EasyChair Preprint

№ 8760

FPGA Implementation of BIKE for Quantum-Resistant TLS

Andrea Galimberti, Davide Galli, Gabriele Montanaro, William Fornaciari and Davide Zoni

August 30, 2022

FPGA implementation of BIKE for quantum-resistant TLS

Andrea Galimberti
DEIB
Politecnico di Milano
Milano, Italy
andrea.galimberti@polimi.it

Davide Galli
DEIB
Politecnico di Milano
Milano, Italy
davide11.galli@mail.polimi.it

Gabriele Montanaro
DEIB
Politecnico di Milano
Milano, Italy
gabriele.montanaro@polimi.it

William Fornaciari
DEIB
Politecnico di Milano
Milano, Italy
william.fornaciari@polimi.it

Davide Zoni
DEIB
Politecnico di Milano
Milano, Italy
davide.zoni@polimi.it

***Abstract*—The recent advances in quantum computers impose the adoption of post-quantum cryptosystems into secure communication protocols. This work proposes two FPGA-based, client- and server-side hardware architectures to support the integration of the BIKE post-quantum KEM within TLS. Thanks to the parametric hardware design, the paper explores the best option between hardware and software implementations, given a set of available hardware resources and a realistic use-case scenario. The experimental evaluation comparing our client and server designs against the reference AVX2 and hardware implementations of BIKE highlighted two aspects. First, the proposed client and server architectures outperform the reference hardware implementation of BIKE by eight and four times, respectively. Second, the performance comparison between our client and server designs against the reference AVX2 implementation strongly depends on the available resource. Our solution is almost twice as fast as the AVX2 implementation while implemented on the Artix-7 200 FPGA, while it is up to six times slower when targeting smaller FPGAs, thus motivating a careful analysis of the available hardware resources and the optimization of the design's parallelism before opting for hardware support.**

***Index Terms*—Post-quantum cryptography, code-based cryptography, QC-MDPC codes, hardware accelerators, BIKE, FPGA**

I. Introduction

Public-key cryptography (PKC) [1]–[3] allows exchanging keys over an insecure channel without sharing a secret key. Its technology is at the core of current secure communication protocols such as Transport Layer Security (TLS) [4] and Secure Shell (SSH) [5]. TLS is the most widely used cryptographic protocol, providing encryption to the HTTPS communication protocol, secure communication in mobile apps, and encrypted access to email servers.

Notably, the security of current public-key cryptography relies on the hardness of factoring large integers and of computing discrete logarithms in a cyclic group. However, algorithms such as Shor's [6] can solve these problems in polynomial time on quantum computers, whose recent technological advances threaten to break traditional PKC and, consequently, the secure communication protocols that make use of it [7]. To mitigate such risk, post-quantum cryptography (PQC) aims to develop cryptosystems that are secure against attacks from quantum and classical computers and can interoperate with existing communications protocols and networks [8]. In this scenario, the US National Institute of Standards and Technology (NIST) is currently leading a standardization process to identify a set of post-quantum algorithms to replace current public-key cryptosystems. Submitted proposals span over a wide portion of state of the art in computational theory, including algebraic geometry [9], coding theory [10], and lattice theory [11]. Despite the theoretical differences, each proposal must satisfy two requirements. First, post-quantum cryptosystems must leverage computationally hard problems that even quantum computers cannot solve in polynomial time. Second, their computational complexity and the importance of their adoption in scenarios ranging from HPC to resource-constrained platforms force NIST to also require efficient hardware implementations in addition to traditional software ones. Specifically, the NIST PQC standardization process selected the Intel Haswell CPUs and Xilinx Artix-7 FPGAs as the software and hardware targets.

From the theoretical point of view, code-based cryptography has a remarkably good security track, dating back to the McEliece cryptosystem [10] proposed in 1978, thus motivating its adoption by several candidates of the NIST PQC standardization process. However, the strong security and performance of the original McEliece cryptosystem, which employs binary Goppa codes [12], come at the cost of large memory requirements, in the order of megabits, to store the key pairs. To this end, quasi-cyclic moderate-density parity-check (QC-MDPC) codes [13] emerged as an effective alternative to binary Goppa codes, reducing the key size of code-based cryptosystems to

This work was supported by the EU Horizon 2020 "TEXTAROSSA" project (Grant No. 956831).

tens of kilobits while maintaining security against quantum attacks. BIKE [14] is a key encapsulation mechanism (KEM) based on QC-MDPC codes, and it is a current candidate for standardization in the fourth round of the NIST PQC initiative [15].

Despite the vast literature targeting efficient hardware support for BIKE, each proposal is meant to deliver novel computing platforms aiming either to maximize the performance or to minimize the resource utilization of the hardware, without *i)* considering the integration of the hardware support in current secure communication protocols, such as TLS, and *ii)* addressing the critical question of the actual advantage of employing hardware support in place of an optimized software implementation. Indeed, state-of-the-art lacks a complete comparison to identify the best implementation strategy between hardware and software, considering the available computational resources.

Contributions - This paper presents a complete hardware implementation of BIKE that targets the Xilinx Artix-7 family of FPGAs and supports client and server KEM operations in a quantum-resistant TLS [16]. The proposed architecture leverages a set of state-of-the-art configurable accelerators [17]–[19] that implement the key operations of the KEM primitives to provide the best hardware support. Our architecture is evaluated against the reference AVX2 software [20] and FPGA-based hardware [21] implementations of BIKE.

Apart from the complete hardware implementation of BIKE, the novelty of this work concerns the analysis between different hardware and software state-of-the-art implementations of BIKE. The goal is to highlight the best option between hardware and software implementations, given a set of available hardware resources and a realistic use-case scenario. This is in contrast to previous state-of-the-art, which usually targets the design of efficient hardware support without considering *i)* the application of a complete KEM scheme and *ii)* a comparison between optimized hardware and software implementations. More in detail, the two main contributions of our work are listed below.

- **Efficient hardware support for BIKE in TLS -** The proposed solution delivers two architectures tailored to the client and server operations of a quantum-resistant TLS integrating the BIKE KEM. The parametric components, taken from the state of the art, are integrated into a complete design of the KEM primitives. The experimental results show that the proposed client and server designs outperform the reference hardware implementation of BIKE [21] by 9 and 6 times, respectively.
- **Design space exploration and guidelines -** A design space exploration based on our complexity-based heuristic allows selecting the parameters of the components integrated into the proposed client and server architectures to provide the best hardware support given the available resources. The extensive comparison between the reference AVX2 implementation and our designs demonstrates the performance advantage of the hardware implementation, with a speedup up to $1.91\times$, if the available hardware resources are, at least, those of Artix-7 35 and 200 FPGAs for the TLS server and client use cases, respectively. Otherwise, the software implementation overcomes the hardware one by up to 6 times.

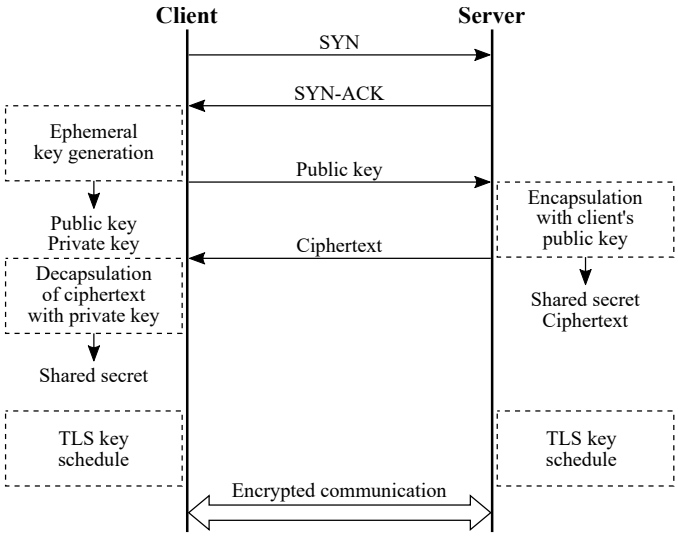

Fig. 1: KEM operations in post-quantum TLS handshake [16].

The rest of the paper is organized into four parts. Section II discusses the background of BIKE and TLS and the state-of-the-art targeting hardware realizations for QC-MDPC code-based cryptosystems. Section III presents the client- and server-side architectures to support BIKE operations in TLS, as well as the complexity-based heuristic used in our design space exploration. Section IV discusses the experimental results, while some design guidelines and conclusions are drawn in Section V.

II. Background and related works

This section discusses the background of BIKE and TLS as well as the state of the art for QC-MDPC code-based cryptosystems. The background discussion emphasizes identifying the computational complexity within the BIKE KEM when employed within TLS. In contrast, the state-of-the-art analysis focuses on the hardware implementations for QC-MPDC codes that can be applied to BIKE.

A. Background

Quantum-resistant TLS - In TLS 1.3, the handshake phase between the client and server nodes is required to establish a shared secret key, useful to generate session keys which are then employed within a symmetric cryptosystem to efficiently exchange encrypted data [4]. The establishment of such shared secret requires making use of public-key cryptography. Diffie-Hellman [2] currently represents the most commonly employed public-key cryptosystem in TLS 1.3. However, the goal of designing a quantum-resistant TLS imposes instead the use of a post-quantum KEM such as the ones currently part of the NIST PQC standardization process [16].

Figure 1 shows the three main steps of the handshake phase between clients and servers when employing a post-quantum KEM in a post-quantum version of TLS [16].

First, the client performs the key generation primitive, producing a private-public key pair and sending the public key to the server. The server node then generates a shared secret and encrypts it with the public key of the client. Finally, the client retrieves the shared secret by decapsulating with its own private key the ciphertext received by the server node. As a result, the client and server endpoints obtained the same shared secret.

TLS 1.3 mandates using ephemeral keys to enforce the perfect forward secrecy (PFS) property. The design of a computationally efficient key generation primitive is thus as crucial as the encapsulation and decapsulation ones.

BIKE key encapsulation mechanism - BIKE [14] is a QC-MDPC code-based KEM, based on the Niederreiter cryptosystem, that leverages quasi-cyclic matrices with coefficients over $\mathbb{Z}_2$. The employed quasi-cyclic matrices are composed of n_0 circulant blocks with size $p \times p$, that can be equivalently represented by n_0 binary polynomials in $GF(2^p)$, with coefficients equal to the first row of the corresponding circulant blocks. The arithmetic of $p \times p$ circulant matrices over $\mathbb{Z}_2$ is equivalent to the arithmetic of binary polynomials in $\mathbb{Z}_2[x]/(x^p+1)$. The addition of two binary polynomials in $\mathbb{Z}_2[x]/(x^p+1)$ corresponds to their bit-wise XOR, while their multiplication consists in their carry-less multiplication followed by a modular reduction with respect to the x^p+1 irreducible polynomial. Moderate-density parity-check codes feature sparse parity-check H matrices, i.e., only a small percentage of values are set to 1, allowing for a sparse representation by enumerating the positions of bits set to 1. QC-MDPC codes possess both the quasi-cyclic and moderate-density properties.

In the rest of this section, we describe the key generation, encapsulation, and decapsulation primitives of the BIKE key encapsulation mechanism by making use of the following notation.

$e = [e_0|e_1]$ is a random n-bit error vector with $t \approx \sqrt{n}$ bits set to 1, where $n = 2p$ and each e_i is a p-bit vector. $H = [h_0|h_1]$ is the private key, composed of two circulant blocks h_i of size $p \times p$, with $v \approx \sqrt{n}$ bits set to 1 for each row of each block h_i. h is the public key, which is a circulant block of size $p \times p$. s is the syndrome. m is a message. c is the ciphertext. K is the shared secret.

Algorithm 1 Key generation primitive of BIKE [14].

1: **function** $[H, \sigma, h]$ KEYGEN ( )
2: $seed =$ TRNG ();
3: $H =$ PRNG(SHAKE256($seed$));
4: $h_{0_{inv}} =$ INVERT(h_0);
5: $h = h_1 \odot h_{0_{inv}}$;
6: $\sigma =$ TRNG ();
7: **return** $\{H, \sigma, h\}$;

Key generation - Algorithm 1 details the key generation primitive, that requires the pseudorandom generation of the public key $H = [h_0|h_1]$ (line 3), the binary polynomial inversion (line 4) of h_0, and the binary polynomial multiplication (line 5) between the sparse h_1, with Hamming weight equal to v, and the dense $h_{0_{inv}}$. The key generation primitive outputs the private key H, the corresponding public key h, and a 256-bit message σ.

Algorithm 2 Encapsulation primitive of BIKE [14].

1: **function** $[K, c]$ ENCAPS (h)
2: $m =$ TRNG ();
3: $e =$ PRNG(SHAKE256(m));
4: $s = e_0 \oplus (e_1 \odot h)$;
5: $m' = m \oplus$ TRUNC$_{256}$(SHA3-384(e));
6: $c = \{s, m'\}$;
7: $K =$ TRUNC$_{256}$(SHA3-384($\{m, c\}$));
8: **return** $\{K, c\}$;

Encapsulation - Algorithm 2 details the encapsulation primitive, which takes as its only input the public key h. Such primitive consists in the execution of the pseudorandom bits generation function, which employs SHAKE256 (see line 3) to generate the n-bit error vector e with Hamming weight equal to t, of a binary polynomial multiplication (see line 5) between the sparse e_1, with Hamming weight up to t, and the dense h, and two hashing operations making use of the SHA3-384 cryptographic hash function, respectively of n-bit (line 5) and $(p+512)$-bit (line 7) messages. The concatenation of the syndrome s and the message m' obtained at lines 4 and 6, respectively, corresponds to the ciphertext c, that can then be sent from the server node to the client endpoint within the handshake phase, while the 256-bit digest K obtained at line 7 is the shared secret.

Algorithm 3 Decapsulation primitive of BIKE [14].

1: **function** $[K]$ DECAPS (H, σ, c)
2: $s' = h_0 \odot s$;
3: $e' =$ DECODE (s', H);
4: $m'' = m' \oplus$ TRUNC$_{256}$(SHA3-384(e'));
5: $a = (e' =$ PRNG(SHAKE256(m''))) ? m'' : σ;
6: $K =$ TRUNC$_{256}$(SHA3-384($\{a, c\}$));
7: **return** K;

Decapsulation - Algorithm 3 lists the key operations that compose the decapsulation KEM primitive, which takes as its inputs the H public key, the ciphertext c, and the σ message. They are one binary polynomial multiplication (line 2 of Algorithm 3), where the h_0 operand is sparse with Hamming weight equal to v, one instance of QC-MDPC bit-flipping decoding (line 3), the computations of two 384-bit hash digests through SHA3-384, respectively of n-bit (line 4) and $(p+512)$-bit (line 6) messages, and the pseudorandom generation (line 5) of a polynomial with Hamming weight equal to t. The decapsulation primitive outputs the shared secret K.

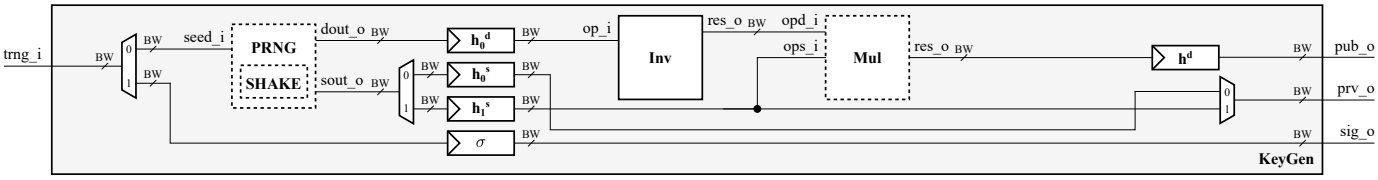

Fig. 2: Top-view architecture of the key generation module. Dashed blocks are shared with the `Decaps` module (see Figure 3).

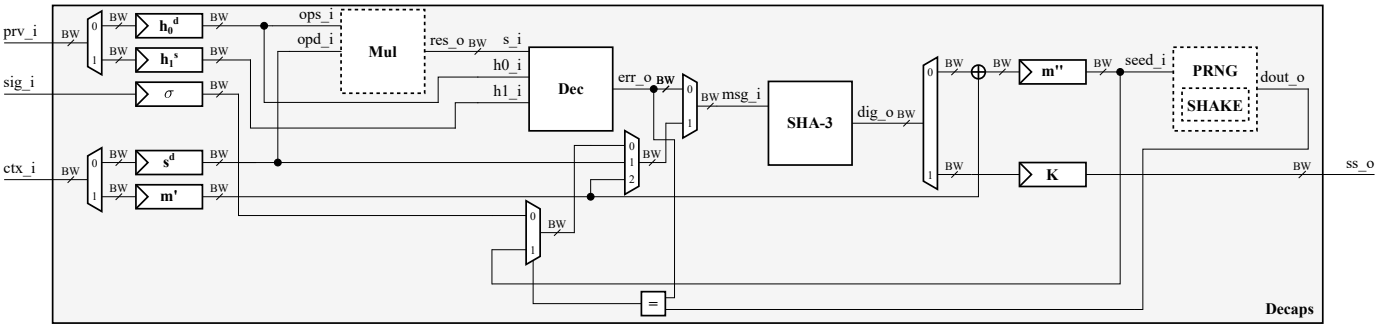

Fig. 3: Top-view architecture of the decapsulation module. Dashed blocks are shared with the `KeyGen` module (see Figure 2).

B. State of the art

The literature contains several proposals targeting efficient hardware implementations for QC-MDPC code-based cryptosystems. [22], [23] proposed the implementation of the McEliece cryptosystem with QC-MDPC codes on FPGAs. In particular, [22] targeted a performance-oriented design while [23] focused on a resource-optimized one. [24] discussed a fast implementation of QC-MDPC Niederreiter encryption for FPGAs, outperforming the work in [22] thanks to using a hardware module to estimate the Hamming weight of large vectors and proposing a hardware implementation tailored to low-area devices for encryption and decryption used in QC-MDPC code-based cryptosystems. [21] presented an FPGA-based implementation of BIKE, which provided support for the key generation, encryption, and decryption KEM primitives, but it is custom-tailored to small FPGA targets.

Several other works focused instead on the optimal design of single key operations to support QC-MDPC code-based cryptosystems. [25] considered the FPGA-based design of two polynomial multipliers for BIKE, one implementing the multiplication between two dense operands and one implementing the multiplications between a dense and a sparse operand. [18] proposed a binary polynomial multiplier for dense-sparse multiplications that can be configured to scale across a variety of FPGA targets. [26] discusses a Karatsuba multiplier for dense binary polynomials, whose performance is, however, insufficient compared to a multiplier tailored to dense-sparse multiplication, when one of the two operands has a low Hamming weight. [19] presented a bit-flipping decoder for generic QC-MDPC codes that is highly configurable in terms of bandwidth and degree of parallelism, allowing it to scale across a range of FPGA targets. [27] also proposed a bit-flipping decoder for QC-MDPC codes, that is, however, only configurable in the bandwidth of its datapath. [17] described a configurable architecture for binary polynomial inversion that scales across the entire Artix-7 FPGA family and makes use of the Karatsuba binary polynomial multiplier proposed in [26].

III. Methodology

This section presents the scalable architectures of BIKE to support client and server KEM operations in TLS and the complexity-oriented heuristic for design space exploration. The scalability of the client and server architectures is obtained by mixing state-of-the-art configurable components for the most complex operations with hard-coded ones. The client and server architectures employ configurable components to implement binary polynomial inversion [17], binary polynomial multiplication [18], and QC-MDPC bit-flipping decoding [19]. The decoding component was adapted to implement the Black-Gray-Flip (BGF) decoding algorithm employed by BIKE and introduced in [28]. Fixed, non-parametric components [29] implement instead the SHA3-384 and SHAKE256 operations. Considering the pseudorandom generation functionality, we wrapped up the SHAKE256 component, as described in [21], to produce random bitvectors with the desired Hamming weight. The proposed design space exploration heuristic leverages the time- and space-complexity analysis of the employed configurable components to steer the fast identification of the combination of parameters that delivers the best hardware support.

The rest of this section is organized into three parts. Section III-A and Section III-B present the client and the server architectures, respectively, while the complexity-oriented heuristic is discussed in Section III-C.

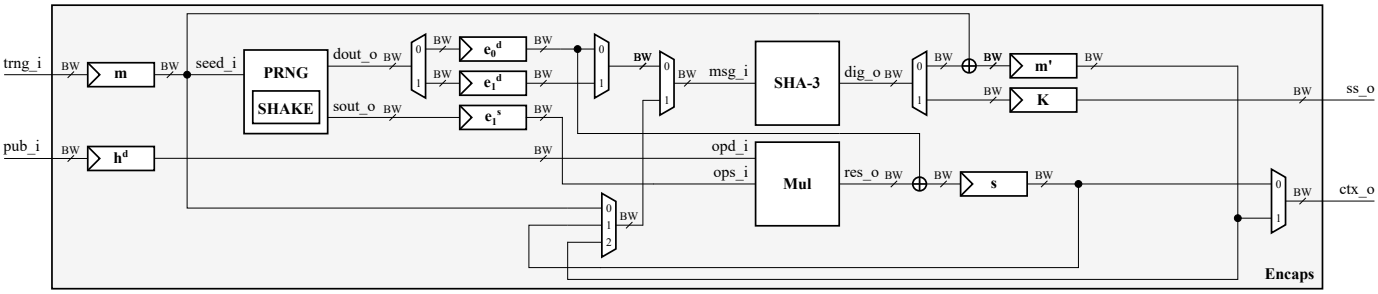

Fig. 4: Top-view architecture of the encapsulation module.

A. Client architecture

The client architecture implements the cryptographic core to support the client-side execution of BIKE in TLS. The architecture of the client consists of two main modules, key generation (`KeyGen`) and decapsulation (`Decaps`), which are depicted in Figure 2 and Figure 3, respectively. To minimize duplicate hardware resources, the pseudorandom generator (`PRNG`) and the multiplier (`Mul`) are shared between the modules (see dashed blocks in Figure 2 and Figure 3).

Key generation module - The `KeyGen` module, depicted in Figure 2, implements Algorithm 1. It features a BW-bit input (`trng_i`) to receive a 256-bit random value from the external true random number generator (TRNG), and outputs the public (h) and private (H) keys using the `pub_o` and `prv_o` outputs, respectively. Notably, the random output value σ that is part of the key generation procedure (see Algorithm 1), is produced by the external TRNG during the key generation procedure and output via the `sig_o` port.

The implementation of the key generation algorithm of BIKE requires performing three subsequent hardware operations, namely pseudorandom generation (`PRNG`), binary polynomial inversion (`Inv`), and binary polynomial multiplication (`Mul`). The `PRNG` component is tasked with generating the private key H, producing the h_0 polynomial in both the sparse and dense forms, and the sparse-only form for h_1. The dense-represented h_0 polynomial is then inverted by the `Inv` component, whose output is finally multiplied by the sparse h_1 within the `Mul` component. `Mul` produces the public key h, that is output through the `pub_o` BW-bit port, while the private key H is output through the `prv_o` BW-bit port.

Decapsulation module - The `Decaps` module,, shown in Figure 3, implements Algorithm 3. It features three BW-bit inputs to receive the private key H (`prv_i`), σ (`sig_i`), and the shared secret encrypted by the server c (`ctx_i`), and it outputs the shared secret K through `ss_o`.

The implementation of the decapsulation primitive of BIKE requires performing a sequence of four hardware operations, namely binary polynomial multiplication (`Mul`), QC-MDPC bit-flipping decoding (`Dec`), computation of SHA-3 hash digest (`SHA-3`), and pseudorandom generation (`PRNG`). The dense syndrome s, which is part of the ciphertext c, is multiplied by the sparse polynomial h_0, obtaining the s' vector, which is then fed to the `Dec` module together with H to decode it and obtain the e' n-bit vector. The latter is hashed by the `SHA-3` module and the resulting digest is XORed with m', obtaining m''. If the result of the pseudorandom generation seeded by m'' is equal to e', then the `SHA-3` module computes the digest of $\{m'', c\}$, i.e., m'' concatenated to the ciphertext c, otherwise it hashes $\{\sigma, c\}$ to avoid information leakage while also raising a decapsulation error flag. In case of a successful decoding, the resulting digest is the shared secret, which is output by the `ss_o` BW-bit port.

B. Server architecture

The server architecture implements the cryptographic core to support the server-side execution of BIKE in TLS. The architecture of the server consists of the encapsulation module (`Encaps`). Although the software execution of the encapsulation is significantly faster than the more complex key generation and decapsulation, the web server scenario foresees a multitude of concurrently active TLS connections, thus mandating for efficient hardware support also for encapsulation.

Encapsulation module - The `Encaps` module, which is depicted in Figure 4, implements Algorithm 2. It takes as inputs a 256-bit random message m and the public key h through the BW-bit `trng_i` and `pub_i` ports, respectively, and outputs the shared secret K, through `ss_i`, and the ciphertext c, i.e., the shared secret encrypted with the public key received from the client, through `ctx_o`.

The implementation of the encapsulation primitive of BIKE requires performing a sequence of three hardware operations, namely pseudorandom generation (`PRNG`), binary polynomial multiplication (`Mul`), and computation of the SHA-3 hash function (`SHA-3`). m is first expanded by the `PRNG` component to generate the random error vector $e = [e_0|e_1]$ with Hamming weight t, and the dense-represented h is then multiplied by the sparse-represented e_1 in the `Mul` component. The resulting product is then XORed with e_0, obtaining the syndrome s. m' is computed by XORing the message m with the SHA-3 hash digest of the error vector e, and the concatenation of s and m' corresponds to the ciphertext c. Finally, the shared secret K is produced as the SHA-3 hash digest of the message m concatenated to the ciphertext c. c is output through the `ctx_o` BW-bit port, while the `ss_o` BW-bit port outputs the shared secret K.

TABLE I: Breakdown of relative execution time for each operation, expressed as a percentage, and total execution time, expressed in milliseconds, of AVX2 software KEM primitives.

Operation	TLS client				TLS server	
	Key generation		Decapsulation		Encapsulation	
	AES-128	AES-192	AES-128	AES-192	AES-128	AES-192
PRNG	7%	5%	2%	1%	29%	33%
Invert	90%	94%	-	-	-	-
Multiply	3%	1%	1%	1%	21%	26%
SHA3-384	-	-	3%	2%	50%	41%
Decode	-	-	93%	95%	-	-
Execution time	0.21	0.69	0.83	2.69	0.05	0.11

C. Design space exploration

In order to provide the best hardware support, the proposed client and server architectures leverage a set of state-of-the-art configurable accelerators for the most complex operations employed within the KEM primitives. However, such flexibility comes at the cost of a broad design space, which imposes the use of an efficient search strategy to minimize the exploration time.

Therefore, a four-step complexity-oriented heuristic drives the design space exploration according to the time and space complexity of the most computationally intensive operations in the three KEM primitives. We note that the overall computation time on the client side can be considered as the sum of the execution times of the key generation and decapsulation KEM primitives [14], while encapsulation represents the sole server-side functionality. Moreover, the configurable components employed to implement multiplication [18], inversion [17], and decoding [19] highlight block RAM (BRAM) as the scarcest resource thus their space complexity can be approximated as the sum of BRAMs used for key generation and decapsulation, on the client side, and for encapsulation, on the server side.

In the following, we describe the four steps of the employed heuristic.

Step 1 - Starting from the computational time of the AVX2 implementation of BIKE, the heuristic computes the fraction of time spent executing each primitive in the server and the client. Table I shows that the fraction of time for key generation and decapsulation is 20% and 80%, respectively, on the client side, while the encapsulation represents the entire server time. Such ratios are used to assign the amount of resources devoted to each KEM primitive module in the client and server architectures.

Step 2 - For each KEM primitive module, the heuristic identifies the operations executed by parametric components that require the largest fraction of execution time. In particular, the heuristic considers the set of parametric operations for which the execution time is at least 90% of the total execution time of the primitive or the entire set of configurable components otherwise. Table I shows that multiplication is the sole parametric operation in our implementation and accounts for up to 26% of the encapsulation time. In contrast, decoding, which is computed by a configurable component, accounts for more than 90% of the execution time in the AVX2 implementation of decapsulation.

Step 3 - For each component or group of components, the heuristic explores time- and space-complexity formulas to identify the combination of parameters that allows maximizing performance within the assigned resource budget. The exhaustive search in the parameter space to find out the best parameter configurations for each module is very fast since it leverages the configurable components' time- and space-complexity formulas without involving any time-consuming hardware synthesis and place-and-route tasks.

Step 4 - The heuristic implements the client and server designs employing the configurations obtained at **Step 3**. Notably, our algorithm is robust and conservative to account for *i)* the non-predictability of the synthesis and implementation of EDA tools, and *ii)* the fact that a small change in the parameters can severely affect the performance and resource utilization. Therefore, the heuristic could land to an unfeasible configuration or to a configuration for which not all the available resources can be used since small increments in the parameters would make it unfeasible within the resource budget. In the former case, the heuristic iteratively re-implements the failed design by lowering the values of the parameters for which the time-complexity formulas show the smallest performance degradation, and this process keeps on until the design becomes feasible. In the latter case, the heuristic iteratively re-implements the non-optimal design by increasing the values of the parameters for which the space-complexity formulas highlight the smallest resource utilization increase, until either the performance improvement is lower than a certain threshold or the design saturates the available hardware resources.

TABLE II: Parameters of QC-MDPC codes for BIKE [14].

Code	Security	p	t	v	$iter$
B1	AES-128	12323	134	71	5
B3	AES-192	24659	199	103	5

IV. Experimental evaluation

This section analyzes the area and performance of the instances of the proposed quantum-resistant TLS client and server architectures identified through the design space exploration. The rest of this section is organized in two parts. Section IV-A provides an overview of the BIKE code parameters and of the experimental setup, while Section IV-B details the area and performance results.

TABLE III: Area results for the proposed and reference [21] client cores, expressed in terms of look-up tables (LUT), flip-flops (FF), and block RAM (BRAM), and relative resource utilization, expressed as a percentage within round brackets.

Code	Our						BIKE [21]					
	Artix-7 50			Artix-7 200			Lightweight (LW)			High-speed (HS)		
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM
B1	31792 (98%)	17805 (27%)	43.5 (58%)	126510 (94%)	51492 (19%)	357 (98%)	11454 (55%)	4602 (11%)	14 (28%)	43084 (32%)	610 (2%)	39 (11%)
B3	31411 (96%)	20181 (31%)	45.5 (61%)	124891 (93%)	53067 (20%)	360 (99%)	-	-	-	-	-	-
Available	32600	65200	75	134600	269200	365	20800	41600	50	134600	269200	365

TABLE IV: Area results for the proposed and reference [21] server cores, expressed in terms of look-up tables (LUT), flip-flops (FF), and block RAM (BRAM), and relative resource utilization, expressed as a percentage within round brackets.

Code	Our						BIKE [21]					
	Artix-7 35			Artix-7 200			Lightweight (LW)			High-speed (HS)		
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM
B1	19804 (95%)	11401 (27%)	30 (60%)	91422 (68%)	46208 (17%)	275.5 (75%)	6730 (32%)	3298 (8%)	3 (6%)	14829 (6%)	3471 (1%)	10 (3%)
B3	19979 (96%)	12282 (30%)	28 (56%)	72725 (54%)	37795 (14%)	235.5 (65%)	-	-	-	-	-	-
Available	20800	41600	50	134600	269200	365	20800	41600	50	134600	269200	365

A. Experimental setup

BIKE code parameters - The proposed architectures target the security levels 1 and 3 of the BIKE KEM, which correspond to AES-128- and AES-192-equivalent security and each with a different underlying QC-MDPC code. Such two security levels are also targeted by the reference software [30] and hardware [31] implementations. The employed QC-MDPC codes have a $2p$-bit code word length and a p-bit information word length. For each BIKE code *Bj*, where *j* corresponds to the security level, Table II reports the size p of h_i blocks of H, the Hamming weight v of the rows of h_i blocks, the Hamming weight t of e, and the number of decoding iterations $iter$.

Hardware setup - The architectures discussed in Section III have been described in SystemVerilog and implemented in Xilinx Vivado 2020.2, targeting Artix-7 FPGAs and a clock frequency of 91 MHz.

All the identified instances satisfy the area constraints given by the available resources on the target FPGAs and the timing requirements, i.e., a 91 MHz clock frequency. For each considered FPGA and code configuration, in the following we only reported the best hardware implementations. Such instances have been identified after a design space exploration that employed the four-step, complexity-oriented heuristic described in Section III-C considering the configurable parameters of the architecture, not described in detail due to the lack of space, that are the parallelism of the sparse-dense multiplier (PAR_M), of the bit-flipping decoder (PAR_D), and of the multiplication (PAR_{I_M}) and exponentiation components (PAR_{I_E}) within the inversion module.

Software setup - The software implementation of BIKE we considered as our software reference is the open source version freely available online [30]. It provides both a baseline C99 portable software implementation and an optimized code that employs the Intel AVX2 extension, thus providing higher performance. The two software versions were executed on an Intel Core i5-10310U CPU, forcing a fixed operating frequency of 4.4 GHz to avoid performance variability due to power management. For each BIKE code configuration, the execution times of key generation, encapsulation, and decapsulation for the C99 and AVX2 software have been obtained as the average of 30 executions.

Functional validation - The proposed architectures have been functionally validated through both post-implementation timing simulation and board prototype execution, checking the correctness of the obtained results against the software implementation of BIKE [30]. For each code configuration, correctness was checked against software execution of 10000 key generation, encapsulation, and decapsulation primitives.

Post-implementation simulation targeted the Xilinx Artix-7 35 (*xc7a35tcpg236-1*), Artix-7 50 (*xc7a50tcpg236-1*), and Artix-7 200 (*xc7a200tsbg484-1*) FPGAs, while board prototype execution targeted the Digilent Nexys 4 DDR board, that features an Artix-7 100 (*xc7a100tcsg324-1*) FPGA. In both cases, we implemented instances of the proposed architectures for each BIKE code configuration and for each target FPGA. Each hardware instance executed 10000 key generations, encapsulations, and decapsulations, whose results were compared with the corresponding outputs of software execution.

B. Experimental results

Area results - The proposed architecture makes extensive use of the FPGA BRAM for storage purposes, allowing the cryptographic core to fit on smaller FPGAs even for codes with a large block size p. Flip-flops would otherwise quickly become the scarcest resources on small FPGAs, due to the need to store multiple p-bit vectors, where p ranges between 12323 and 24659. Indeed, the smallest considered FPGA, i.e., Artix-7 35, features just 41600 flip-flops while, instead, packing 50 36kb BRAM memories, that can store overall up to 1843200 bits.

TABLE V: Client-side execution times, expressed in milliseconds, and speedup over AVX2 software, within round brackets.

Code	CPU BIKE [30]		FPGA Our		FPGA BIKE [21]	
	C99	AVX2	Artix-7 50	Artix-7 200	LW	HS
B1	8.56 (0.12×)	1.03	5.71 (0.18×)	0.58 (1.78×)	35.25 (0.03×)	4.66 (0.22×)
B3	27.65 (0.12×)	3.40	19.27 (0.18×)	1.71 (1.91×)	-	-

TABLE VI: Server-side execution times, expressed in milliseconds, and speedup over AVX2 software, within round brackets.

Code	CPU BIKE [30]		FPGA Our		FPGA BIKE [21]	
	C99	AVX2	Artix-7 35	Artix-7 200	LW	HS
B1	0.28 (0.18×)	0.05	0.03 (1.70×)	0.03 (1.70×)	1.25 (0.04×)	0.13 (0.38×)
B3	0.92 (0.12×)	0.11	0.08 (1.38×)	0.06 (1.83×)	-	-

However, we identified a few factors that concurred to limit the maximum degree of parallelism. For the multiplier component `Mul`, the PAR_M parallelism is bounded by the values of v and t. Concerning the inversion component `Inv`, increasing the PAR_{I_M} parallelism over 3, i.e., implementing parallel computation of 4 or more Karatsuba recursions within the multiplier functional unit, requires a number of LUTs and BRAMs that is not available on any FPGA from the Artix-7 family. The PAR_{I_E} parallelism is instead bounded by the value of the bandwidth BW. Finally, the degree of parallelism PAR_D of the decoding component `Dec` is limited by the imposed timing constraint of a 91MHz clock frequency.

Table III and Table IV detail the resource utilization, in terms of look-up tables (LUT), flip-flops (FF), and block RAM (BRAM), for the instances targeting the Artix-7 35 and 200 FPGAs of the proposed client and server architectures and for the lightweight and high-speed instances of the reference hardware implementations [21]. The reported results demonstrate how the proposed cryptographic cores can scale from the smaller Artix-7 35 FPGA up to the larger Artix-7 200 FPGA. Moreover, they show that BRAM memories are the most used resources, relatively to the ones available on the target chip, on the larger Artix-7 200 FPGAs, while instances targeting the smaller chips are bounded by the LUT utilization. The proposed architectures usually employ a large fraction of the available look-up tables, while requiring a smaller fraction of flip-flops. On the contrary, the state-of-the-art implementation [21] chosen as the hardware reference can not effectively use all the resources available on larger FPGAs, since the high-speed instance employs only 32%, 2%, and 11% of the LUT, FF, and BRAM resources available on the largest Artix-7 FPGA, respectively.

Performance results - Performance of the proposed architectures are assessed by comparing the execution times of client-side and server-side computations to the ones of the C99 and AVX2 reference software and hardware implementations of BIKE. To better evaluate the performance compared to software execution, we define the speedup as the ratio between the execution time of the AVX2 software and the one resulting from the execution on a specific software or hardware instance. A speedup value greater than 1 indicates a performance improvement over the AVX2 software while a value below 1 corresponds to worse performance. Table V reports the performance results for the two software references, i.e., C99 and AVX2, the instances of the proposed client architecture that target the Artix-7 50 and 200 FPGAs, and the lightweight and high-speed instances of the reference hardware implementation [21]. Performance data are reported as the execution time, expressed in milliseconds, and as the corresponding speedup over AVX2 software execution, reported within round brackets. Table VI provides performance data for the server-side hardware support.

The instance of the proposed client architecture targeting the Artix-7 50 FPGA provides a hardware support that is around six times slower than the reference AVX2 software execution, as shown by the speedup of $0.18\times$ for the *B1* and *B3* BIKE configurations. On the contrary, instantiating the client module on the Artix-7 200 FPGA results in a significant performance improvement over the AVX2 reference, with speedups of $1.78\times$ for *B1* and $1.91\times$ for *B3*. Referring to the $B1$ use case in the client scenario, our Artix-7 50 implementation is around six times faster than the lightweight instance of [21], while our Artix-7 200 client implementation is around eight times faster than the high-speed instance of [21].

In the server scenario, both the Artix-7 35 and Artix-7 200 instances improve performance over AVX2. Artix-7 35 provides speedups of $1.70\times$ and $1.38\times$ for *B1* and *B3*, while Artix-7 200 is $1.70\times$ and $1.83\times$ faster than AVX2, respectively. Moreover, both the Artix-7 35 and 200 implementations of our server architectures outperform the high-speed instance of [21].

V. Conclusions

This paper presented a complete hardware implementation of BIKE that targets the Xilinx Artix-7 family of FPGAs and supports client and server KEM operations in a quantum-resistant TLS. Apart from the complete hardware implementation of BIKE, our research explored, thanks to a new complexity-based design space exploration heuristic, the best option between hardware and software implementations, given a set of available hardware resources and a realistic use-case scenario. The experimental evaluation comparing our client and server support against the reference AVX2 and hardware implementations of BIKE highlighted two aspects. First, our implementation always overcomes the reference

hardware implementations of BIKE. Considering the $B1$ use case in the client scenario, our implementations on the Artix-7 50 and Artix-7 200 are six and eight times faster than the lightweight and high-speed instances of [21]. Moreover, both instances of our server architectures outperform the high-speed instance of [21] by more than four times. Second, our client architecture targeting the Artix-7 35 FPGA is around six times slower than the reference AVX2 software execution, while the one targeting the Artix-7 200 is almost twice as fast. Such results highlighted that *i)* the use of the hardware support imposes a careful design of the architecture to maximize the parallelism and *ii)* the optimized software implementation is advantageous in case the hardware resources for a custom accelerator are scarce.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, p. 120–126, Feb. 1978. [Online]. Available: https://doi.org/10.1145/359340.359342

[2] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[3] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228.

[4] S. Turner, "Transport layer security," *IEEE Internet Computing*, vol. 18, no. 6, pp. 60–63, 2014.

[5] T. Ylonen, "Ssh–secure login connections over the internet," in *Proceedings of the 6th USENIX Security Symposium*, vol. 37, 1996.

[6] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[7] National Institute of Standards and Technology (NIST) - U.S. Department of Commerce, "Post-quantum cryptography: A q&a with nist's matt scholl," https://www.nist.gov/blogs/taking-measure/post-quantum-cryptography-qa-nists-matt-scholl, 2021.

[8] ——, "Post-quantum cryptography," https://csrc.nist.gov/projects/post-quantum-cryptography, 2021.

[9] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced oil and vinegar signature schemes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 206–222.

[10] R. J. McEliece, "A Public-Key Cryptosystem Based on Algebraic Coding Theory," *DSN Progress Report*, pp. 114–116, 1978.

[11] J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *Algorithmic Number Theory*, J. P. Buhler, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 267–288.

[12] E. Berlekamp, "Goppa codes," *IEEE Transactions on Information Theory*, vol. 19, no. 5, pp. 590–592, 1973.

[13] M. Baldi, M. Bodrato, and F. Chiaraluce, "A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes," in *Security and Cryptography for Networks, 6th Int.'l Conference, SCN 2008, Amalfi, Italy, Sep. 10-12, 2008. Proc.*, 2008.

[14] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor, "BIKE: Bit flipping key encapsulation - round 3 submission," https://bikesuite.org/files/v4.2/BIKE_Spec.2021.09.29.1.pdf, 2021.

[15] National Institute of Standards and Technology (NIST) - U.S. Department of Commerce, "Nistir 8413, status report on the third round of the nist post-quantum cryptography standardization process," https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf, 2022.

[16] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, *Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH*. New York, NY, USA: Association for Computing Machinery, 2020, p. 149–156. [Online]. Available: https://doi.org/10.1145/3386367.3431305

[17] A. Galimberti, G. Montanaro, and D. Zoni, "Efficient and scalable fpga design of gf(2m) inversion for post-quantum cryptosystems," *IEEE Transactions on Computers*, pp. 1–1, 2022.

[18] A. Barenghi, W. Fornaciari, A. Galimberti, G. Pelosi, and D. Zoni, "Evaluating the trade-offs in the hardware design of the ledacrypt encryption functions," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 739–742.

[19] D. Zoni, A. Galimberti, and W. Fornaciari, "Efficient and scalable fpga-oriented design of qc-ldpc bit-flipping decoders for post-quantum cryptography," *IEEE Access*, vol. 8, pp. 163 419–163 433, 2020.

[20] N. Drucker, S. Gueron, and D. Kostic, "Fast polynomial inversion for post quantum qc-mdpc cryptography," in *Cyber Security Cryptography and Machine Learning*, S. Dolev, V. Kolesnikov, S. Lodha, and G. Weiss, Eds. Cham: Springer International Publishing, 2020, pp. 110–127.

[21] J. Richter-Brockmann, J. Mono, and T. Guneysu, "Folding bike: Scalable hardware implementation for reconfigurable devices," *IEEE Transactions on Computers*, 2021.

[22] S. Heyse, I. von Maurich, and T. Güneysu, "Smaller keys for code-based cryptography: Qc-mdpc mceliece implementations on embedded devices," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, G. Bertoni and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 273–292.

[23] I. von Maurich and T. Güneysu, "Lightweight code-based cryptography: Qc-mdpc mceliece encryption on reconfigurable devices," in *2014 Design, Automation and Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.

[24] J. Hu and R. C. Cheung, "Area-time efficient computation of niederreiter encryption on qc-mdpc codes for embedded hardware," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1313–1325, 2017.

[25] J. Hu, W. Wang, R. C. Cheung, and H. Wang, "Optimized polynomial multiplier over commutative rings on fpgas: A case study on bike," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 231–234.

[26] D. Zoni, A. Galimberti, and W. Fornaciari, "Flexible and scalable fpga-oriented design of multipliers for large binary polynomials," *IEEE Access*, vol. 8, pp. 75 809–75 821, 2020.

[27] K. Koleci, P. Santini, M. Baldi, F. Chiaraluce, M. Martina, and G. Masera, "Efficient hardware implementation of the ledacrypt decoder," *IEEE Access*, vol. 9, pp. 66 223–66 240, 2021.

[28] N. Drucker, S. Gueron, and D. Kostic, "Qc-mdpc decoders with several shades of gray," in *Post-Quantum Cryptography*, J. Ding and J.-P. Tillich, Eds. Cham: Springer International Publishing, 2020, pp. 35–50.

[29] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "Keccak implementation overview," https://keccak.team/obsolete/Keccak-implementation-3.1.pdf, 2011.

[30] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor, "BIKE website," https://www.bikesuite.org/, 2021.

[31] Chair for Security Engineering @ Ruhr-Universität Bochum, "Folding bike: Scalable hardware implementation for reconfigurable devices," https://github.com/Chair-for-Security-Engineering/BIKE, 2021.