Implementing the NewHope-Simple Key Exchange on Low-Cost FPGAs

Tobias Oder[1] and Tim Güneysu[1,2⋆]

[1] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
[2] DFKI, Germany
{tobias.oder,tim.gueneysu}@rub.de

Abstract. Lattice-based cryptography is one of the most promising candidates being considered to replace current public-key systems in the era of quantum computing. In 2016 Alkim, Ducas, Pöppelmann, and Schwabe proposed the lattice-based key exchange scheme `NewHope`. The scheme has gained some popularity in the research community as it is believed to withstand attacks by quantum computers with a comfortable security margin and provides decent efficiency and low communication cost. In this work, we evaluate the efficiency of `NewHope` on reconfigurable hardware. We provide the up to our knowledge first field-programmable gate array (FPGA) implementation of `NewHope-Simple` that is a slight modification of `NewHope` proposed by the authors themselves in 2016. `NewHope-Simple` is basically `NewHope` with different error correction mechanism. Our implementation of the client-side scheme requires 1,483 slices, 4,498 look-up tables (LUTs), and 4,635 flip-flops (FFs) on low-cost Xilinx Artix-7 FPGAs. The implementation of the server-side scheme takes 1,708 slices, 5,142 LUTs, and 4,452 FFs. Both cores use only two digital signal processors (DSPs) and four 18-kb block memories (BRAMs). The implementation has a constant execution time to prevent timing attacks. The server-side operations take 1.4 milliseconds and the client-side operations take 1.5 milliseconds.

Keywords: Ideal lattices, NewHope, FPGA

1 Introduction

Public-key cryptography provides important security services to protect information sent over untrusted channels. Unfortunately, most well-established public-key cryptographic primitives rely either on the factorization or the discrete logarithm problem. As both problems are closely connected, a mathematical breakthrough in one of the problems would render primitives based on either of the problems insecure. In this context, the possible advent of the quantum computer is another crucial threat. A significant number of experts believe that

⋆ This work was partially funded by the European Union H2020 SAFEcrypto project (grant no. 644729), European Union H2020 PQCRYPTO project (grant no. 645622)

quantum computers that are large enough to pose a threat to cryptographic schemes are built within the next decade [6,22]. The National Institute of Standards and Technology (NIST) recently published a call for proposals [25] that asks to submit public-key encryption, key exchange, or digital signature schemes for standardization.

Lattice-based cryptography is a family of primitives that is believed to be secure against attacks by quantum computers. It has efficient instantiations for all three types of most relevant security services and provides reasonable parameter sizes for a decent level of security. Especially the `NewHope` key exchange by Alkim et al. [2] has gained significant attention from the research community and the industry. Google even tested the scheme in its Chrome browser [9]. While the original `NewHope` proposal contains a tricky error reconciliation, Alkim et al. proposed an improved version called `NewHope-Simple` [1] that avoids this error reconciliation at the price of increasing the size of the message that is sent from the client to the server from 2048 bytes to 2176 bytes.

In this work we present an implementation of `NewHope-Simple` for Field-Programmable Gate Arrays (FPGAs). FPGAs are widely used as platform for cryptographic hardware (e.g., also in the Internet of Things) and thus a highly interesting platform for the evaluation of `NewHope-Simple`. Our target platform is a low-cost Xilinx Artix-7 FPGA, but we expect similar implementation results on other reprogrammable hardware devices.

1.1 Related Work

Alkim et al. evaluated the performance of `NewHope` on Intel CPUs. They utilize the SIMD instructions of the AVX2 instructions set to achieve a high performance. Another implementation of `NewHope` targets ARM Cortex-M processors [4]. In both works [2,4] the authors implemented the original `NewHope` scheme and not `NewHope-Simple`. We are not aware of any hardware implementations of `NewHope-Simple`.

Besides `NewHope`, there is also a lattice-based key exchange called `Frodo` [7]. In contrast to `NewHope`, Frodo is based on standard lattices instead of ideal lattices. The difference between both types of lattices is that ideal lattices include a fundamental structure and thus allow a more efficient instantiation. It is, however, unclear whether this additional structure can be exploited by an attacker. So far no attacks that exploit the structure of ideal lattices and have a better runtime than the best known lattice attacks are known. Due to the higher memory consumption, `Frodo` is less suited for implementation on low-cost hardware. Another lattice-based key exchange has been developed by Del Pino et al. [10]. They present a generic approach and their scheme can be instantiated with any suitable signature scheme and public-key encryption scheme. Note that the scheme of [10] is an authenticated key exchange while `NewHope` and `Frodo` are unauthenticated and thus require an additional signature scheme for the authentication part.

While we are not aware of any hardware implementations of `NewHope`, the ring-learning with errors encryption scheme (`ring-LWE`) has been implemented in

works like [19,20,24]. Furthermore, there is an implementation of a lattice-based identity-based encryption scheme (IBE) for Xilinx FPGAs [14]. IBE, ring-LWE, and NewHope-Simple share most operations like the number theoretic transform and the Gaussian sampling. Additionally to the operations required by ring-LWE and IBE, NewHope-Simple also requires the on-the-fly generation of the polynomial $\mathbf{a}$ (usually precomputed in the implementations of ring-LWE), SHAKE-128, SHA3-256, and a compression function.

1.2 Contribution

NewHope has been first proposed in late 2015 [3]. But there are still no hardware implementations of the scheme published. In this work we aim to close this gap. We present the up to our knowledge first implementation of NewHope-Simple for reconfigurable hardware devices. We optimized our implementation for area while taking care to still achieving a decent performance. Our work shows that NewHope-Simple is practical on constrained reconfigurable hardware. Our implementations takes 1,483 slices, 4,498 LUTs, and 4,635 FFs for the client and 1,708 slices, 5,142 LUTs, and 4,452 FFs for the server. Both cores use only 2 DSPs and four 18-kb block memories. It has a constant execution time to prevent timing attacks and hamper simple power analysis. We achieved a performance of 350,416 cycles at a frequency of 117 MHz for the entire protocol run. We will also provide the source code with the publication of our work [3].

2 Preliminaries

In this chapter, we discuss the mathematical background that is crucial for the understanding of this paper.

2.1 Notation

Let $\mathbb{Z}$ be the ring of rational integers. We denote by $\mathcal{R}$ the polynomial ring $\mathbb{Z}[x]_q/\langle x^n+1\rangle$ where n is a power of two and x^n+1 is the modulus. The coefficients of the polynomials have the modulus q. In case χ is a probability distribution over $\mathcal{R}$, then $x \xleftarrow{\$} \chi$ means the sampling of x according to χ. The point-wise multiplication of two polynomials is denoted by the operator $\circ$. Polynomials in the time domain are denoted by bold lower case letters (e.g. $\mathbf{a}$) and polynomials in the frequency domain are described by an additional hat-symbol (e.g. $\hat{\mathbf{a}}$). Polynomials that have been compressed by the NHSCompress function are marked by a bar (e.g. $\bar{\mathbf{a}}$).

[3] https://www.seceng.rub.de/research/publications/implementing-newhope-simple-key-exchange-low-cost-/

2.2 The NewHope Scheme

In this paper we implemented the `Simple` version [1] of the `NewHope` protocol [2] that improves previous approaches to lattice-based key exchange [8,11,17]. `NewHope` is a server-client key exchange protocol as described in Protocol 1. Its security is based on the ring learning with errors problem. Note that `NewHope` is unauthenticated. Thus, an additional signature scheme is required. The scheme is parametrized by a lattice dimension n, a modulus q, and a standard deviation $\sigma = \sqrt{k/2}$ where k is used as a parameter for the binomial sampler. In this work, we implemented the scheme with the parameters $n = 1024$, $q = 12289$, and $k = 16$. As stated in [1], the security of `NewHope-Simple` is the same as the security of `NewHope` and therefore the chosen parameters yield at least a post-quantum security level of 128 bits with a comfortable margin [2], or more specifically 255 bits of security against known quantum attackers and 199 bits of security against the best plausible attackers.

Parameters: $q = 12289 < 2^{14},\ n = 1024$
Error distribution: ψ_{16}^n

Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,\ldots,255\}^{32}$		
$\hat{\mathbf{a}} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\hat{\mathbf{s}} \leftarrow \mathsf{NTT}(\mathbf{s})$		
$\hat{\mathbf{b}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{s}} + \mathsf{NTT}(\mathbf{e})$	$\xrightarrow[1824\ \text{Bytes}]{m_a = \mathsf{encodeA}(seed, \hat{\mathbf{b}})}$	$(\hat{\mathbf{b}}, seed) \leftarrow \mathsf{decodeA}(m_a)$
		$\hat{\mathbf{a}} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\hat{\mathbf{t}} \leftarrow \mathsf{NTT}(\mathbf{s}')$
		$\hat{\mathbf{u}} \leftarrow \hat{\mathbf{a}} \circ \hat{\mathbf{t}} + \mathsf{NTT}(\mathbf{e}')$
		$\nu \xleftarrow{\$} \{0,\ldots,255\}^{32}$
		$\nu' \leftarrow \mathsf{SHA3\text{-}256}(\nu)$
		$\mathbf{k} \leftarrow \mathsf{NHSEncode}(\nu')$
		$\mathbf{c} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{b}} \circ \hat{\mathbf{t}}) + \mathbf{e}'' + \mathbf{k}$
$(\hat{\mathbf{u}}, \bar{\mathbf{c}}) \leftarrow \mathsf{decodeB}(m_b)$	$\xleftarrow[2048\ \text{Bytes}]{m_b = \mathsf{encodeB}(\hat{\mathbf{u}}, \bar{\mathbf{c}})}$	$\bar{\mathbf{c}} \leftarrow \mathsf{NHSCompress}(\mathbf{c})$
$\mathbf{c}' \leftarrow \mathsf{NHSDecompress}(\bar{\mathbf{c}})$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu')$
$\mathbf{k}' \leftarrow \mathbf{c}' - \mathsf{NTT}^{-1}(\hat{\mathbf{u}} \circ \hat{\mathbf{s}})$		
$\nu' \leftarrow \mathsf{NHSDecode}(\mathbf{k}')$		
$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu')$		

Protocol 1: A full description of the `NewHope-Simple` key exchange. The functions $\mathsf{NHSEncode}, \mathsf{NHSDecode}, \mathsf{NHSCompress}$, and $\mathsf{NHSDecompress}$ are defined in [1]. The functions $\mathsf{encodeA}, \mathsf{encodeB}, \mathsf{decodeA}$, and $\mathsf{decodeB}$ describe a simple transformation into a representation that dependents on the channel over which the information will be sent (e.g. bit-wise or byte-wise format).

The most notable difference between `NewHope` and `NewHope-Simple` is that `NewHope` avoids the error-reconciliation mechanism originally proposed by Ding [11]. As a consequence, `NewHope-Simple` is less complex but also 6.25% bigger in terms of the size of the transmitted messages. The polynomial $\mathbf{a}$ could be fixed to a constant. The authors of `NewHope` decided to generate a fresh $\mathbf{a}$ for every run to prevent backdoors and all-for-the-price-of-one attacks.

The idea behind the key exchange is that the server generates an ephemeral key pair and transmits the public key to the client. The client uses the public key to encrypt a secret symmetric key and transmits the ciphertext to the server. The server decrypts the ciphertext to retrieve the same symmetric key that can be used for further communication. As the scheme is based on the ring learning with errors problem, error polynomials are used to hide the symmetric key in the ciphertext. Therefore an error correction mechanism is required to recover the symmetric key. `NewHope-Simple` itself has no built-in authentication, it relies on external authentication, for instance with a signature scheme. The major components of the scheme are a Parse function that is used to generate $\mathbf{a}$, a binomial sampler to generate error polynomials, the number-theoretic transform (NTT) to speed up polynomial multiplication, and the Keccak function that is used to compute SHA3-256 hashes and for the SHAKE-128 extendable output function.

2.3 Binomial Sampling

In [2] a binomial sampler is used as substitution for the Gaussian sampler that is required in many lattice-based schemes. The discrete, centered Gaussian distribution is defined by assigning a weight proportional to $\exp(\frac{-x^2}{2\sigma^2})$ where σ is the standard deviation of the Gaussian distribution. According to [2] the binomial distribution that is parametrized by $k = 2\sigma^2$ is sufficiently close to a discrete Gaussian distribution with standard deviation σ and does not significantly decrease the security level. A binomial sampler is basically realized by uniformly sampling two k-bit vectors and computing their respective Hamming weights. The binomial distributed result is obtained by subtracting the Hamming weights of both bit vectors. Binomial sampling does not require any look-up tables and has a constant runtime. But as k scales quadratically with σ the binomial approach is only suited for small σ as used in lattice-based encryption or key exchange schemes. Signature schemes usually require larger standard deviations.

2.4 Number-Theoretic Transform (NTT)

The number-theoretic transform (NTT) is a discrete Fourier transform over a finite field. An interesting property of the discrete Fourier transform, which is also highly interesting for lattice-based cryptography, is the ability to reduce the overall complexity of (polynomial) multiplication to $\mathcal{O}(n \cdot \log n)$. To allow efficient computation of the NTT the coefficient ring has to contain primitive roots of unity.

Definition 1 (Primitive root of unity [12]). *Let $\mathcal{R}$ be a ring, $n \in \mathbb{N}_{\geq 1}$, and $\omega \in \mathcal{R}$. The value ω is an n-th root of unity if $\omega^n = 1$. The value ω is a primitive n-th root of unity (or root of unity of order n) if it is an n-th root of unity, $n \in \mathcal{R}$ is a unit in $\mathcal{R}$, and $\omega^{n/t} - 1$ is not a zero divisor for any prime divisor t of n.*

For a given primitive n-th root of unity ω in Z_q, the NTT of a vector $a = (a_{n-1}, \ldots, a_0)$ is the vector $A = (A_{n-1}, \ldots, A_0)$ and computed as

$$A_i = \sum_{0 \leq j < n} a_j \omega^{ij} \bmod q,\ i = 0, 1, \ldots, n-1.$$

The idea is to transform two polynomials $a = a_{n-1} \cdot x^{n-1} + \ldots + a_0$ and $b = b_{n-1} \cdot x^{n-1} + \ldots + b_0$ into their NTT representations $A = A_{n-1} \cdot x^{n-1} + \ldots + A_0$ and $B = B_{n-1} \cdot x^{n-1} + \ldots + B_0$ and computing the coefficient-wise multiplication as $C = \sum_{0 \leq i < n} A_i \cdot B_i \cdot x^i$. The result $c = a \cdot b$ is obtained after applying the inverse transform to C. For $q = 1 \bmod 2n$ the way the result has to be interpreted depends on the input.

- Assuming one expanded a and b to vectors of length $2n$ by padding n zeros, the result c equals the schoolbook multiplication of a and b without reduction.
- Without padding, the result c is already reduced modulo $f = x^n - 1$. This is called the *positive* wrapped convolution. In contrast to the first case, the resulting polynomial is only of degree n.

This reduction for free is beneficial concerning the computation time, but for `NewHope` one performs arithmetic in $\mathbb{Z}[x]/\langle x^n + 1 \rangle$. Thus, the input and output have to be modified so that the *negative* wrapped convolution gets computed to exploit the reduction property. Let ψ be the square root of ω. Now one computes $a' = \sum_{0 \leq i < n} a_i \cdot \psi^i \cdot x^i$ and $b' = \sum_{0 \leq i < n} b_i \cdot \psi^i \cdot x^i$ before the polynomials are transformed into their NTT representation. To obtain $a \cdot b \bmod x^n + 1$, one also has to multiply c', the output of the inverse transform of C, by powers of the *inverse* of ψ.
There are many ways to compute the number-theoretic transform. In this work, we follow the optimized approach from [21]. For a complete description of the algorithms we refer to [21].

3 FPGA Implementation

In this chapter, we present the details of our implementation and explain our design decisions.

3.1 Overview

Our target device is a Xilinx Artix-7 FPGA. It features DSPs blocks that can multiply, add, and subtract and have a configurable number of pipeline stages. It furthermore has several 18 Kb block memories that can be used in dual-port mode. The LUTs of the Artix-7 can either be used as 6-input LUTs or 5-input

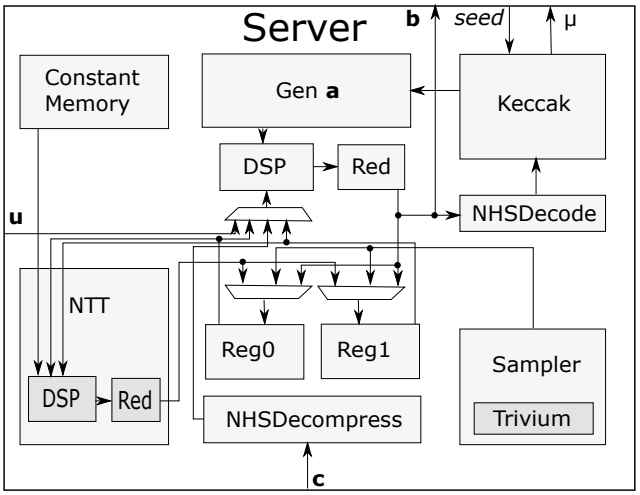

Fig. 1. Our server architecture.

LUTs with two outputs. Figure 1 and Figure 2 provide an overview of our server architecture and the client architecture. We incorporated two read/write-BRAM blocks with a width of 14 bits and a depth of $n = 1024$ in dual-port mode to store polynomials ($Reg0$ and $Reg1$). Two read-only memories are used to store the twiddle factors used in the NTT. One DSP block with subsequent modular reduction serves as general-purpose DSP that is used in most sub-modules, like the NTT or the point-wise multiplication.

The client and the server side of the scheme contain almost the same set of operations. However there are slight differences, for instance the decoding operation is replaced by an encoding and the decompression is replaced by a compression module. We decided to develop two separate modules for the server and the client side as we expect an embedded device to usually be either server or client but not both. Applications that require both sides can share many of the components, like the NTT or the sampler. A neat solution for such an applications could be to replace the components that are required by only one side through dynamic partial reconfiguration. In Algorithm 1 we present the temporal structure of our implementation in pseudocode. Each line of Algorithm 1 lists operations that are executed simultaneously. All operations are constant-time, i.e. the execution time is independent from data that is processed and thus the implementation is invulnerable to timing attacks. We employed the modular reduction from [18] for the prime $q = 12289$.

3.2 Efficient Implementation of NTT

The optimized NTT approach from [21] uses a Cooley-Tukey butterfly for the forward transformation and a Gentleman-Sande butterfly for the backwards trans-

Algorithm 1 Pseudocode for our implementation. Operations in the same line are executed simultaneously.

procedure NewHope(Registers R_0, R_1)
  Server-side computations:
  $R_0 \leftarrow \mathsf{Sample}()$
  $R_0 \leftarrow \mathsf{NTT}(R_0)$; $R_1 \leftarrow \mathsf{Sample}()$
  $R_1 \leftarrow \mathsf{NTT}(R_1)$
  $R_1 \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed)) \circ R_0 + R_1$
  Transmit $seed$ and $\hat{\mathbf{b}} = R_1$

  Client-side computations:
  $R_0 \leftarrow \mathsf{Sample}()$
  $R_0 \leftarrow \mathsf{NTT}(R_0)$; $R_1 \leftarrow \mathsf{Sample}()$
  $R_1 \leftarrow \mathsf{NTT}(R_1)$
  $R_1 \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed)) \circ R_0 + R_1$
  Transmit $\hat{\mathbf{u}} = R_1$
  $R_0 \leftarrow R_0 \circ \hat{\mathbf{b}}$
  $R_0 \leftarrow \mathsf{NTT}^{-1}(R_0)$; $R_1 \leftarrow \mathsf{Sample}()$
  $R_0 \leftarrow \mathsf{NHSCompress}(R_0 + R_1 + \mathsf{NHSEncode}(\nu'))$; $\nu' \leftarrow \mathsf{SHA3\text{-}256}(random)$
  $\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu')$
  Transmit $\bar{\mathbf{c}} = R_0$

  Server-side computations:
  $R_0 \leftarrow \hat{\mathbf{u}} \circ R_0$
  $R_0 \leftarrow \mathsf{NHSDecompress}(\bar{\mathbf{c}}) - \mathsf{NTT}^{-1}(R_0)$
  $\mu \leftarrow \mathsf{SHA3\text{-}256}(\mathsf{NHSDecode}(R_0))$
end procedure

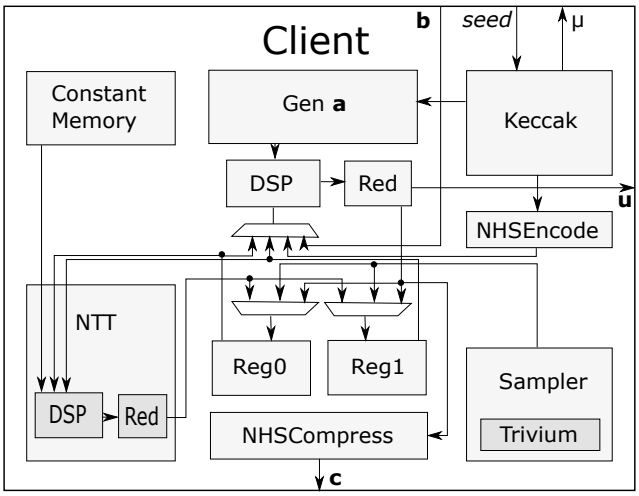

Fig. 2. Our client architecture.

formation. A butterfly takes two coefficients as input and combines them according to the butterfly instructions. As the transformation operates in-place, it outputs two coefficients that replace the input coefficients. Both butterfly constructions can be computed with a regular Artix-7 DSP block. To accelerate the butterfly computation, we use two DSP blocks and compute both output coefficients in parallel. We use the multi-purpose DSP of our `NewHope-Simple` core and add another DSP to the NTT core. As the lattice dimension in `NewHope-Simple` is $n = 1024$, the computation of the NTT could be further parallelized, i.e. up to 512 butterflies could be computed in parallel. However, this would require more DSPs and a complicated memory scheduling as we can only access two coefficients at the same time with a dual-port memory block. Therefore we decided to compute the butterflies serially to keep the area consumption of the implementation low. The only other difference between the forward and the backward transformation is the address generation for the memory blocks storing the input coefficients. Hence minimal changes to the state machine allow our NTT core to be able to perform both operations, the forward and the backward transformation. The computation of a butterfly takes 7 clock cycles that consist of 2 cycles for the memory access, 1 cycle for the DSP calculation and 4 cycles for the modular reduction. In total, $\frac{n}{2}\log(n)$ butterfly computation per transformation are necessary, therefore $512 \cdot 10 \cdot 7 = 35,840$ cycles.

3.3 Point-Wise Multiplication

Our implementation of the point-wise multiplication is straight-forward. We use the multi-purpose DSP of our `NewHope-Simple` core with subsequent modular reduction to serially compute the product of two coefficients. Therefore the

point-wise multiplication is a simple counter to access the coefficients of the polynomials to be multiplied iteratively. A point-wise multiplication of two coefficients takes 9 cycles. It could be sped up to take only 7 cycles like a butterfly operation but to keep the point-wise multiplication in sync with the generation of the polynomial $\mathbf{a}$ (see below) we slow it down to 9 cycles per coefficient, i.e. $1024 \cdot 9 = 9,216$ cycles in total.

3.4 Generation of a

The implementation of the generation of the Parse function is a challenging task. Usually a b-bit number is sampled from a source of uniform randomness for a modulus q with $2^{b-1} < q < 2^b$. Simply applying a modular reduction to this number to obtain a result in $[0, q-1]$ will introduce a bias in the result. Thus the authors of `NewHope` reject any sampled number that is larger or equal to the modulus and restarting the sampling. Gueron and Schlieker [13] proposed a faster method to generate the polynomial $\mathbf{a}$. By increasing b such that $2^{b-1} < 5q < 2^b$ they managed to reduce the rejection rate from 25% per sample to 6% per sample. However, this method requires up to four subtractions of the modulus q to get a properly reduced result.

For our implementation, we avoided these subtractions and sampled $\lceil \log_2 12289 \rceil = 14$ bits. But we still want to achieve a lower rejection rate. Thus we perform one execution of `SHAKE-128` that gives us 1344 bits of pseudo-randomness. Three processes analyze this randomness buffer word-by-word in parallel and store 14-bit words that are smaller than the modulus in a buffer. As 96 14-bit words fit into the 1344-bit output of `SHAKE-128`, the probability that less than three words are found that are smaller than the modulus is $0.25^{96} + 96 \cdot 0.75 \cdot 0.25^{95} + \frac{96 \cdot 95}{2} \cdot 0.75^2 \cdot 0.25^{94} \approx 2^{-178}$ and can therefore be neglected.

To generate the 1344-bit output of `SHAKE-128` we need 27 cycles. As the point-wise multiplication takes 9 cycles, we can run it three times during the execution of `SHAKE-128`. Therefore, the runtime of the generation of $\mathbf{a}$ is equal to the runtime of the point-wise multiplication, i.e. $9,216$ cycles.

3.5 Binomial Sampling

We implemented a binomial sampler that counts the Hamming weight of two k-bit vectors and subtracts theses Hamming weights. The required randomness is generated by a `Trivium` PRNG [18] that outputs one bit per clock cycle. This implementation of `Trivium` uses a fixed seed that in practice would have to be generated from a secure random number generator. However, true random number generators in FPGAs is a research field on its own and out of scope for this work. As $k = 16$ the generation of a binomial distributed sample takes 33 cycles and thus $33,792$ cycles for an entire polynomial. More instances of the Trivium PRNG would accelerate the generation of the samples. However, we refrained from applying this optimization to keep the implementation small. Only the generation of the first error polynomial of each party has an influence on the performance as the remaining error polynomials can be generated during other

computations (like the NTT). One possible optimization would be to perform the generation of the first error polynomial in an offline computation so that every time a key exchange is triggered, an error polynomial is already available.

3.6 Hash Function

`NewHope-Simple` requires the instantiation of a hash function and an extendable output function. Thus our design contains a `Keccak` core that is able to compute both, `SHA3-256` and `SHAKE-128`. Our implementation of Keccak executes one round per clock cycle. To synchronize it with the generation of $\mathbf{a}$, we slow it down to take 27 cycles for the entire 24 rounds of `Keccak`.

3.7 Compression

The compression function as described in [1] requires a division by q. However, as the modulus q is fixed and the result is limited to $[0, 8]$, we precompute the thresholds at which the result of the division changes and use a simple multiplexer cascade to implement the division. By doing so we obtain the compressed result in two clock cycles. Similarly, the decompression requires a multiplication by q. Again, we use multiplexers as the input is limited to $[0, 7]$.

4 Results and Comparison

In this chapter, we discuss the results of our implementation and compare it with others.

4.1 Evaluation Methodology

We implemented `NewHope-Simple` for a Xilinx Artix-7 FPGA with the part number XC7AA35TCPG236. Our development environment was Xilinx Vivado v2015.3. If not stated otherwise all results were obtained after post-place and route (Post-PAR).

4.2 Results

We optimized our implementation for area-efficiency. Our implementation takes 1,483 slices, 4,498 LUTs, and 4,635 FFs for the client and 1,708 slices, 5,142 LUTs, and 4,452 FFs for the server. We restricted our design to two DSPs to address a use-case for moderate throughput. Note that the use of additional DSPs will lead to a considerable speed-up. Our implementation uses four 18-kb block memories, two for the NTT twiddle factors and two as temporary storage for intermediate polynomials. The overall runtime of 350,416 cycles is divided into 115,784 cycles for the first set of server-side operations, 179,292 cycles for the client-side operations, and 55,340 cycles for the second set of server-side operations. The client-side design was successfully placed and routed with a

Table 1. This table presents the exact cycle counts for our implementation. The line numbers given in the table refer to the line numbers of Algorithm 1 and are followed by a short summary of the respective step.

Operations (server)	**Cycles**	**Operations (client)**	**Cycles**
Line 3: Sampling	33,794	Line 10: Sampling	33,794
Line 4: Sampling + NTT	35,843	Line 11: NTT + Sampling	35,843
Line 5: NTT	35,845	Line 12: NTT	35,845
Line 6: Parse+Multiplication	9,277	Line 13: Parse + Multiplication	9,277
Line 7: Output $\hat{\mathbf{b}}$	1,025	Line 14: Output $\hat{\mathbf{u}}$	1,025
Total	**115,784**	Line 15: Multiplication	9,220
		Line 16: Inverse NTT+Sampling	35,845
Line 22: Multiplication	9,219	Line 16: Multiplication with n^{-1}	9,221
Line 23: Inverse NTT	35,845	Line 17-19: Encode + Output $\bar{\mathbf{c}}$	9,219
Line 23: Multiplication with n^{-1}	9,219	**Total**	**179,292**
Line 24: Decode	1,028		
Line 25: Hashing	29		
Total	**55,340**		

maximum frequency is thus 117 MHz. The maximum path delay is 8.037 ns and is located between the DSP output and the modular reduction. The server-side design achieved a slightly better performance and was successfully placed and routed with a a frequency of 125 MHz. In this case the maximum path delay is 7.179 ns and located between the output of the modular reduction and the input of the BRAM. The actual cycle counts for our implementation are listed in Table 1.

4.3 Comparison

To the best of our knowledge, this is the first hardware implementation of `NewHope` or `NewHope-Simple`. Hence, a comparison with previous work is somewhat difficult. We therefore add references to works that implement basic encryption schemes, such as `ring-LWE` or lattice-based `IBE`. Table 2 summarizes our results and previous related work. However, please note again that a straight comparison of the presented schemes is neither fair nor possible, for the following reasons:

- **Parameter sizes.** When comparing the implementations we have to consider that in our implementation the lattice dimension is $n = 1024$ while most other implementations use $n = 512$ or even $n = 256$. The lattice dimension n determines how much memory is needed to store polynomials. It furthermore has a linear influence on the run time of every module beside the `Keccak` core. The `NTT` is even slowed down by a factor of 2.22. when increasing n from 512 to 1024.

- **Additional components.** The implementation of `NewHope-Simple` requires a number of components that are not present in `ring-LWE`, `standard-LWE`, and `IBE`. Especially NHSCompress, NHSDecompress, SHA3-256, and SHAKE-128 are required by `NewHope-Simple` only and thus lead to a higher resource consumption.
- **Key generation.** The `NewHope-Simple` protocol basically performs all three `ring-LWE` operations: key generation, encryption, decryption. The works of [14,23,15] only present implementations of the encryption and the decryption. Thus, the cost of the key generation would have to be added first for a fair comparison.
- **Precomputation.** `NewHope-Simple` requires the on-the-fly generation of the public polynomial **a**, while previous work usually assumes that **a** is a global constant and is thus treated as precomputed value. In contrast to that, `NewHope-Simple` requires the implementation of a Parse function. Furthermore, to minimize the communication cost, both parties have to generate **a** while in lattice-based encryption schemes, **a** is usually generated only once during the key generation if not assumed to be a global constant anyway. Thus, we have to spend additional cycles and FPGA resources to meet the requirement of generating **a** on-the-fly.
- **Security level.** The authors of `NewHope-Simple` claim a security level of 255 bits against known quantum attackers [2] while the works on `ring-LWE` have a much lower security level of 131 bits against known quantum attackers [16]. The work of Roy et al. [23] further reduces the security as they limit the secret key to have binary coefficients instead of Gaussian distributed coefficients without discussing the implications on the security level. Such a limitation has a huge impact on the performance as a polynomial multiplication can be replaced by simple additions. We decided to stick to the recommendations of [2] as we do not want to lower the security level significantly. The parameters for lattice-based `IBE` are chosen to have a 80-bit resp. 192-bit security level against *classical* attackers.

Considering the aforementioned factors, our implementation compares well to other lattice-based schemes. Further optimization might even lead to a smaller implementation or better performance.

5 Conclusion

In this work, we presented the first implementation of the `NewHope-Simple` key exchange. The scheme is arguably one of the most promising candidates for quantum-secure key exchange. Hence, we expect a high interest in an instantiation of `NewHope-Simple` in hardware. We demonstrate that `NewHope-Simple` is well suited for implementations on constrained hardware devices and still maintains a decent performance on available platforms. To allow independent verification of our results and further improvements, our source files will be made publicly available with publication of this work.

Table 2. Our implementation results in comparison with implementations of similar schemes. The first row denotes the server-side operations for key exchange and encryption for encryption schemes. The second row denotes the client-side operation for key exchange and decryption for encryption schemes. If only one of the two rows contains numbers for the resource consumption, the authors of the respective work present a combined implementation for both operations. The clock frequency is given in MHz.

Implementation	Clock	(LUT-FF-BRAM-DSP)	Cycles
`NewHope-Simple` (XC7A35T, **this work**)	125	(5,142 - 4,452 - 4 - 2)	171,124
(1024/12289)	117	(4,498 - 4,635 - 4 - 2)	179,292
`IBE` (S6LX25, [14])	174	(7,023 - 6,067 - 16 - 4)	13,958
(512/16813057)			9,530
`IBE` (S6LX25, [14])	174	(8,882 - 8,686 - 27 - 4)	28,586
(1024/134348801)			19,535
`ring-LWE` (V6LX75T, [19])	251	(5,595 - 4,760 - 14 - 1)	13,769
(512/12289)			8,883
`ring-LWE` (V6LX75T, [23])	278	(1,536 - 953 - 3 - 1)	13,300
(512/12289)			5,800
`standard-LWE` (S6LX45, [15])	125	(6,078 - 4,676 - 73 - 1)	98,304
(256/4096)	144	(63 - 58 - 13 - 1)	32,768

For future work, we plan to further improve the performance of the implementation. Especially the NTT could benefit from some ideas that Roy et al. incorporated in their implementation of `ring-LWE` [24]. Furthermore an in-depth analysis of side-channel vulnerabilities of the scheme is required before `NewHope` hardware accelerators could be deployed in the field. Due to the ephemeral nature of the scheme, an attacker is limited to a single execution to gain side-channel information. Nevertheless, simple power analysis or template attacks should be considered.

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: NewHope without reconciliation (2016), http://cryptojedi.org/papers/#newhopesimple
2. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange – a new hope. In: Proceedings of the 25th USENIX Security Symposium. USENIX Association (2016), document ID: 0462d84a3d34b12b75e8f5e4ca032869, http://cryptojedi.org/papers/#newhope
3. Alkim, E., Ducas, L., Pppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092 (2015), http://eprint.iacr.org/2015/1092
4. Alkim, E., Jakubeit, P., Schwabe, P.: A new hope on ARM cortex-m. IACR Cryptology ePrint Archive 2016, 758 (2016), http://eprint.iacr.org/2016/758
5. Batina, L., Robshaw, M. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, LNCS, vol. 8731. Springer (2014)

6. Bauer, B., Wecker, D., Millis, A.J., Hastings, M.B., Troyer, M.: Hybrid quantum-classical approach to correlated materials. Physical Review X 6(3), 031045 (2016)
7. Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1006–1018. ACM (2016)
8. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 553–570. IEEE Computer Society (2015), https://doi.org/10.1109/SP.2015.40
9. Braithwaite, M.: Experimenting with post-quantum cryptography. Google Security Blog 7 (2016)
10. Del Pino, R., Lyubashevsky, V., Pointcheval, D.: The whole is less than the sum of its parts: Constructing more efficient lattice-based akes. In: International Conference on Security and Cryptography for Networks. pp. 273–291. Springer (2016)
11. Ding, J.: A simple provably secure key exchange scheme based on the learning with errors problem. IACR Cryptology ePrint Archive 2012, 688 (2012), http://eprint.iacr.org/2012/688
12. Gathen, J.V.Z., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, New York, NY, USA, 2 edn. (2003)
13. Gueron, S., Schlieker, F.: Speeding up R-LWE post-quantum key exchange. IACR Cryptology ePrint Archive 2016, 467 (2016), http://eprint.iacr.org/2016/467
14. Güneysu, T., Oder, T.: Towards lightweight identity-based encryption for the post-quantum-secure internet of things. In: 18th International Symposium on Quality Electronic Design, ISQED 2017, Santa Clara, CA, USA, March 14-15, 2017. pp. 319–324. IEEE (2017), https://doi.org/10.1109/ISQED.2017.7918335
15. Howe, J., Moore, C., O'Neill, M., Regazzoni, F., Güneysu, T., Beeden, K.: Standard lattices in hardware. In: Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016. pp. 162:1–162:6. ACM (2016), http://doi.acm.org/10.1145/2897937.2898037
16. Oder, T., Schneider, T., Pppelmann, T., Gneysu, T.: Practical cca2-secure and masked ring-lwe implementation. Cryptology ePrint Archive, Report 2016/1109 (2016), http://eprint.iacr.org/2016/1109
17. Peikert, C.: Lattice cryptography for the Internet. In: Mosca, M. (ed.) Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings. LNCS, vol. 8772, pp. 197–219. Springer (2014), http://dx.doi.org/10.1007/978-3-319-11659-4_12
18. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: Batina and Robshaw [5], pp. 353–370
19. Pöppelmann, T., Güneysu, T.: Towards practical lattice-based public-key encryption on reconfigurable hardware. In: Lange, T., Lauter, K.E., Lisonek, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 68–85. Springer (2013)
20. Pöppelmann, T., Güneysu, T.: Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In: IEEE International Symposium on Circuits and Systemss, ISCAS 2014, Melbourne, Victoria, Australia, June 1-5, 2014. pp. 2796–2799. IEEE (2014), http://dx.doi.org/10.1109/ISCAS.2014.6865754
21. Pöppelmann, T., Oder, T., Güneysu, T.: High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In: Lauter, K.E., Rodríguez-Henríquez, F. (eds.) Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America,

Guadalajara, Mexico, August 23-26, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9230, pp. 346–365. Springer (2015)
22. PQCRYPTO-EU-project: TU Eindhoven leads multi-million euro project to protect data against quantum computers (2016), https://pqcrypto.eu.org/press/press-release-post-quantum-cryptography-ENGLISH.docx
23. Roy, S.S., Vercauteren, F., Mentens, N., Chen, D.D., Verbauwhede, I.: Compact Ring-LWE based cryptoprocessor. Eprint Archive 2013, 866 (2013)
24. Roy, S.S., Vercauteren, F., Mentens, N., Chen, D.D., Verbauwhede, I.: Compact Ring-LWE cryptoprocessor. In: Batina and Robshaw [5], pp. 371–391
25. of Standards, N.I., Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf