JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 37, 1211-1229 (2021)
DOI: 10.6688/JISE.202109_37(5).0015

High Performance Post-Quantum Key Exchange on FPGAs

PO-CHUN KUO[1], YU-WEI CHEN[1], YUAN-CHE HSU[1], CHEN-MOU CHENG[1],
WEN-DING LI[2] AND BO-YIN YANG[2]
[1]*Department of Electrical Engineering*
National Taiwan University
Taipei, 116 Taiwan
[2]*Institute of Information Science*
Academia Sinica
Taipei, 115 Taiwan
E-mail: kbj@crypto.tw; r05921032@ntu.edu.tw; r05921056@ntu.edu.tw;
doug@crypto.tw; thekev@crypto.tw; by@crypto.tw

Lattice-based cryptography is a highly potential candidate that protects against the threats of quantum attack. At Usenix Security 2016, Alkim, Ducas, Pöpplemann, and Schwabe proposed a post-quantum key exchange scheme called NewHope, based on a variant of lattice problem, the ring-learning-with-errors (RLWE) problem. In this work, we propose a high performance hardware architecture for NewHope. Our implementation requires 6,680 slices, 9,412 FFs, 18,756 LUTs, 8 DSPs and 14 BRAMs on Xilinx Zynq-7000 equipped with 28mm Artix-7 7020 FPGA. In our hardware design of NewHope key exchange, the three phases of key exchange costs 51.9, 78.6 and 21.1 μs, respectively. It achieves more than 4.8 times better in terms of area-time product compared to previous results of hardware implementation of NewHope-Simple from Oder and Güneysu at Latincrypt 2017.

Keywords: cryptography, post-quantum cryptography, lattice-based cryptography, LWE, RLWE, key exchange, FPGA implementation

1. INTRODUCTION

In the last decade, post-quantum cryptography has drawn widespread interest. Not only will postquantum cryptography potentially save us from the threats from large quantum computers, but also provide provable security in many cases. Lattice-based cryptography is a candidate for post-quantum cryptography that provides strong theoretical security guarantees such as worst-case to average-case reduction. It also provides the initial constructions of many new cryptographic functionalities, *e.g.*, fully homomorphic encryption [1]. Furthermore, such cryptosystems are usually very efficient. For example, the computation of some public-key encryption based on (Ring-) LWE is faster than RSA/ECC, even though the key size is usually larger [2, 3].

Recently, National Institute of Standards and Technology (NIST) announced a post-quantum crypto project, aiming to select new standard cryptographic primitives for the post-quantum era [4]. The key establishment algorithm is one of the most important primitives in this project. At Usenix 2016, Alkim, Ducas, Pöpplemann, and Schwabe proposed the NewHope post-quantum key agreement scheme, based on the ring-learning-with-errors (RLWE) problem [5]. Google conducted a set of experiments using NewHope on internet

Received September 28, 2019; accepted December 11, 2019.
Communicated by Fu-Hau Hsu.
* This work was supported by the Ministry Of Science and Technology (MOST) for support under Grants No. MOST 1082218-E-002-045 and 108-2221-E-002-066.

through the Google Chrome Canary Browser starting from July, 2016. The results show that NewHope is computationally inexpensive, with only a slight increase in latency for some slow internet connections.

In the era of heterogeneous computing, special purpose computing device can be accessed by the CPU to offload the computation to achieve lower cost or higher power efficiency. However, application specific integrated circuits (ASICs) are very expensive, so a more cost-effective way to deploy hardware accelerators is to use Field-Programmable Gate Arrays (FPGAs).

As a result, FPGAs are one of the most popular ways today to deploy hardware accelerators. An FPGA contains an array of programmable logic components and a hierarchy of reconfigurable interconnects. In fact, people can now launch instances with FPGAs attached on Amazon Web Service (AWS). Therefore, many foresee the use of cloud services with on-demand FPGAs to increase computation resource when load is high.

Our contribution: In this work, we implement the NewHope key exchange protocol which achieves:

- 4.8 times better than the best previous results in terms of time-area product,
- the first work adapts Longa-Naehrig modular reduction on hardware implementation,
- and the pipeline framework is finely configured.

In detail, our implementation requires 6,680 slices, 9,412 FFs, 18,756 LUTs, 8 DSPs and 14 BRAMs on Xilinx Zynq-7000 equipped with 28mm Artix-7 7020 FPGA, and, in our hardware design of NewHope key exchange, the three phases of key exchange costs 51.9, 78.6 and 21.1μs, respectively. Based on our implementation and analysis on the performance bottleneck, our results can also apply to other RLWEbased cryptography, as the new standards are still evolving.

2. BACKGROUND

2.0.1 Notation

Let χ be a probability distribution over a set S. We use $x \xleftarrow{\$} \chi(S)$ to denote x sampled from S according to χ, and $x \xleftarrow{\$} S$ to denote x uniformly sampled from S. We define ring $\mathscr{R} = \mathbb{Z}[X]/(X^n + 1)$ as the ring of integer polynomials modulo $X^n + 1$ and $\mathscr{R}_q = \mathscr{R}/q\mathscr{R}$ as the ring of $\mathscr{R}$, where each coefficient is reduced modulo q.

2.0.2 LWE and RLWE

The LWE problem is first introduced by Regev [6] and can be quantum-reduced to certain worst-case lattice problems. Moreover, Peikert [7] and Brakerski *et al.* [8] further improve the situation by providing classical reductions to lattice problems. LWE based public key cryptosystem is proposed in various variant schemes [7, 9-13]. One important variant of LWE is Ring-LWE, which introduces ring structure into play [14]. The RLWE problem is defined as following: let $s \in \mathscr{R}_q$ be the secret, generate $a \xleftarrow{\$} \mathscr{R}_q$ and $e \xleftarrow{\$} \chi(\mathscr{R}_q)$, compute $b = a * s + e$, and the search version of RLWE is to find s given a list of (a, b).

For the setting of most cryptosystems, only one pair of (a, b) is given.

2.0.3 Post-quantum key exchange

Recently there are two majority ways to construct a post-quantum key exchange: lattice-based and isogeny-based. Super-singular isogeny Diffie-Hellman key exchange is the key exchange scheme based on isogeny [15]. However, the best hardware implementation of SIDH to date has the running time which is typically an order of magnitude larger than similar schemes based on (R-)LWE. Thus, RLWE may still be the most efficient choice of post-quantum key exchange scheme so far.

The first LWE-based key exchange is proposed by Ding [16], subsequently modified by Peikert [17]. At Bos *et al.* 2015, implemented Peikert's version of RLWE key exchange with a parameter set of their choice [18]. They also integrated their implementation into the TLS protocol into OpenSSL.

NewHope is the key exchange scheme proposed by Alkim *et al.* [5], which further improves the performance from [18] by choosing a different set of parameters. Their analysis shows that the new scheme still remains secure while using a smaller modulus, efficient noise sampling, and fast reconciliation. The details of NewHope will be introduced in next section. Frodo is the key exchange scheme based on LWE problem instead of RLWE problem, proposed by Bos *et al.* after NewHope [19]. Without the additional assumption of ring-structure, they selected the parameter with a smaller security margin. Because it based on LWE rather than RLWE, Frodo is still less efficient than NewHope. More precisely, the computation cost of Frodo is around ten times larger, and the communication size is around six times larger than NewHope. However, Frodo is an alternative for post-quantum key exchange since RLWE-based cryptosystem might be potentially insecure [20, 21] due to the ring structure.

2.1 NewHope Protocol

As mentioned earlier, NewHope is a variant of Ding's and Peikert's protocols [16, 17]. The protocol is described in Protocol 1. All the variables except for $\boldsymbol{r} \in \mathbb{R}^4$ are in the ring $\mathscr{R}_q = \mathbb{Z}[X]/(X^n + 1)$, where $n = 1024$ and $q = 12289$. This parameter setting is suitable for a number-theoretic transform (NTT) since $q \equiv 1 \bmod 2n$.

The key idea of the protocol is: Use the property of $\boldsymbol{ass'} + \boldsymbol{es'} = \boldsymbol{bs'} \approx \boldsymbol{us} = \boldsymbol{ass'} + \boldsymbol{e's}$, where Alice can compute the left-hand side part and Bob can compute the right-hand side part. A problem arises in this situation: The codeword is decided by ass', so the rounding technique usually used in an LWE-based cryptosystem does not work. More precisely, the value of ass' may be near the boundary between where a point rounds to 0 and where it rounds to 1. Then Alice and Bob will add different noise vectors, which may lead to different rounding results. The technique to solve this problem is called *reconciliation*. The main idea is that one party (in NewHope, Bob) sends a hint to the other party (in NewHope, Alice), and the two parties can use the hint to decode the message into the same shared secret. The algorithm to generate hint is shown in Algorithm 1, and the reconciliation algorithm is shown in Algorithm 2.

Finally, to transmit a 256-bits key with 1024 coefficients, NewHope encodes 1 bit of codeword into 4 coefficients in order to increase the error resilience and better security.

Table 1. NewHope key exchange scheme.

Parameters: $q = 12289 < 2^{14}, n = 1024$ Error Distribution: ψ_{16}		
Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\boldsymbol{a} \leftarrow$ Parse(SHAKE-128($seed$))		
$\boldsymbol{s}, \boldsymbol{e} \xleftarrow{\$} \psi_{16}$		$\boldsymbol{s}', \boldsymbol{e}', \boldsymbol{e}'' \xleftarrow{\$} \psi_{16}$
$\boldsymbol{b} \leftarrow \boldsymbol{as} + \boldsymbol{e}$	$\xrightarrow{(\boldsymbol{b}, seed)}$	$\boldsymbol{a} \leftarrow$ Parse(SHAKE-128($seed$)) $\boldsymbol{u} \leftarrow \boldsymbol{as}' + \boldsymbol{e}'$ $\boldsymbol{v} \leftarrow \boldsymbol{bs}' + \boldsymbol{e}''$
$\boldsymbol{v}' \leftarrow \boldsymbol{us}$	$\xleftarrow{(\boldsymbol{u}, \boldsymbol{r})}$	$\boldsymbol{r} \xleftarrow{\$}$ HelpRec(v)
$\boldsymbol{\upsilon} \leftarrow$ Rec($\boldsymbol{v}', \boldsymbol{r}$)		$\boldsymbol{\upsilon} \leftarrow$ Rec($\boldsymbol{v}, \boldsymbol{r}$)
$\boldsymbol{\mu} \leftarrow$ SHA3-256($\boldsymbol{\upsilon}$)		$\boldsymbol{\mu} \leftarrow$ SHA3-256(υ)

2.2 Algorithms

2.2.1 Reconciliation

We follow [5] in implementing the reconciliation function. The main idea of the recovery mechanism is to encode and decode over the lattice $\tilde{D}_4$, which is the densest lattice sphere packing in Dimension 4 so that it provides the lowest failure rate. $\tilde{D}_4$ consists the two shifted copies $\mathbb{Z}^4$ with the shift vector $g = (1/2, 1/2, 1/2, 1/2)^t$. The basis of $\tilde{D}_4$ is ($\boldsymbol{e}_1$, $\boldsymbol{e}_2$, $\boldsymbol{e}_3$, $\boldsymbol{g}$).

$$\tilde{D}_4 = \mathbb{Z}^4 \cup (\mathbb{Z}^4 + \boldsymbol{g})$$

Algorithm 1: Help Rec

Require: r-bits reconciliation information, $w \in \mathbb{Z}_q^4$.
Ensure: 4 dimension r-bits reconciliation information, $\{0, 1, \ldots, 2^r - 1\}^4$
1: $b \xleftarrow{\$} \{0, 1\}$
2: $\boldsymbol{x} \leftarrow (\frac{2^r}{q}(\boldsymbol{w} + b \cdot (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^t))$
3: $\boldsymbol{v}_0 \leftarrow \lfloor \boldsymbol{x} \rceil$
4: $\boldsymbol{v}_1 \leftarrow \lfloor \boldsymbol{x} - (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^t \rceil$
5: $k \leftarrow (\|\boldsymbol{x} - \boldsymbol{v}_0\| < 1)\ 0 : 1$
6: $(v_0, v_1, v_2, v_3)^t \leftarrow \boldsymbol{v}_k$
7: return $(v_0, v_1, v_2, v_3)^t + v_3 \cdot (-1, -1, -1, 2)^t \bmod 2^r$

Algorithm 2: Rec

Require: r-bits reconciliation information, $w \in \mathbb{Z}_q^4$, $\boldsymbol{r} = (r_0, r_1, r_2, r_3)$
Ensure: 1-bit shared information
1: $\boldsymbol{x} \leftarrow (\frac{1}{q}\boldsymbol{w} - \frac{1}{2^r} \cdot (r_0 + \frac{r_3}{2}, r_1 + \frac{r_3}{2}, r_2 + \frac{r_3}{2}, \frac{r_3}{2},)^t))$
2: $\boldsymbol{v} \leftarrow \boldsymbol{x} - \lfloor \boldsymbol{x} \rceil$
3: **return** 0 if $\|\boldsymbol{v}\|_1 \le 1$, and 1 otherwise

The encoding method is to equally split the 4-dimensional space by the 1-norm distance to g, that is, the regular 24-cells icositetrachoron shape. The r-bit assisted reconciliation algorithm, the algorithm to generate hints, is shown in Algorithm 1, the reconciliation algorithm is shown in Algorithm 2, and NewHope selects the parameter $r = 2$.

2.2.2 Number-theoretic transform

Direct multiplication (using school-book algorithm) between two elements in polynomial ring costs n^2 multiplications and roughly as many additions or subtractions. The best way to accelerate the computation is to use fast Fourier transform. The number theoretic transform (NTT) is a discrete version of fast Fourier transform defined over a finite ring $\mathbb{Z}_p$. The NTT algorithm is shown in Algorithm 3, and the inverse number theoretic transform, INTT is very similar to NTT except for an additional final multiplication by n^{-1} for each coefficient of the polynomial.

2.2.3 Negative wrapped convolution

The NewHope uses the anti-cyclic ideal $\mathbb{Z}_q[X]/(X^n + 1)$, which does not lead to a classical cyclic convolution when we multiply two ring elements. We use what is called "negative wrapped convolution" to solve the problem. Negative wrapped convolution is first introduced in [22], and Chen *et al.* implemented the algorithm on FPGA [23]. Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be the negative wrapped convolution of $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ and $b = (b_0, b_1, b_{n-1})$, it is defined by

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}.$$

This is exactly the polynomial multiplication over $\mathbb{Z}_q[X]/(X^n + 1)$. Using the NTT multiplication with negative wrapped convolution, the complexity of multiplication over the polynomial ring $\mathbb{Z}_q[X]/(X^n + 1)$ becomes $O(n\log n)$. The pseudo-code of negative wrapped convolution is shown in Algorithm 4.

Algorithm 3: Number-Theoretic Transform, NTT

Require: $\boldsymbol{a} \in \mathbb{Z}_q[X]/(X^n+1)$, ω is a primitive nth root of unity in $\mathbb{Z}_q[X]$, n and q
Ensure: $\boldsymbol{A} = NTT_{\omega}^{n}(\boldsymbol{a})$
  $\boldsymbol{a} \leftarrow Order\_revers(\boldsymbol{a})$
  for $i = 0$ to $\log_2 n - 1$ **do**
    for $j = 0$ to $n/2 - 1$ **do**

$$P_{ij} \leftarrow \left\lfloor \frac{j}{2^{\log_2 n - 1 - i}} \right\rfloor \times 2^{\log_2 n - 1 - i}$$

$$A_j \leftarrow a_{2j} + a_{2j+1}\omega^{P_{ij}} \bmod q$$

    end for
    if $i \neq \log_2 n - 1$ then
      $\boldsymbol{a} \leftarrow \boldsymbol{A}$
    end if

end for
return A

Algorithm 4: Polynomial Multiplication using NTT over $\mathbb{Z}_q[X]/(X^n+1)$

Require: $\boldsymbol{a} \in \mathbb{Z}_q[X]/(X^n+1)$, ω is a primitive nth root of unity in $\mathbb{Z}_q[X]$, $\phi^2 = \omega$, n, and q
Ensure: $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b} \in \mathbb{Z}_q[X]/(X^n+1)$
Precompute: ω^i, ω^{-i}, ϕ^i, ϕ^{-i}, where $i = 0, 1, \ldots, n-1$
for $i = 0$ to $n-1$ **do**
$\bar{a}_i \leftarrow a_i \phi^i \bmod q$
$\bar{b}_i \leftarrow b_i \phi^i \bmod q$
end for
$\bar{\boldsymbol{A}} \leftarrow NTT_{\omega}^{n}(\bar{\boldsymbol{a}})$
$\bar{\boldsymbol{B}} \leftarrow NTT_{\omega}^{n}(\bar{\boldsymbol{b}})$
for $i = 0$ to $n-1$ **do**
$\bar{C}_i \leftarrow \bar{A}_i \bar{B}_i \bmod q$
end for
$\bar{\boldsymbol{c}} \leftarrow NTT_{\omega}^{-n}(\bar{\boldsymbol{C}})$
for $i = 0$ to $n-1$ **do**
$c_i \leftarrow \bar{\boldsymbol{c}}\, \phi^{-i} \bmod q$
end for
return c

2.2.4 Noise sampling

The Knuth-Yao algorithm [24] is a common way to sample high-precision discrete Gaussian distribution, which is implemented in [25]. However, such near optimality may result in non-constant execution time, which might lead to side-channel attack. Thus, we do not use the algorithm in this work. NewHope samples the noise from the binomial distribution instead of discrete Gaussian distribution, which needs high precision and much more computation resources. Moreover, sampling from the centered binomial distribution ψ_{16} is cheap in both hardware and software. One can use the property that the centered binomial distribution follows $\sum_{i=0}^{15} b_i - b_i'$, where the b_i, b_i' are random bits. Thus, the sampling algorithm needs 32 random bits to generate one coefficient.

2.3 FPGA

The basic building block of FPGAs is the look-up tables (LUTs). In Xilinx 7 series FPGA, each LUT can be programmed either as a 6-input 1-output function or two 5-input 1-output functions. To implement sequential circuits, each LUT can be connected to two flip-flops. Certain number of LUTs are then grouped into a slice, and a few slices are grouped into a configurable logic block (CLB). Building around CLBs, FPGAs have other circuitries for, *e.g.*, multiplexing input and output, carry-propagation chains for accelerating arithmetic computation, as well as routing fabrics for connecting LUTs. Furthermore, FPGAs also have fixed multipliers in so-called "DSP slices" that can carry out (fixed-point)

arithmetic operations, as well as block RAM as the fast on-die working memory. We use Xilinx Zynq-7000 all programmable SoC (AP SoC), which is equipped with a dual-core ARM Cortex-A9 processors running at 667 MHz and integrated with 28nm Artix-7 Z-7020 FPGA. This FPGA has 46,200 look-up tables and 220 DSP slices.

3. IMPLEMENTATION

The block diagram is in Fig. 1. There are three main blocks in the diagram representing the flowchart of our hardware implementation of NewHope. First, Alice (Server) uses the TRNG and PRNG to generate the seed of $\hat{a}$ and $b = as + e$ in NTT domain. Bob (Client) receives the seed of $\hat{a}$ and $\hat{b}$ (b in NTT domain), computes $u = as' + e'$ in NTT domain and the his shared secrete $v = bs' + e''$, and compute the reconciliation information and the shared key. In the last step, Alice (Server) receives $\hat{u}$ (u in NTT domain) and reconciliation information r, compute their shared secret $v = us$, and derive shared key though the reconciliation function with r. We explain the techniques in our implementation.

3.1 Random Number Generator

There are two phases in generating the randomness: TRNG (true random number generator) and PRNG (pseudorandom number generator). In the TRNG phase, we use a credible way from Wold and Tan's work to generate the randomness by oscillator rings, which has passed NIST and DIEHARD statistical tests [26]. The throughput of the implementation from Wold and Tan is 100Mbps with less than 100 logic elements in an Altera Cyclone II FPGA. In our implementation, we use 32 oscillators rings to generate the randomness, and their experiment showed if the number of oscillator rings exceeds 25, the result can pass the statistical tests. In the PRNG phase, NewHope uses SHAKE128 as the PRNG, which is the Extendable Output Functions (XOF's) of SHA-3 family. NewHope uses the extendable property to generate 1024 uniform coefficients in $\mathbb{Z}_p$ with 256 bits true randomness since the randomness is sufficiently to resist either classic brute-force attack or quantum attack (Grover's algorithm). We extract the SHAKE128 portion from open-source code [27], which usually provides only standard SHA-3 on FPGA.

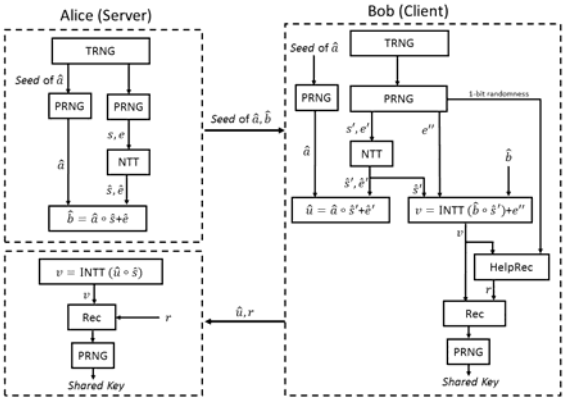


Fig. 1. Flowchart of our implementation.

3.2 Number-Theoretical Transform

We use the design of optimized NTT hardware implementation in [23, 28]. The main differences are that we use 4 butterfly units, and the modulus is different.

Fig. 2 is the high level design of our NTT implementation, it combined both NTT and INTT. For NTT, it processes multiplication on ϕ^i, order reverse, and butterfly units in order. In contrast, INTT processes order reverse, butterfly units and multiplication on ϕ^i in order.

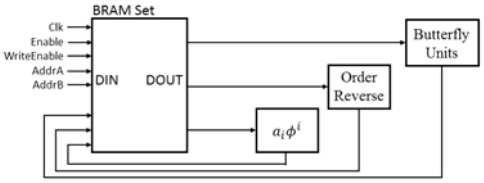


Fig. 2. Overview of our NTT implementation, which consists three components of circuit: multiplication on ϕ^i, order reverse, and butterfly units.

3.2.1 Order-reverse

Unlike software implementation, the order-reverse part, whose latency is shown in Table 2, is one of the bottleneck of NTT in hardware implementation. We point out that this part can be ignored since we can assume that either the input generated from random number generator is ordered as the input of the butterfly units in NTT or it is not necessary to reverse the order in INTT by both of two parties. But, both parties must agree to do or not to do the order-reverse process in order to reconcile the same shared-key. Thus, we can remove this part in order to accelerate NTT around 40%.

3.2.2 Butterfly units

In [23], they use 8 and 2 butterfly units and compare the performance. In [28]'s implementation, they use a single butterfly unit to compute the NTT function in order to optimize the area usage. We use 4 butterfly units to compute the NTT since our implementtation aims to be more speed-optimized. Also, we follow the idea of [23] we use the architecture shown in Fig. 3 that places the data into the memory in the correct positive in order to achieve higher efficiency.

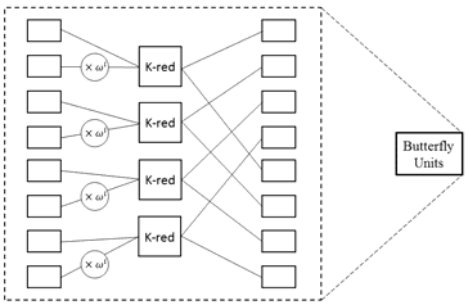


Fig. 3. Illustration of the design of the butterfly unit.

3.2.3 Modular reduction

A common way to do modular reduction is Barrett reduction.

$$c \bmod p = c - \left\lfloor (c \cdot \tfrac{1\ll 32}{12289}) \gg 32 \right\rfloor \cdot 12289$$

In this viewpoint, we can use DSP to multiply the reciprocal of 12289 without computing the floating number. Since the algorithm chops rather than rounds the result, the result is possibly slightly large than p. Thus, the algorithm subtracts p if it is larger than p in the final step. We can further improve the computation since $12289 = (1\ll 13) + (1\ll 12) + 1$ by following equation, where $\bar{a}$ is the complement of $\left\lfloor (c \cdot \tfrac{1\ll 32}{12289}) \gg 32 \right\rfloor$

$$c \bmod p = (c + (\bar{a} \ll 13)) + ((\bar{a} \ll 12) + \bar{a}).$$

So, a Barrett modular reduction with $q = 12289$ is around 5 cycles. But there is a multiplication between 32 bit and 19-bit numbers leading to a long critical path and limiting the frequency.

Therefore, we opt for the efficient reduction method from [29] for modular reduction. The method is a variant of Montgomery reduction with the auxiliary modulus k, which is defined by $q = k \cdot 2^m + 1$. For $q = 12289$, we have $m = 12$ and $k = 3$.

function K−RED(C)
  $C_0 \leftarrow C \bmod 2^m$
  $C_1 \leftarrow C/2^m$
  return $kC_0 - C_1$
function K−RED−2x(C)
  $C_0 \leftarrow C \bmod 2^m$
  $C_1 \leftarrow C/2^m \bmod 2^m$
  $C_2 \leftarrow C/2^{2m}$
  return $k^2C_0 - kC_1 + C_2$

This algorithm is suitable for hardware implementation, since the operations in the function K−RED and K−RED2x are bit selections plus a final step which is equal to $(C_0\ll 1) + C_0 - C_1$ and $(C_0\ 3) + C_0 - (C_1\ll 1) - C_1 + C_2$, respectively. Using this technique, we replace Line 5&6 in Algorithm 3 and get Algorithm 5.

Algorithm 5: Number-Theoretic Transform with K−RED

Require: $\boldsymbol{a} \in \mathbb{Z}_q[X]/(X^n + 1)$, ω is a primitive nth root of unity in $\mathbb{Z}_q[X]$, n and q
Ensure: $\boldsymbol{A} = NTT^n_\omega(\boldsymbol{a})$
  $\boldsymbol{a} \leftarrow Order\_reverse(\boldsymbol{a})$
  for $i = 0$ to $\log_2 n - 1$ **do**
    for $j = 0$ to $n/2 - 1$ **do**
      $P_{ij} \leftarrow \left\lfloor \frac{j}{2^{\log_2 n - 1 - i}} \right\rfloor \times 2^{\log_2 n - 1 - i}$
      $U \leftarrow$ **K−RED**(a_{2j})
      $V \leftarrow$ **K−RED2x**$(a_{2j+1}\omega^{P_{ij}})$

$A_j \leftarrow U + V$
$A_{j+n/2} \leftarrow U - V$
end for
if $i \neq \log_2 n - 1$ **then**
$\boldsymbol{a} \leftarrow \boldsymbol{A}$
end if
end for
return A

We replace Line 3, 4, 8, 9 and 11 in Algorithm 4 to get Algorithm 6.

Note that K−RED function does not compute the exact value C mod q but kC mod q. Correspondingly K−RED2x function computes k^2C mod q, and we eliminate the extra factor of k by storing $\omega_{ij}^P k^{-1}$ instead of ω_{ij}^P. Thus, after multiplication of $\omega_{ij}^P k^{-1}$ and K−RED2x function, the result kC has the correct value. Since $n = 1024 = 2^{10}$, there are ten stages in NTT function, the output vector from NTT with K−RED is $k^{10}v$, where v is the correct output vector of NTT. It is easy to transform the output vector to correct one, but we wait until the last step of INTT, which now becomes a final multiplication by the pre-computable $n^{-1}k^{-14}$.

One trick in the modified algorithm is to pre-compute $\phi^i k^{-(2+\log n)}$ instead of ϕ^i. This ensures that the output of our modified algorithm exactly is the same as that from the original NTT. We also replace $INTT_n$ by NTT_n, and multiply instead by $n-1$ $\phi - i$ (which can also be pre-computed and stored in the block RAM) in Line 11. This way we only need 1024 multiplications.

Note that the output of both functions are bounded by not a fixed value but by $q + |C|/2^m$ which is related the input value C. Applying results of [29] to our algorithm, the input size of function K−RED and K−RED2x are 16 bits and 32 bits, respectively. One technique to maintain a plus sign for the output of these two functions (in order to multiply using DSP slices in the next stage) is to add multiples of q = 12289. It can be verified that $U + V$ and $U - V$ are larger than $-2q$ and $-4q$, respectively. But directly adding $2q$ and $4q$ to $U + V$ and $U - V$ causes a new problem: it may exceed 16 bits. BRAM reads 64 bit at a time, so 17 bits as the input of K−RED slows each BRAM read to 3 data points.

Thus, we propose the method to solve the problem:

Let s be bit 11 (corresponding to 2048) of $a_{2j+1}\omega^{Pij}$ in Line 6 in Algorithm 5.
If $s = 0$, $A_j \leftarrow U + V + 2q$ and $A_{j+n/2} \leftarrow U - V + 2q$.
If $s = 1$, $A_j \leftarrow U + V$ and $A_{j+n/2} \leftarrow U - V + 4q$.

Algorithm 6: Polynomial Multiplication using NTT with **K−RED** over $\mathbb{Z}_q[X]/(X^n + 1)$

Require: $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q[X]/(X^n + 1)$, ω is a primitive nth root of unity in $\mathbb{Z}_q[X]$, $\phi^2 = \omega$, n, and q
Ensure: $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b} \in \mathbb{Z}_q[X]/(X^n + 1)$
Precompute: ω^i, ω^{-i}, ϕ^i, ϕ^{-i}, where $i = 0, 1, ..., n - 1$
for $i = 0$ to $n - 1$ **do**
$\bar{a}_i \leftarrow$ **K−RED2x**$(a_i(\phi^i k^{-(2+\log n)}))$
$\bar{b}_i \leftarrow$ **K−RED2x**$(b_i(\phi^i k^{-(2+\log n)}))$
end for
$\bar{\mathbf{A}} \leftarrow \mathbf{NTT}_{\omega}^{n}(\bar{\mathbf{a}})$

$\bar{\mathbf{B}} \leftarrow \mathbf{NTT}_{\omega}^{n}(\bar{\mathbf{b}})$
for $i = 0$ **to** $n - 1$ **do**
$\bar{C}_i \leftarrow$ K-RED2x$(\bar{A}_i \bar{B}_i)$
end for
$\bar{c} \leftarrow NTT_{-\omega}^{n}(\bar{C})$
for $i = 0$ to $n - 1$ **do**
$c_i \leftarrow$ K-RED2x$(\bar{c}_i(\phi^i k^{-(4+\log n)} n^{-1}))$
end for
return c

Note that both sets of values are computed and then selected using s to avoid side-channels. This modification makes sure that the results of that step are positive. This method is a consequence of the properties of the K-RED and K-RED2x functions, and we give the proof in Appendix A. Note that the outputs of function K-RED and K-RED2x are signed 14 bits and signed 16 bits, respectively. Combined all the techniques describe above, the design of K-RED in the butterfly unit is shown in Fig. 4.

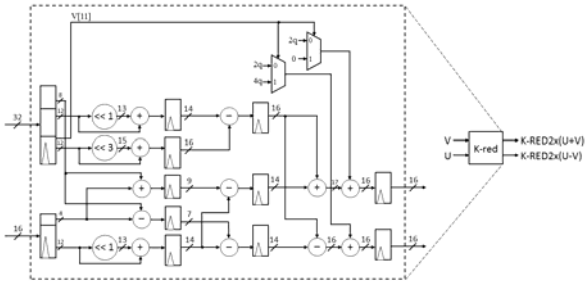


Fig. 4. Illustration of the design of K-RED.

3.2.4 Previous method

According to our knowledge, most of the previous method to achieve modular reduction is long division with pipeline. A nature problem with this method is the stage number of the pipeline is decided by ⌈log(dividend)-log(divisor)⌋. But our method for specific modular number has much less stages (which is 4, long division is 13), which reduces the area.

3.3 Reconciliation

A naive way to implement the HelpRec and Rec function on FPGA is to pre-compute $1/q$ and to use DSPs to compute the multiplication in runtime. This way is inefficient and wastes many logic elements. In our implementation of reconciliation, instead of trying to determine $\sum_{n=0}^{3} x_i/q < 1$ or not, we determine where $\sum_{n=0}^{3} x_i$ is less than q or not, in order to avoid floating-point number computation. Other divisors do not need this trick because they are all powers of 2. We designed 6 stages pipeline architecture for HelpRec modulo and 3 stages pipeline architecture for Rec modulo.

4. RESULTS

The three phases of key exchange cost 51.9, 70.1 and 21.1μs, respectively. The resource consumption of each component is shown in Table 2 and the design of hardware architecture is shown in Fig. 5. The area of PRNG (SHAKE from SHA-3) is quite large among the components. It occupies 44.3% of FFs and 18.7% of LUTs in our implementation. However, it is not the focus of this work. In theory we could have taken any FPGA SHA-3 implementation, such as the area-optimized one from [30] which only uses one tenth of the area. Alternatively, one can use a lightweight PRNG to generate the randomness for ψ_{16}. The implementation of SHAKE outputs 1344 bits per 24 cycles with a few cycles for setting up. To generate the uniform coefficient a, we use reject sampling with 16 bits: if the number is less than $5q$, accept it, otherwise, reject it. Thus, the accept rate is $5 * 12289/2^{16} \sim 93.76\%$ and the expected number of SHAKE is 13. For the binomial random variable ψ_{16}, 32 bits randomness is required to generate one coefficient. Thus, the total latency is around 2 times of the latency of generating $\tilde{a}$. The area of NTT component is reasonable since it is around 4 times that of [28]. Note that we use 4 butterfly units in each NTT component, and they use only one. For the reconciliation, we use 2 copies of `HelpRec/Rec` circuits in order to get high performance. Therefore, the latency of `HelpRec+Rec` and `Rec` in our implementation are 141 and 135 cycles, respectively. Note that, the output of `HelpRec` immediately sends to `Rec` in Bob part. Thus, the latency can be hidden and it is only 6 clocks slower than `Rec` in Alice part.

Table 2. The resource consumption of each component.

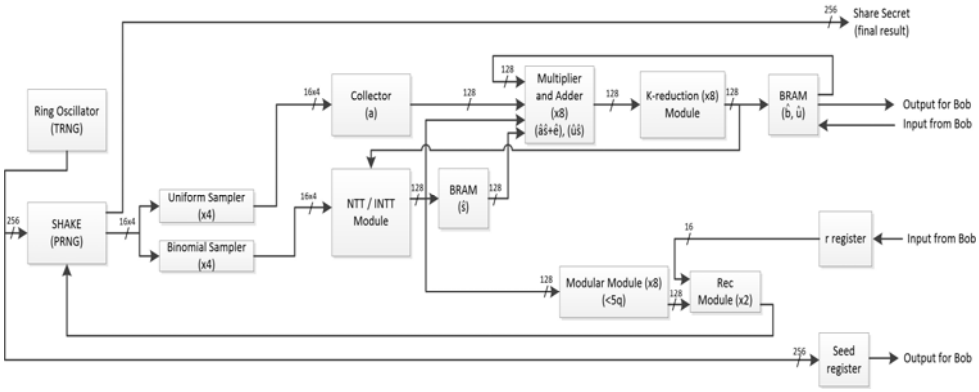
Component	Area				Clock Count	Max Freq. (MHz)
	#LUTs	#Slices Registers	#DSPs	#BRAMs		
TRNG	310	258	0	0	1	–
PRNG (SHA-3)	3,516	2,976	0	0	24	355
–generate a	–	–	–	–	312	–
–generate $\psi 16$	–	–	–	–	613	–
pipelined NTT	2,832	1,381	8	10	2616	150
–multiply ϕ^i	–	–	–	–	132	–
–Order Reverse	–	–	–	–	1024	–
–Butterfly Units	–	–	–	–	1330	–
`HelpRec+Rec` (Bob)	968	406	0	0	269	229
`Rec` (Alice)	557	127	0	0	263	229

To date, our implementation is the fastest post-quantum key exchange, which is 222, 138 and 19.1 times faster than that of SIDH [31-33], respectively.

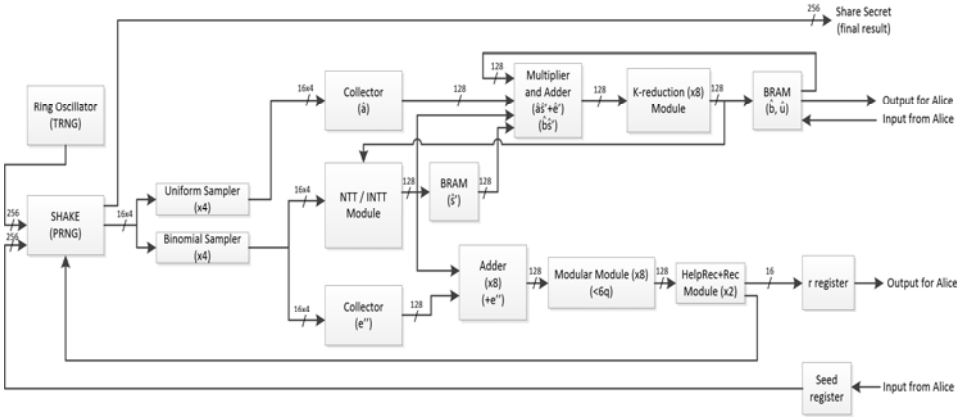
In Table 3, we also show the best record of hardware implementations of lattice-based key exchange and public key encryption (PKE).

Comparing to implementation of NewHope-Simple In [33], their implementation uses 1,483 and 1,708 slices for client and server side, and our implementation uses 6,680 and 7,153 slices. For postquantum key exchange, the resource we use is less than four times larger than NewHope-Simple implementation [33], but the total time is 19.1 times faster.

That is, the time-area product is more than 4.8 times better. The first reason is that we design 4 stages of pipeline in the K−RED modulo and second reason is we adapted the Longa-Naehrig modular reduction to reduce the resource. Also, one can observe that the reconciliation is relatively cheap and would not be the bottleneck of the key exchange scheme.



(a) Alice (Server) side.



(b) Bob (Client) side.

Fig. 5. Our design of hardware architecture.

Comparing to lattice-based PKE At first glance, our results is worse than the hardware implementation of PKE. But the computation of NewHope is about 3.3 times larger than the computation of RLWE with $(p, q, \sigma) = (512, 12289, 4.92)$. The computation of NTTs dominates both schemes (in fact, NewHope has higher load because it has to expand a and compute Rec and HelpRec) Totally, NewHope has 6 NTT parts (include INTT) and RLWE-based PKE typically has 4 NTT parts (include INTT). And considering that the size of the NTT is $n \log n$, the overall computation ratio is at least 3.3. The total time of our implementation is 151.6μs, and the total time of RLWE (512, 12289, 4.92) is 68.9μs. However, the two primitives are different. For a public-key encryption scheme to provide forward secrecy, a one-time public key needs to be generated and transmitted every time before being used. That would probably make up much of the difference.

Table 3. Hardware comparison of post-quantum key exchange and some post-quantum public key encryption. In the column of area and frequency, the slash serves to denote the cost of Alice-modulo and Bob-modulo. In the column of latency and total time, the slash serves to denote the cost of Alice0, Bob, and Alice1, the three phases.

Scheme	Parameters	Security Parameter	Area				Time		
			#FFs	#LUTs	#DSPs	#BRAMs	Freq (MHz)	Latency ($\times 10^3$)	Total time (μs)
SIDH [31]	prime: 511 bits	128 bits	30,031	24,499	192	27	177	5,967	33,700
SIDH [32]	prime: 503 bits	125 bits	26,659	19,882	192	40	181.4	3,800	20,900
NewHope (This Work)	$n = 1024$, $p = 12289$, noise dist. ψ_{16}	128 bits	9,412 /9,975	18,756 /20,826	8/8	14/14	133/131	6.9/10.3 /2.8	51.9/78.6 /21.1
NewHope-Simple [33]	$n = 1024$, $p = 12289$, noise dist. ψ_{16}	128 bits	4,452	5,142	2	4	125/117	171/179	988/1434 /473
RLWE(PKE) [28]	$n = 256$, $q = 7681$, $\sigma = 4.516$	80 bits	860	1,349	1	2	313	6.3/2.8	20.1/9.1
	$n = 512$, $q = 12289$, $\sigma = 4.92$	128 bits	953	1,536	1	3	278	13.3/5.8	47.9/21
RLWE(PKE) [34]	$n = 256$, $q = 7681$, $\sigma = 11.32$	80 bits	3,624	4,549	1	12	262	7.24/6.86/4.40	27.6/26.19/16.8
	$n = 512$, $q = 12289$, $\sigma = 12.18$	128 bits	4,760	5,595	1	14	251	14.5/13.8/8.8	57.9/54.9/35.4
LWE (PKE) [35]	$n = 256$, $q = 4096$, $\sigma = 3.39$	128 bits	4,804	6,152	1	73	125	98.3/32.8	786/262
NT RU [36] *ees*761*ep*1	$n = 761$, $q = 2048$, $p = 3$	128 bits	#logic elm: 42,642, #reg: 16,746				75.36	0.44	5.89

However, as we mentioned in the introduction, the functionality of key transport is not the same as key agreement. Therefore, there is a need for a post-quantum key exchange scheme as well as its hardware implementation.

Comparing to software implementation In [5], they implemented two versions of NewHope: a C reference implementation and an optimized AVX2 implementation. The three phases costs 73.95, 110.25, 24.71 and 25.46, 31.78, 5,56 μs, respectively. Our implementation is 1.38 times faster than a C reference implementation. The optimized AVX2 version is 2.41 times faster than our implementation. But an Intel Core i7-4770K (the CPU they used) costs 350 USD, compared to a Xilinx Zynq-7000 costs only 100 USD. Thus, the FPGA is more cost-effective than software implementation. Note that the price of all CPUs equipped with AVX2 are much higher than 100 USD so far.

5. CONCLUSION

In this work, we propose a high performance hardware implementation of lattice-based key exchange, which is also the fastest hardware implementation of post-quantum key exchange so far. Compare to the previous NewHope-Simple hardware implementation, our implementation did 4.8× better in time-area product. This is the first pipeline implementation of lattice-based key exchange, and is the first work to adapt Longa-Naehrig modular reduction into hardware design. We also show the cost of reconciliation, which is quite cheap. Our code will be made public available.

5.1 Future Work

A countermeasure for side channel attacks (SCA) is an urgent priority. For example,

we may use a method such as the masked RLWE decryption implementation resistant to first-order SCA is proposed in [37] and apply it in our implementation. It is also interesting to optimize the SCA countermeasures for post-quantum key exchange scheme.

REFERENCES

1. C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of Annual ACM Symposium on Theory of Computing*, 2009, pp. 169-178.
2. N. Göttert, T. Feller, M. Schneider, J. A. Buchmann, and S. A. Huss, "On the design of hardware building blocks for modern lattice-based encryption schemes," in *Proceedings of the 14th International Workshop on Cryptographic Hardware and Embedded Systems*, 2012, pp. 512-529.
3. H. M. Fadhil and M. I. Younis, "Article: Parallelizing rsa algorithm on multicore cpu and gpu," *International Journal of Computer Applications*, Vol. 87, 2014, pp. 15-22.
4. N. I. of Standards and T. (NIST), "Post-quantum crypto project," http://csrc.nist.gov/groups/ST/post-quantum-crypto/, 2016.
5. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange a new hope," in *Proceedings of the 25th USENIX Security Symposium*, 2016, pp. 327-343.
6. O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of ACM*, Vol. 56, 2009, pp. 1-40.
7. C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem: extended abstract," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, 2009, pp. 333-342.
8. Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé, "Classical hardness of learning with errors," in *Proceedings of Symposium on Theory of Computing Conference*, 2013, pp. 575-584.
9. S. Agrawal, D. Boneh, and X. Boyen, "Lattice basis delegation in fixed dimension and shorterciphertext hierarchical IBE," in *Proceedings of the 30th Annual Cryptology Conference*, 2010, pp. 98-115.
10. S. Agrawal, D. Boneh, and X. Boyen, "Efficient lattice (H)IBE in the standard model," in *Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010, pp. 553-572.
11. Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Proceedings of the 31st Annual Cryptology Conference*, 2011, pp. 505-524.
12. Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *Electronic Colloquium on Computational Complexity*, Vol. 18, 2011, p. 109.
13. R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *Proceedings of Cryptographers' Track at the RSA Conference*, 2011, pp. 319-339.
14. V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proceedings of EUROCRYPT*, 2010, pp. 1-23.
15. C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for super singular isogeny diffiehellman," in *Proceedings of the 36th Annual International Cryptology Conference*, 2016, pp. 572-601.

16. J. Ding, "A simple provably secure key exchange scheme based on the learning with errors problem," *IACR Cryptology ePrint Archive*, Vol. 2012, 2012, p. 688.
17. C. Peikert, "Lattice cryptography for the internet," in *Proceedings of the 6th International Workshop on Post-Quantum Cryptography*, 2014, pp. 197-219.
18. J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, "Post-quantum key exchange for the TLS protocol from the ring learning with errors problem," in *Proceedings of IEEE Symposium on Security and Privacy*, 2015, pp. 553-570.
19. J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, "Frodo: Take off the ring! practical, quantum-secure key exchange from LWE," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1006-1018.
20. Y. Elias, K. E. Lauter, E. Ozman, and K. E. Stange, "Provably weak instances of ring-lwe," in *Proceedings of the 35th Annual Cryptology Conference*, 2015, pp. 63-92.
21. W. Castryck, I. Iliashenko, and F. Vercauteren, "Provably weak instances of ring-lwe revisited," in *Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016, pp. 147-167.
22. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," in *Proceedings of the 15th International Workshop on Fast Software Encryption*, 2008, pp. 54-72.
23. D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede, "High-speed polynomial multiplication architecture for ring-lwe and SHE cryptosystems," *IEEE Transactions on Circuits and Systems*, Vol. 62-I, 2015, pp. 157-166.
24. D. Knuth and A. Yao, "The complexity of nonuniform random number generation," *Algorithms and Complexity: New Directions and Recent Results*, Academic Press, NY, 2006, pp. 156-169.
25. S. S. Roy, F. Vercauteren, and I. Verbauwhede, "High precision discrete gaussian sampling on fpgas," in *Proceedings of the 20th International Conference on Selected Areas in Cryptography*, 2013, pp. 383-401.
26. K. Wold and C. H. Tan, "Analysis and enhancement of random number generator in FPGA based on oscillator rings," *International Journal of Reconfigurable Computing*, Vol. 2009, 2009, pp. 501 672:1-501 672:8.
27. OpenCores, "SHA3(KECCAK)," https://opencores.org/project,sha3, 2012.
28. S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-lwe crypto processor," in *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems*, 2014, pp. 371-391.
29. P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Proceedings of the 15th International Conference on Cryptology and Network Security*, 2016, pp. 124-139.
30. S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. M. de Dormale, and F. Standaert, "Compact FPGA implementations of the five SHA-3 finalists," in *Proceedings of the 10th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications*, 2011, pp. 217-233.
31. B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Post-quantum cryptography on FPGA based on isogenies on elliptic curves," *IEEE Transactions on Circuits and Systems*, Vol. 64-I, 2017, pp. 86-99.

32. M. M. K. B. Koziel, and R. Azarderakhsh, "Fast hardware architectures for super singular isogeny Diffie-Hellman key exchange on FPGA," *Cryptology ePrint Archive*, Report 2016/1044, 2016.
33. T. Oder and T. Güneysu, "Implementing the newhope-simplekey exchange on low-cost fpgas," in in *Proceedings of the 6th International Conference on Cryptology and Information Security*, 2017, pp. 128-142.
34. T. Pöppelmann and T. Güneysu, "Towards practical lattice-based public-key encryption on reconfigurable hardware," in *Proceedings of SAC 20th International Conference*, 2013, pp. 68-85.
35. J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and K. Beeden, "Standard lattices in hardware," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 162:1-162:6.
36. B. Liu and H. Wu, "Efficient multiplication architecture over truncated polynomial ring for ntruencrypt system," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2016, pp. 1174-1177.
37. O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "A masked ring-lwe implementation," in *Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems*, 2015, pp. 683-702.

APPENDIX A: PROOF OF THE INPUT SIZE AND OUTPUT SIZE OF K−RED BUTTERFLY

Here we consider the case of $q = 12289$, $k = 3$. The input U, V satisfies the following condition.

$$V = V_0 + V_1 \cdot 2^{12} + V_2 \cdot 2^{24} \text{ and } U = U_0 + U_1 \cdot 2^{12}$$

where $0 \le V_0 < 2^{12}$, $0 \le V_1 < 2^{12}$, $0 \le V_2 < 2^6$, $0 \le U_0 < 2^{12}$, and $0 \le U_1 < 2^4$

$$A = \text{K−RED}(U) = 3U_0 - U_1 \text{ and } B = \text{K−RED2x}(V) = 9V_0 - 3V_1 + V_2.$$

Thus, the output of the butterfly unit is:

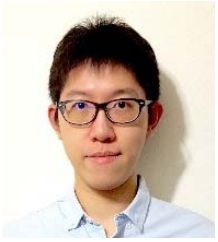
$$A + B = 9V_0 + 3U_0 + V_2 - (3V_1 + U_1) \text{ and } A - B = 3(U_0 + V_1) - (9V_0 + V_2 + U_1)$$

Define $s = (V_0 \gg 11) \bmod 2$; let's first consider $s = 0$, *i.e.* $V_0 < 2^{11}$,
$9V_0 + 3U_0 + V_2 \ge A + B \ge -(3V_1 + U_1)$
$9 \cdot 2^{11} + 3 \cdot 2^{12} + 2^{12} \ge A + B \ge -(3 \cdot 2^{12} + 2^{12})$
$2^{16} > A + B + 2q > 0$
$A - B \ge -(9V_0 + V_2 + U_1) \ge -(9 \cdot 2^{11} + 2^6 + 2^4) > -2q$
$A - B \le 3(U_0 + V_1) < 3 \cdot (2^{11} + 2^{11}) < 2^{16} - 2q$
$2^{16} > A - B + 2q \ge 0$

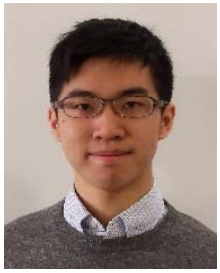
Thus, we prove that when $s = 0$, adding $2q$ always make the output between 0 and 2^{16}:

When $s = 1$, *i.e.*, $2^{11} \le V_0 < 2^{12}$,
$9V_0 + 3U_0 + V_2 \ge A + B \ge 9V_0 - (3V_1 + U_1)$
$9 \cdot 2^{12} + 3 \cdot 2^{12} + 2^6 > A + B > 9 \cdot 2^{11} - 2^{14}$
$2^{16} > A + B > 0$
$A - B \ge -(9V_0 + V_2 + U_1) \ge -(9 \cdot 2^{12} + 2^{12} + 2^{12}) > -4q$
$A - B \le 3(U_0 + V_1) - 9 \cdot 2^{11} < 3(2^{12} + 2^{12}) - 9 \cdot 2^{11} < 2^{16} - 4q$
$2^{16} > A - B + 4q \ge 0$
We prove that adding $4q$ to $B - A$ makes the output between 0 and 2^{16}.

Po-Chun Kuo received the BS, MS, and Ph.D. degrees in Electrical Engineering from National Taiwan University in 2010, 2011 and 2020, respectively. His research interests include Post-Quantum Cryptography, high-performance computing, and graph theory. Currently, his main research activities focus on the blockchain field such as consensus algorithm, mining mechanism, and decentralized cryptographic protocol.



Wen-Ding Li received his BS and MS degrees in Electrical Engineering from National Taiwan University. He is now a Research Assistant at Research Center for Information Technology Innovation, Academia Sinica, Taiwan. His research interests include high performance computing, symbolic computation and cryptography.



Yu-Wei Chen received his BS degree in Computer Science from National Chiao Tung University in 2016. He later enrolled in the master's program in Electrical Engineering at National Taiwan University, under the supervision of Prof. Chen-Mou Cheng. His research interest includes high-performance computing, cryptography, embedded system. Focusing on topics that combine both cryptography and low latency FPGA implementations.

Yuan-Che Hsu received his BS and MS degrees in Electrical Engineering from National Taiwan University. He is now working on AI Processing Unit for smartphones and Hardware Security Module for automobiles. His research interests include efficient implementations for cryptographic hardware and side-channel analysis.

Chen-Mou Cheng received his BS and MS in Electrical Engineering from National Taiwan University in 1996 and 1998, respectively, and his Ph.D. in Computer Science from Harvard University in 2007. He joined the Department of Electrical Engineering of National Taiwan University in 2007, where he is currently an Associate Professor. His main research area is in cryptographic hardware and embedded systems (CHES), as well as electronic system-level (ESL) design. Currently, his main research activities focus on the design and analysis of efficient algorithms to solve several important problems arising from cryptology, as well as the development and implementation of these algorithms on massively parallel computers. These problems include solving systems of polynomial equations over finite fields, integer factorization, elliptic-curve discrete logarithm, and lattice reduction.

Bo-Yin Yang received his Ph.D. in Mathematics from Massachusetts Institute of Technology in 1991. After teaching mathematics at Tamkang University in Taiwan, he started working with cryptography in 2002. Eventually moved to the Institute of Information Science at Academia Sinica in 2006. He is known for his work on efficient crypto implementations, algebraic cryptanalysis, and post-quantum public-key cryptography.