Implementation of the NewHope Protocol for Post-Quantum Cryptography

Zhenghao Lin, Guanghua Chen*

School of Mechatronic engineering and Automation, Shanghai University, Shanghai, China

*Corresponding author: chghua@shu.edu.cn

Abstract. In this paper, we propose a faster and less resource intensive implementation of the NewHope protocol to address the problems of resource-intensive random number generators and slow running NTT modules in NewHope implementations. In the random number generation module, choose the lightweight pseudo-random number generator (PRNG) Trivium instead of the SHAKE function to reduce the resource usage in the random number generation module. In the NTT transformation module, a pipeline structure is used in combination with a parallel structure of four butterfly units to shorten the cycle time of butterfly operations and achieve acceleration. The experimental results show that, compared with Kuo's design, the overall time for key exchange in this paper is reduced by 25.3%, and the consumption of FFs and LUTs is reduced by 20.2% and 38.0%, respectively.

Keywords: polynomial ring, Post-Quantum Cryptography, key exchange, Ring-LWE, NewHope.

1. Introduction

With the advent of the Shor [1] quantum algorithm, key schemes and digital signature schemes based on large integer decomposition problems, discrete logarithm problems, and elliptic curves in traditional cryptography will be most severely affected. Thus, it is urgent to investigate how to construct schemes and mathematical problems that are resistant to quantum computing attacks. Post-quantum cryptography has attracted much attention in the cryptography community.

NewHope, first proposed by Alkim et al. [2], is a post-quantum key exchange scheme based on a grid. So far, there are not many studies related to the hardware implementation of NewHope series schemes. Oder [3] was the first to implement the NewHope series algorithm in hardware. Oder limited the design to two DSPs to solve the medium throughput problem. The NTT calculations are performed using a single butterfly unit architecture to reduce the hardware area consumption. However, this also limits the speed of its NTT modules. Compared to Oder's work, the PRNG in Kuo's [4] design takes up most of the resources. The NTT module is designed with reference to the pipeline structure of Chen [5], and the two butterfly units are used to process NTT operations in parallel, which speeds up NTT operations to a great extent. Kuo's solution is 19.1 times faster than Oder's in terms of total time spent, but uses four times the resources.
In this paper, we designed a less resource-intensive and faster hardware solution to implement the NewHope protocol. Choose lightweight PRNG instead of the SHAKE function to greatly reduce resource usage. Design a parallel structure of four butterfly units in the NTT transformation section to speed up the NTT stage operation. Use constant geometry NTT instead of the traditional in-place NTT algorithm to optimize the data storage pattern to speed up the implementation process.

2. Introduction to the NewHope

2.1. NewHope key exchange protocol

The NewHope protocol is shown in Protocol 1, where Alice and Bob represent the two parties of encryption and decryption, respectively.

Parameter: $q = 12289 < 2^{14}$, $n = 1024$
Error distribution: $\Psi_8$

| Alice | Bob |
|-------|-----|
| seed $\leftarrow \{0,1\}^{256}$ | $s', e', e' \leftarrow \Psi_8$ |
| $a \leftarrow$ Parse($\text{SHAKE-128(seed)}$) | $a \leftarrow$ Parse($\text{SHAKE-128(seed)}$) |
| $s, e \leftarrow \Psi_8$ | $u \leftarrow a's' + e'$ |
| $b \leftarrow$ as + e $\in R_q$ | $\mu \leftarrow \{0,1\}^{256}$ |
| $(b, seed)$ | $v \leftarrow$ Encode($\mu$) $\in R_q$ |
| $v' \leftarrow$ Decompress($h$) | $v' \leftarrow b's' + e'' + v \in R_q$ |
| $v \leftarrow v' - us$ | $h \leftarrow \text{Compress}(v')$ |
| $\mu \leftarrow$ Decode($v$) | |

Protocol 1. NewHope Key Exchange Protocol,

Alice calculates the left part and Bob calculates the right part, where $b$ and $s$ are the public and private keys on the Alice side, $u$ and $s'$ are the public and private keys on the Bob side, $a$ is the public parameter, and $e$ and $e'$ are the error terms. The public key $b$ for Alice and $u$ for Bob. The key generation phase Alice side generates a random array seed and generates public parameters $a$, error $e$ and private key $s$ by sampling the seed through PRNG. Finally, the NTT post-transformation operation generates the public key polynomial $b$. The public keys $b$ and seed are passed to Bob as output. In the second stage, Bob gets the public key and generates errors $e'$, $e''$ and private key $s'$ using the same seed. The message $\mu$ is encrypted to generate $v$, which is then computed to generate $v'$. The secret message $h$ is compressed and sent to Alice.

2.2. NewHope Implementation

In the previous section, we described the encryption and decryption process of the NewHope protocol. According to the principle of the algorithm, the implementation involves the generation of random polynomials $a$, $e$, $e'$, $e''$, $s$, $s'$, NTT transformation, modal reduction and encryption compression. Thus, the hardware implementation of this paper is mainly divided into random number generator, NTT module, modal reduction and encryption compression. Figure 1 shows a flow diagram of the hardware implementation scheme designed in this paper.
Figure 1. Flowchart of hardware implementation.

Take the key generation process as an example: the TRNG design uses the ring oscillator proposed by Wold [6] to generate the seed. The seed is fed into the pseudo-random number generator SHAKE function to generate the public parameter \(a\), and the error \(e\) and private key \(s\) are generated by the lightweight PRNG Trivium. \(a\), \(e\), and \(s\) as inputs to the NTT conversion circuit, performing polynomial operations in the NTT domain to generate the public key \(b = a \cdot s + e\). The sequence of each module is shown in Figure 2. The modal reduction algorithm proposed by Patrick [7] is used in this paper because most of the operations in this algorithm are bit addition and subtraction operations, which are more suitable for hardware implementation. Encryption compression uses the method in Tomas's work [8] to reduce the size of a cipher without affecting the bit error rate by shedding the signal at a certain minimum number of bits.

Figure 2. Sequence Diagram of the key generation phase

In the whole encryption process of NewHope, the NTT transform is the key to implementing the NewHope algorithm, both in hardware and software implementations. So the NTT transform circuit is the focus of the design of this paper. The main component unit of NTT is the butterfly unit, and the core of the design is the butterfly computing unit. Choosing a parallel architecture to increase implementation speed requires a large area of resources, while a serial architecture sacrifices implementation speed for reduced resource use. The main challenge in NTT module design is how to balance implementation speed and resource-using area. The design of the NTT module is described in detail in the next section.
3. NTT Module

In theory, the implementation of NTT partial circuits requires three parts: a butterfly arithmetic unit, a twiddle factor multiplier, and bit reversal. In the study of Alkim, it is pointed out that since the input coefficients of the NTT module are randomly generated, skipping the bit inversion process in the hardware implementation and not performing bit inversion operation in the subsequent encryption and decryption process can speed up the NTT process by about 40%, without affecting the security level of the encryption scheme.

The core of the NTT transformation is the butterfly operation, and the principle of the 8-point butterfly operation is shown in Figure 3. A single butterfly unit consists mainly of an adder, subtracter, and multiplier. A butterfly unit requires two inputs and outputs two results from one butterfly operation. The 2^n-point NTT operation requires \( \log_2 2^n = n \) phases of the NTT operation, and \( 2^{n-1} \) butterfly operations per phase of the NTT operation. Theoretically, a 1024-point NTT can complete a phase of butterfly operations in one cycle using 512 butterfly units operating in parallel. In order to reduce resource usage, pipeline structure is often used in NTT implementations to reduce the use of butterfly units. The difference between this paper's design and Oder's design is that four butterfly units are used to form a parallel structure to speed up NTT operations. Each butterfly cycle can compute 8 bits of data simultaneously, and the design of this paper is based on a single 14-bit piece of data, which means that 112 bits of data need to be computed simultaneously. The maximum data width that BRAM (Block RAM) can support in Cyclone III series FPGAs is 36 bits. To satisfy the 112-bit data required for this design, at least 4 BRAMs are used. The low data depth makes the actual utilization of each piece of BRAM extremely low. In contrast, distributed RAM is more flexible and has better timing performance than BRAM. In order to increase the computational efficiency and timing performance of the NTT module, distributed RAM is used to store the NTT phase transition data. In addition, two RAMs are designed to store the twiddle factors used in the NTT and inverse NTT phases, and calculating the stored twiddle factors in advance saves time in calculating the twiddle factors for each calculation.

![Figure 3. 8-point constant geometry NTT butterfly schematic.](image)

The input and output memory addresses of traditional in-place NTT remain the same, but the memory addresses are different at each stage of the computing process. It means reading and writing addresses with different control logic at each stage, adding a lot of work. As can be seen in Figure 3, constant geometry NTT have the same data path in the intermediate stages, making NTT calculations more efficient and well suited for hardware implementation of pipelined structures. This paper is designed to improve the computational speed of NTT module, so it uses constant geometry NTT instead of in-place NTT as an optimization. Eight consecutive points are extracted from RAM at a time as inputs to the four butterfly units. If the read and write data is stored in a RAM, you can't read and write to an address at the same time, resulting in a read-write conflict. To avoid such problems, the NTT module is optimized
for reading and writing data using ping-pong storage. Add a piece of RAM to be used alternately, and the result will still be stored in the same RAM after 10 levels of NTT transformation.

4. Result and Conclusion

4.1. Experimental Result
The FPGA device is Cyclone III EP3C80F780I7, which is simulated by ModelSim and Quartus II. The experimental screenshot of the NTT part of the simulation process is shown in Figure 4. Figure 4 shows the results of the NTT variation of the common parameter a. Setting the same input signal and comparing the simulation results with the MATLAB programming results. The two results are consistent and validate the correct design of the NTT part.

![Figure 4. Alice side NTT partial simulation screenshot.](image)

Since the hardware implementation of NewHope is relatively little work, this paper mainly compares the design with Kuo's design. Kuo's design uses the Xilinx Zynq-7000 series of chips. The overall resource consumption of the two solutions and the resource consumption of each part are listed in Table 1.

| Component     | NewHope(our) | NewHope(Kuo) |
|---------------|--------------|--------------|
|               | FFs | LUTs | DPSs | FFs | LUTs | DPSs | BRAMs |
| TRNG          | 276 | 330 | 0    | 258 | 310 | 0    | 0     |
| Trivium(PRNG) | 293 | 389 | 0    | —   | —   | —    | —     |
| SHAKE128(PRNG)| 1522| 4351| 0    | 2976| 3516| 0    | 0     |
| NTT(Alice)    | 2876| 4784| 8    | 1381| 2832| 8    | 10    |
| NTT(Bob)      | 2973| 4890| 8    | 1381| 2832| 8    | 10    |
| Overall Resource Use | 7626/7854 | 11453/13089 | 8/8 | 9412/9975 | 18756/20826 | 8/8 | 14/14 |
| Time/µs       | 113.2|     |      | 51.9/78.6/21.1 |

As can be seen in Table 1, the use of FFs in the NTT part of this paper is 2.11 times that of Kuo, and the use of LUTs is 1.61 times that of Kuo. This is because this design uses four butterfly units and uses distributed RAM instead of BRAM, which causes this design to use more logical resources in the NTT phase. However, in the penultimate row of Table 1, two scenarios are recorded as consuming resources on the Alice side and on the Bob side. A total of 7626 FFs and 11453 LUTs were used on the Alice side, and 7854 FFs and 13089 LUTs were used on the Bob side. Compared to Kuo's scheme, the scheme designed in this paper reduces the use of FFs and LUTs by 20.2% and 38.0%, respectively.
4.2. Conclusion
In this paper, we design a less resource-intensive and faster hardware implementation of the NewHope protocol based on FPGAs. The Random Number Generator module uses the lightweight random number generator Trivium to replace the SHAKE function, significantly reducing resource consumption. The NTT module uses four butterfly units to accelerate the butterfly operation, using constant geometry NTT instead of traditional in-situ NTT, and optimizing the data storage mode of the NTT transformation module. This solution achieves the entire key exchange process in 113.2μs, which is 33.9% faster than Kuo's solution. In terms of the overall hardware resource consumption, this scheme consumes 24,542 LUTs and 15,480 FFs, which is 20.2% and 38.0% lower than Kuo's design in terms of FFs and LUTs consumption.

References
[1] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”, in SIAM Journal on Computing, 1995.
[2] E. Alkim, L. Ducas, et al., “Post-quantum key exchange-A new hope”, in Proc. 25th USENIX Secur. Symp. (USENIX Secur.), 2016, pp. 327-343.
[3] T. Oder and T. Guneysu. “Implementing the newhope-simple keyexchange on low-cost fpgas”, in Progress Cryptology-LATINCRYPT, T. Lange, O. Dunkelman (Eds.), Cham, Switzerland: Springer, 2019, pp. 128-142.
[4] P. C. Kuo, W. D. Li, Y. W. Chen, et al., “High performance post-quantum key exchange on fpgas, Cryptol”, ePrint Arch., Tech. Rep. 2017/690, 2017.
[5] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, and I. Verbauwhede, “High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems”, Circuits & Systems I: Regular Papers, IEEE Transactions on, 62 (1) (2015), 157-166.
[6] K. Wold, C. H. Tan, “Analysis and enhancement of random number generator in FPGA based on oscillator rings”, International Journal of Reconfigurable Computing, 2009: 501672:1–501672:8, 2009.
[7] L. Patrick, M. N. Patrick, “Speeding up the number theoretic transform for faster ideal lattice-based cryptography”, International Conference on Cryptology and Network Security Springer International Publishing, 2016.
[8] P. Thomas, T. Güneysu, “Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware.” International Conference on Selected Areas in Cryptography Springer, Berlin, Heidelberg, 2013.
[9] R. Oded, "On lattices, learning with errors, random linear codes, and cryptography." Proceedings of the Annual ACM Symposium on Theory of Computing 56.6(2009).
[10] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, “Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme,” Ima International Conference on Cryptography & Coding Springer, Berlin, Heidelberg, 2013.
[11] C. Peikert, “A Decade of Lattice Cryptography”, Foundations & Trends® in Theoretical Computer Science 10.4(2016):283-424.
[12] S. S. Roy, et al., “Compact Ring-LWE Cryptoprocessor”, International Workshop on Cryptographic Hardware & Embedded Systems Springer, Berlin, Heidelberg, 2014.
[13] N. Gitert, et al. “On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes”(2012).
[14] Y. Xing, S. Li, “An Efficient Implementation of the NewHope Key Exchange on FPGAs”, Circuits and Systems I: Regular Papers, IEEE Transactions on PP.99 (2019): 1-13.