Optimal Implementation of Quantum Circuits of Core Operations in BIKE

Yujin Yang[1], Kyungbae Jang[2], Yujin Oh[3], Sejin Lim[4], and Hwajeong Seo[5]

Hansung University, Seoul, South Korea
yujin.yang34@gmail.com[1], starj1023@gmail.com[2], oyj0922@gmail.com[3], dlatpwls834@gmail.com[4], hwajeong84@gmail.com[5]

Abstract

Quantum computers, which are rapidly being developed by Google, IBM, and many other companies, are threatening the safety of public key cryptography algorithms based on the problem, coupled with the emergence of Shor algorithm known to solve small factorization and discrete logarithm problems in polynomial time. In order to respond to upcoming security vulnerabilities, NIST hosted a Post-Quantum Cryptography Contest, selected 4 standardized algorithms, and selected 3 code-based cryptography as alternative algorithms. Implementing post-quantum cryptography and conducting safety testing in a quantum computer environment are of significant importance, particularly prior to the transition to post-quantum cryptography. Therefore, in this paper, core arithmetic operations in the phase used in the key generation and encapsulation stage of BIKE, one of the Round 4 alternative algorithms, are optimized and implemented as quantum circuits and the quantum resources are estimated and analyzed.

1 Introduction

Quantum computers developed based on quantum mechanics techniques such as quantum superposition and entanglement can solve certain problems faster than classical computers [9]. Quantum computers are being developed by many companies and research teams. Google, one of the leading companies in quantum computer development, has focused on correcting errors that occur during quantum computer operations, starting with proving quantum supremacy in 2019 [5]. As a result, in 2022, a low error rate of 2% was achieved by increasing the density of qubits and reducing interference [1]. IBM unveiled the most recent (November 2022) 433-qubit quantum processor Osprey, following the 27-qubit quantum processor Falcon in 2019. Development is being carried out sequentially according to the quantum computer development roadmap announced by IBM, and their goal is to develop a processor that provides more than 4,000 qubits by 2025 [13].

Known as one of the representative quantum algorithms, the Shor algorithm can solve prime factorization and discrete logarithmic problems that take exponential time in polynomial time [24]. Since Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC), which are public key cryptosystems, are mathematically based on prime factorization and discrete log problems, this algorithm is known to be a threat to the safety of public key cryptography. Recently, researches to solve the prime factorization problem [7], which is the mathematical basis of RSA cryptography, and to estimate quantum resources required for attacks of RSA and ECC cryptography using the Shor algorithm are being actively conducted [11, 6, 21, 12]. As such, with the advancement of quantum computers and the emergence of quantum algorithms, the security of cryptographic algorithms is threatened.

In order to respond to the upcoming security vulnerability of public key cryptography, the National Institute of Standards and Technology (NIST) in the United States hosted the

Post-Quantum Cryptography (PQC) contest in 2017. In July 2022, 4 standardization algorithms (3 grid-based and 1 hash-based) were finally selected. As an alternative to the problems concentrated on lattice-based cryptography, three code-based cryptography were selected as alternative algorithms for Round 4. Currently, NIST is in the process of documenting the PQC standardization algorithm and conducting an additional competition for the digital signature algorithm [20]. Prior to the transition to PQC, it is very important to implement PQC in a quantum computer environment that will occupy a larger pie in the future and ultimately test its safety.

In this paper, we present a quantum circuit implementation of core operations on the phase used in BIKE [4], one of the alternative cryptography algorithms of the NIST PQC Standardization Contest Round 4. In order to optimize our quantum circuits, we apply optimized binary field arithmetic operations in core operations such as key generation, encapsulation, and decapsulation. We estimate and analyze the quantum resources used for each core operation.

2 Background

2.1 BIKE

BIKE [4] is one of the ciphers finally selected as an alternate candidate algorithm in the NIST PQC standardization contest. It is a Key Encapsulation Mechanism (KEM) based on the Quasi-cyclic Moderate-density parity-check (QC-MDPC) code. The QC-MDPC code contains all information in the first row to represent it in the form of a quasi-cyclic matrix. The size of the public key is small because there is no need to store all the information in the matrix if only the corresponding row is stored. In addition, in the case of a quasi-circular matrix, the speed of matrix multiplication and encoding is much faster. However, it has a disadvantage in that there is a probability of failure of the decoding algorithm.

Table 1: Parameter size for each BIKE security level

Security	r	w	t	l	DFR
Level 1	12,323	142	134	256	2^{-128}
Level 3	24,659	206	199		2^{-192}
Level 5	40,973	274	264		2^{-256}

Table 1 shows the parameter values of BIKE. r, w, t and l indicate the size of block, row weight value, error weight value, and shared secret value. And Decryption Failure Rate (DFR) means the probability of error correction failure.

The process of BIKE consists of 3 steps and is given as:

2.1.1 Key Generation

This process generates public key h using secret keys h_0, h_1. The secret keys h_0, and h_1 are generated through the function **H**, which uses SHAKE256 hash function based pseudorandom expansion. The public key is generated through the following equation:

$$h = h_1 h_0^{-1} \tag{1}$$

2.1.2 Encapsulation

This process is to encrypt the messages m and to exchange shared key K. Error vector e_0 and e_1 for message m are generated through hash function $\mathbf{H}$, and ciphertext c is generated using sample error vector e_0 and e_1, public key h, message m, and the function $\mathbf{L}$, which uses the standard SHA3-384 hash function. Equation 2 corresponds to this process. And then, in order to exchange key, the shared key K is generated using message m, ciphertext c, and the function $\mathbf{K}$, which relies on SHA3-384 hash function.

$$c \mapsto (e_0 + e_1 h, m \oplus \mathbf{L}(e_0, e_1)) \tag{2}$$

2.1.3 Decapsulation

In this step, the decoding algorithm is executed to decapsulate the encapsulated ciphertext. This process is to execute the decoding algorithm. When a valid error vector e' is recovered by inputting the secret keys and ciphertext into the decoder, a message vector m' is obtained through this and another shared key K' is generated and verified.

2.2 Arithmetic Operations in Binary Fields $GF(2^n)$

$GF(2^n)$ is a finite field containing 2^n different elements, and can be expressed as a polynomial of order $n-1$ or less. It is also referred to as a binary field, and denoted $\mathbb{F}_2$ or $\mathbf{F}_2$. An irreducible polynomial is the smallest non-constant polynomials that can no longer be factored, and is mainly used to define multiplication beyond n-degree order in the $GF(2^n)$ environment. An irreducible polynomial has a feature that several polynomials exist as the degree increases, and is used in most cryptography design.

The arithmetic operations in these binary fields differ somewhat from the arithmetic operations we are familiar with in general. In the case of addition, since modulo-2 addition(XOR) is performed, an XOR operation($\oplus$) is simply used instead of an adder composed of AND and OR gates, and carry does not occur. In the finite field, subtraction is considered the same as addition.

Like addition, carry does not occur in multiplication, but it requires more tedious work, unlike addition, which is simply defined as XOR operation. After multiplying through multipliers, it is necessary to perform modular reduction, which reduces the polynomial to fit the field through irreducible polynomials. Since performance varies depending on which multiplier is used, various multipliers applying multiplication algorithms such as Karatsuba[17], Schoolbook, and Montgomery[18] have been proposed.

Since the squaring operation also corresponds to the multiplication operation in which the same number is multiplied multiple times, the modular reduction must be performed after the squaring operation. In order to increase the operation speed, modular reduction generally converts an irreducible polynomial with linearity into a matrix form and proceeds with the operation. Since this corresponds to a linear operation, it can be implemented as an in-place structure using only the XOR operation by using the PLU decomposition.

The multiplicative inverse can be can be obtained using the Itoh-Tsuji Inversion algorithm [14], the extended binary gcd algorithm (EBGA) [25], or the montgomery inverse algorithm (MIA) [2], etc. Among them, the Itoh-Tsuji algorithm is the most used, and because multiplication and squaring operations are used together, it has the highest computational complexity among operations in binary finite fields.

2.3 Quantum Gates

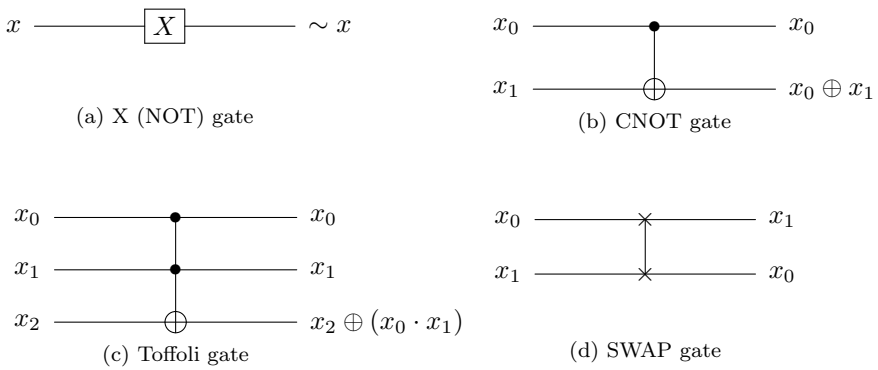
Since in the quantum computer environment they do not provide logic gates such as AND, OR, and XOR, quantum gates are used as replacements for logic gates. In this section, we go over the quantum gates that are required to carry out cryptographic procedures.

Figure 1 depicts quantum gates that are crucial and frequently utilized in the construction of quantum cipher circuits. The simplest reversible logic gate is the X gate (also known as the NOT gate) in Figure 1(a). It is comparable to the traditional bit flip.

The Swap Gate shown in Figure 1(d) exchanges values for a 2-qubit input. SWAP gates are considered not to have quantum cost.

The CNOT gate shown in Figure 1(b) stands for a controlled-not gate and can produce entangled states contrary to the single-qubit gates. Of the two input qubits, one is the control qubit and the other is the target qubit. If the control qubit is $|1\rangle$, the target qubit state is fliped. That is, the control qubit decides the target qubit state. Normally, CNOT gates are used very often in circuits.

Figure 1(c) shows a Toffoli gate which is a controlled-controlled-not gate. Two of the three inputs are control qubits and the other one is a target qubit. The target qubit is affected only when all control qubits are $|1\rangle$. Just as X gate is replaced with NOT gate and CNOT gate is replaced with XOR, Toffoli gate is generally considered to replace with AND gate. This gate is the most quantum costly of the four gates. The Toffoli gate consists of a CNOT gate and T, H, and S gates. The T-count of the standard Toffoli gate [19] is 7 and the T-depth is 6. Many studies are reporting the implementation of Toffoli gate circuits with minimized depth and T-depth [3, 10, 22, 23, 16].



(a) X (NOT) gate (b) CNOT gate (c) Toffoli gate (d) SWAP gate

Figure 1: Quantum gates.

3 Quantum Implementation of BIKE

BIKE consists of Key generation, Encapsulation, and Decapsulation processes. In this section, we implement the most key operations in the key generation and encapsulation processes with quantum circuits since there is no probability of failure in these two processes.

Since even the lowest level of BIKE proceeds in the $GF(2^{12323})$ environment, it requires so many resources that it cannot be executed as a quantum simulator. Due to this reason, the size of the field has been reduced to facilitate implementation. As per the BIKE's design principle, the irreducible polynomial $f(X)$ in $GF(2^r)$ is $(X^r - 1)/(X - 1)$, and the block size, denoted by

r, must be a prime number. Assuming $r = 13$ for implementation, the corresponding irreducible polynomial is as follows:

$$f(X) = X^{12} + X^{11} + \cdots + X^2 + X + 1 \tag{3}$$

In order to implement the quantum circuit, ProjectQ, a quantum programming tool developed by IBM, was utilized.

3.1 Key Generation

The most important part of the key generation part is to obtain the public key h. The public key h is composed of the private keys h_0 and h_1, which are generated through a sparse vector. Equation 1 represents the process of generating a public key using these two private keys.

For this calculation, a multiplicative inversion operation to find the multiplicative inverse of h_0 and a multiplication operation to find the product between the resulting h_0^{-1} and h1 are required. The multiplicative inverse of h_0 can be obtained by applying the Itoh-Tsuji Inversion algorithm [14]. The Itoh-Tsuji algorithm consists of a multiplication operation and a squaring operation. For example, if the Itoh-Tsuji algorithm is used to find the multiplicative inverse of element a of an irreducible polynomial in a $GF(2^n)$ environment, a^{-1} can be obtained by performing the same process as Equation 4 [8].

$$a^{-1} = a^{254} = ((a \cdot a^2) \cdot (a \cdot a^2)^4 \cdot (a \cdot a^2)^{16} \cdot a^{64})^2 \tag{4}$$

For the multiplication operation, a quantum multiplier on a binary finite field proposed by Jang et al. [15] is used. This quantum multiplier reduces the number of Toffoli gates used by recursively applying the Karatsuba multiplication algorithm. Toffoli-depth is optimized to 1 by additionally allocating auxiliary qubits and performing multiplication in parallel. We additionally assign 324 auxiliary qubits to suit this implementation environment and implement the multiplier in [15] consisting of only CNOT gates and Toffoli gates.

The squaring operation uses a matrix of in-place structure generated by an irreducible polynomial. The squaring operation is implemented using CNOT gates and SWAP gates based on the corresponding matrix. The cost of the squaring operation is solely determined by the number of CNOT gates because the SWAP gates do not account for the cost.

Equation 5 shows the result of applying the Itoh-Tsuji algorithm to find the multiplicative inverse of h_0.

$$h_0^{-1} = h_0^{4094} = ((h_0 \cdot h_0^2)^{512} \cdot (h_0 \cdot h_0^2)^{128} \cdot (h_0 \cdot h_0^2)^{32} \cdot (h_0 \cdot h_0^2)^{16} \cdot (h_0 \cdot h_0^2)^2 \cdot h_0)^2 \tag{5}$$

We use h_0^{-1} obtained in this way(Equation 5) and h_1 as inputs, and utilize the quantum multiplier to obtain the public key h.

3.2 Encapsulation

In the encapsulation process that performs encryption, the key is to obtain the ciphertext c. The first ciphertext c_0 is obtained by multiplying the randomly determined error vector e_1 with the public key h and adding the resulting value to another error vector e_0. This multiplication operation also utilizes the quantum multiplier [15]. The multiplication result of e_1 and h is stored in h.

In the case of addition here, XOR operation is performed because addition arithmetic operation in binary field means modulo-2 addition. In a quantum computing environment, since

the XOR operation can be replaced with a CNOT gate, the CNOT gates are applied with e_0 as the control qubit and h as the target qubit.

Afterwards, assuming that the result value obtained by putting e_1 and e_1 into hash function $\mathbf{L}$ is stored in e_1, and designating e_0 as the control qubit and message m as the target qubit, and applying the CNOT gates, the second ciphertext c_1 is obtained. Finally, the ciphertext c is the combination of c_0 and c_1.

4 Quantum Resources Estimation and Analysis

In this section, we estimate and analyze the quantum circuit resources for the implementation of step-by-step core arithmetic operations described in Section 3. The proposed quantum circuits cannot yet be implemented in large-scale quantum computers. Therefore, we use ProjectQ, a quantum programming tool, on a classical computer instead of real quantum computer to implement and simulate quantum circuits.

A large number of qubits can be simulated using ProjectQ's own library, ClassicalSimulator, which is restricted to simple quantum gates (such as X, SWAP, CNOT, and Toffoli). With the aid of this functionality, the ClassicalSimulator is able to test the implementation of a quantum circuit by classically computing the output for a particular input. For the estimation of quantum resources, another internal library called ResourceCounter is needed. ResourceCounter solely counts quantum gates and circuit depth, doesn't run quantum circuits, in contrast to ClassicalSimulator.

Core arithmetic operations include multiplicative inverse operation, multiplication operation, and addition operation. Table 2 shows the quantum resource estimates used when implementing quantum circuits of core arithmetic operations at each stage of BIKE which has Equation 3 as an irreducible polynomial (that is $r = 13$) in $GF(2^{r-1})$. In the Table 2, $\#T$ means the number of T gates, #CNOT means the number of CNOT gates, $\#H$ means the number of H gates, and Full depth means the overall depth when Toffoli gates are decomposed.

Looking at Table 2, it can be seen that quantum resources are generally used in the key generation step. This is because the multiplication operation, which incurs the highest cost among the arithmetic operations, dominates in the corresponding step. Even in the encapsulation stage, the part that occupies the largest proportion of quantum resources is the multiplication operation.

Looking at the T-depth with the greatest optimization, the Toffoli gate [3] used in this implementation consists of 8 Clifford gates and 7 T-gates, and has a T-depth of 4. Therefore, the quantum multiplier of [15] with a Toffoli-depth of 1 has a T-depth of 4 every time it is used. Toffoli gates are used only for multiplication operations. In the key generation part, a total of 5 multiplication operations are used: 4 times in the inversion operation and 1 time in the multiplication between private keys. As a result, T-depth is $4 \times 5 = 20$. In the encapsulation part, the multiplication operation is used only once when obtaining the ciphertext c_0, so the T-depth is 4.

Table 2: Quantum circuit implementation resource estimation of BIKE's step-by-step core arithmetic operations on $\mathbb{F}_{2^{12}}/(X^{12} + X^{11} + \cdots + X^2 + X + 1)$

Step (arithmetic operations)	#Qubits	#CNOT	$\#T$	$\#H$	T-depth	Full depth
KeyGen (Inversion & Multiplication)	468	5,071	2,268	648	20	300
Encaps (Multiplication & Addition)	174	939	378	108	4	40

5 Conclusion

In this paper, a quantum circuit is optimized and implemented using an efficient squaring algorithm [14] and an optimized quantum multiplication [15] for core arithmetic operations on the phase used in the BIKE code-based cryptography, which is an alternative algorithm for the NIST PQC Contest Round 4. In addition, based on the implementation results, quantum resources (such as depth, number of qubits, and number of quantum gates) used in each operation are estimated. It is expected that this quantum circuit will serve as a cornerstone for BIKE's security strength analysis research.

As a future work, we will conduct research on creating quantum circuits by expanding the range of finite field $GF(2^r)$. It would be really nice to apply the actual values of each level of BIKE, but it is still impossible to implement with the resources provided by the quantum computer simulator. Therefore, we will figure out the increasing trend of quantum resources according to each field size and finally calculate the quantum resources required to implement quantum circuits of BIKE levels 1, 3, and 5.

References

[1] Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023.

[2] Adnan Abdul-Aziz Gutub, Alexandre F Tenca, Erkay Savaş, and RCCetin K Koç. Scalable and unified hardware to compute montgomery inverse in gf (p) and gf (2 n). In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 484–499. Springer, 2003.

[3] Matthew Amy, Dmitri Maslov, Michele Mosca, Martin Roetteler, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, Jun 2013.

[4] Nicolas Aragon, Paulo SLM Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. Bike: bit flipping key encapsulation - round 4 submission. 2022.

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[6] Gustavo Banegas, Daniel J Bernstein, Iggy Van Hoof, and Tanja Lange. Concrete quantum cryptanalysis of binary elliptic curves. *Cryptology ePrint Archive*, 2020.

[7] Vaishali Bhatia and KR Ramkumar. An efficient quantum computing technique for cracking rsa using shor's algorithm. In *2020 IEEE 5th international conference on computing communication and automation (ICCCA)*, pages 89–94. IEEE, 2020.

[8] Amit Kumar Chauhan and Somitra Kumar Sanadhya. Quantum resource estimates of grover's key search on aria. In *Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10*, pages 238–258. Springer, 2020.

[9] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[10] Arkady Fedorov, Lars Steffen, Matthias Baur, Marcus P da Silva, and Andreas Wallraff. Implementation of a Toffoli gate with superconducting circuits. *Nature*, 481(7380):170–172, 2012.

[11] Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[12] Élie Gouzien, Diego Ruiz, Francois-Marie Le Régent, Jérémie Guillaud, and Nicolas Sangouard. Computing 256-bit elliptic curve logarithm in 9 hours with 126133 cat qubits. *arXiv preprint arXiv:2302.06639*, 2023.

[13] IBM. Ibm quantum development road map, 2023.

[14] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases. *Information and computation*, 78(3):171–177, 1988.

[15] Kyungbae Jang, Wonwoong Kim, Sejin Lim, Yeajun Kang, Yujin Yang, and Hwajeong Seo. Optimized implementation of quantum binary field multiplication with toffoli depth one. In *International Conference on Information Security Applications*, pages 251–264. Springer, 2022.

[16] Cody Jones. Low-overhead constructions for the fault-tolerant toffoli gate. *Physical Review A*, 87(2):022328, 2013.

[17] Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

[18] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[19] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[20] NIST. Post-quantum cryptography, 2023.

[21] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, and Howon Kim. Another concrete quantum cryptanalysis of binary elliptic curves. *Cryptology ePrint Archive*, 2022.

[22] TC Ralph, KJ Resch, and Alexei Gilchrist. Efficient Toffoli gates using qudits. *Physical Review A*, 75(2):022313, 2007.

[23] Peter Selinger. Quantum circuits of t-depth one. *Physical Review A*, 87(4):042302, 2013.

[24] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[25] Yasuaki Watanabe, Naofumi Takagi, and Kazuyoshi Takagi. A vlsi algorithm for division in gf (2 m) based on extended binary gcd algorithm. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 85(5):994–999, 2002.