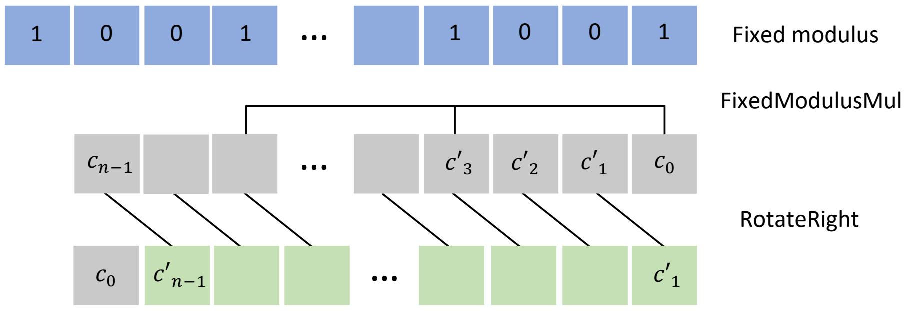

Fig. 4: FixedModulusMul + RotateRight.

The quantum resources required to implement Montgomery multiplication with specified modulus in $GF(2^n)$ as a quantum circuit are as follows. w is the sum of all coefficients of modulus N. For most modulus used in binary fields, w is not high. Therefore, for evaluation, we evaluated the depth by specifying the case of $w = 5$.

- **OperandMul**: n^2 Toffoli gates for schoolbook multiplication.
- **FixedModulusMul**: $n(w-2)$ CNOT gates for modulus multiplication.
- **RotateRight**: $(n-1)$ Swap gates for right rotation.
- **Total**: $3n$ qubits, n^2 Toffoli gates and $n(w-2)$ CNOT gates for Montgomery multiplication and the circuit depth is $6n-5$.

4 Evaluation

Many binary modular multiplication methods have been proposed for quantum computing. In [9], the multiplication part and the reduction part are efficiently connected using LU decomposition for the modulus. In [7, 10], the number of Toffoli gates was reduced by applying the Karatsuba algorithm, instead of increasing the number of qubits. In [11], the Karatsuba algorithm was applied similarly, but it reduced the number of Toffoli gates without increasing the qubits through LU decomposition for the modulus. However, these are all multiplications for a specified modulus. [7] computes Karatsuba products according to the modulus. [10] is also the same. For [9] and [11], LU decomposition for a specific modulus must be performed in advance.

When it comes to hardware implementation of arithmetic by targeting a specific field, it is fast and compact. However, if a field is changed, it must be completely redesigned, which is time consuming and expensive. The proposed binary Montgomery multiplication computes the product without knowing the modulus. That is, compute the Montgomery product on any modulus in $GF(2^n)$. We also implemented a quantum circuit where the operands of binary Montgomery multiplication are all quantum parameters. In [12], quantum resources were evaluated with one of operands A and B set as a classical parameter. In quantum computing, there are many operations that can be pre-computed without allocating qubits for classical parameters. Therefore, quantum-quantum computing has higher complexity than quantum-classical. We utilized ProjectQ [14], a quantum programming tool provided by IBM, to implement the proposed method. Based on this, implementation accuracy and quantum resources were evaluated. Required quantum resources are shown in Table 1, and the study results on the prime field [12] are also shown. In [12], the authors counted only Toffoli gates.

Table 1: Quantum resources for Montgomery multiplication.

Method	Approach	Qubit	Toffoli	CNOT	Depth
Prime field [12]	Quantum - Classical	$5n$	$20n^2$	$\times$	$8n\log_2 n$
		$3n$	$4n^2$	$\times$	$4n^2$
Binary field (ours)	Quantum - Quantum	$4n-1$	$2n^2-n$	$\cdot$	n^2+n
		$3n$	n^2	$3n$	$6n-5$

$w = 5$ (sum of coefficients of Modulus N)

5 Conclusion

In this paper, we focus on binary field multiplication using the Montgomery reduction algorithm. By utilizing the bit-level Montgomery algorithm, it is optimized for quantum computing in which qubit unit operations are performed. The proposed binary field Montgomery multiplication in quantum circuits computes the Montgomery product for any modulus. We also propose a more efficient quantum circuit when the modulus is specified. To the best of knowledge, this is the first work investigating the required quantum resources for binary field Montgomery multiplication.

The future work is going to find cryptography algorithm based on binary field multiplication and optimize quantum circuits. We will also explore to combine other asymptotically faster multiplication with the proposed binary field multiplication. We will find the optimal computation routine for this structure.

References

1. P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.
2. T. Häner, M. Roetteler, and K. Svore, "Factoring using $2n + 2$ qubits with toffoli based modular multiplication," *Quantum Information and Computation*, vol. 17, 11 2016.
3. C. Gidney, "Factoring with $n + 2$ clean qubits and $n - 1$ dirty qubits," *arXiv: Quantum Physics*, 2017.
4. M. Roetteler, M. Naehrig, K. Svore, and K. Lauter, "Quantum resource estimates for computing elliptic curve discrete logarithms," pp. 241–270, 11 2017.
5. T. Häner, S. Jaques, M. Naehrig, M. Roetteler, and M. Soeken, *Improved Quantum Circuits for Elliptic Curve Discrete Logarithms*, pp. 425–444. 04 2020.
6. G. Banegas, D. J. Bernstein, I. van Hoof, and T. Lange, "Concrete quantum cryptanalysis of binary elliptic curves," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 451–472, 2021.
7. S. Kepley and R. Steinwandt, "Quantum circuits for $\mathbb{F}_{\nvDash}^{\ltimes}$-multiplication with subquadratic gate count," *Quantum Information Processing*, vol. 14, no. 7, pp. 2373–2386, 2015.
8. S.-M. Cho, A. Kim, D. Choi, B.-S. Choi, and S.-H. Seo, "Quantum modular multiplication," *IEEE Access*, vol. 8, pp. 213244–213252, 2020.
9. D. Cheung, D. Maslov, J. Mathew, and D. K. Pradhan, "On the design and optimization of a quantum polynomial-time attack on elliptic curve cryptography," 2009.
10. K. Jang, S. J. Choi, H. Kwon, Z. liang Hu, and H. Seo, "Impact of optimized operations a· b, a· c for binary field inversion on quantum computers," in *WISA*, 2020.
11. I. van Hoof, "Space-efficient quantum multiplication of polynomials for binary finite fields with sub-quadratic toffoli gate count," 2020.
12. R. Rines and I. Chuang, "High performance quantum modular multipliers," 2018.
13. Koç and T. Acar, "Montgomery multiplication in gf(2k)," *Designs, Codes and Cryptography*, vol. 14, pp. 57–69, 1998.
14. D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: an open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, 2018.
15. P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.
16. M. Znidaric, O. Giraud, and B. Georgeot, "How many cnot gates does it take to generate a three-qubit state?," *arXiv preprint arXiv:0711.4021*, 2007.