An Evaluation of the State-Of-The-Art Software and Hardware Implementations of BIKE

Andrea Galimberti[1]
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy

Gabriele Montanaro
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy

William Fornaciari
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy

Davide Zoni
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy

Abstract

NIST is conducting a process for the standardization of post-quantum cryptosystems, i.e., cryptosystems that are resistant to attacks by both traditional and quantum computers and that can thus substitute the traditional public-key cryptography solutions which are expected to be broken by quantum computers in the next decades. This manuscript provides an overview and a comparison of the existing state-of-the-art implementations of the BIKE QC-MDPC code-based post-quantum KEM, a candidate in NIST's PQC standardization process. We consider both software, hardware, and mixed hardware-software implementations and evaluate their performance and, for hardware ones, their resource utilization.

2012 ACM Subject Classification Security and privacy → Public key encryption; Hardware → Hardware accelerators; Hardware → Hardware-software codesign

Keywords and phrases Post-quantum cryptography, QC-MDPC code-based cryptography, BIKE, software execution, hardware acceleration, hardware-software co-design, performance evaluation

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2023.4

Funding This work was partially supported by SIAE MICROELETTRONICA, the EU Horizon 2020 "TEXTAROSSA" project (Grant No. 956831), and the ICSC National Research Center in High-Performance Computing, Big Data and Quantum Computing.

1 Introduction

Traditional public-key cryptosystems (PKC), including RSA [27], ECDSA [6], and Diffie-Hellman [11], underpin cryptographically secure key exchange mechanisms and digital signature schemes. Such cryptoschemes are however expected to be broken by quantum computers in the upcoming decades [23]. The threat posed by quantum computers requires the definition and the design of alternative cryptosystems that perform the same functions as PKC ones, maintaining security against traditional computer attacks while ensuring security against quantum computer attacks. Post-quantum cryptography (PQC) aims to develop cryptosystems that are resistant to both traditional attacks and new quantum attack models, which can be implemented on traditional architecture computers and existing devices, and that can be integrated into the networks and communication protocols currently in use [7].

[1] Corresponding author

14th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and 12th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2023).
Editors: João Bispo, Henri-Pierre Charles, Stefano Cherubin, and Giuseppe Massari; Article No. 4; pp. 4:1–4:12
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithm 1 Primitives of the BIKE key encapsulation mechanism [2].

```
1: function [$H, \sigma, h$] KeyGen ($seed$, $\sigma$)
2:     $H = [h_0|h_1] = \text{PRNG}(seed)$;
3:     $h = h_1 \odot h_0^{-1}$;
4:     return $\{H, h, \sigma\}$;
5: function [$K, c$] Encaps ($h$, $m$)
6:     $e = \mathbf{H}(m)$;
7:     $s = e_0 \oplus (e_1 \odot h)$;
8:     $m' = m \oplus \mathbf{L}(e)$;
9:     $K = \mathbf{K}(\{m, \{s, m'\}\})$;
10:    return $\{K, \{s, m'\}\}$;
11: function [$K$] Decaps ($H$, $\sigma$, $c$)
12:    $e' = \text{BGFDecoder}(s, H)$;
13:    $m'' = m' \oplus \mathbf{L}(e')$;
14:    $a = (e' == \mathbf{H}(m''))\ ?\ m''\ :\ \sigma$;
15:    $K = \mathbf{K}(\{a, c\})$;
16:    return $K$;
```

The National Institute of Standards and Technology (NIST) is conducting a process for the standardization of PQC cryptosystems, in particular key encapsulation mechanisms (KEM) and digital signature schemes [21]. After the third round of the PQC standardization process, NIST selected the CRYSTALS-Kyber lattice-based KEM for standardization while appointing the fourth evaluation round to analyze further the code-based BIKE, Classic McEliece, and HQC and the isogeny-based SIKE. The performance of both the software and hardware implementations of such cryptosystems is crucial for evaluating the cryptosystems, in addition to security against traditional and quantum attacks. In particular, NIST takes Intel Haswell processors and Xilinx Artix-7 FPGAs as reference platforms for software and hardware implementations, respectively.

A KEM allows the secure transmission, through a public key algorithm, of a shared secret, which can then be expanded to generate keys to be used in a symmetric cryptosystem, which is more efficient for the transmission of long messages than a PKC scheme [28]. After generating a random element of the finite group that underlies the implemented public key scheme, this element is exchanged between the two parties of the communication, which can finally derive the shared secret by applying a hash function to the element of the finite group.

BIKE is a post-quantum KEM based on quasi-cyclic moderate-density parity-check (QC-MDPC) codes [2]. These codes are used in a scheme similar to that first proposed by Niederreiter [24]. BIKE distinguishes itself for its good trade-off between ciphertext and key lengths and performance, making it a good candidate for standardization after the fourth round [22]. Instances of BIKE are specified for NIST security levels 1, 3, and 5, providing security against quantum attacks equivalent to AES-128, -192, and -256, respectively.

The BIKE cryptosystem can be split into three primitives. Key generation produces a private-public key pair (KeyGen in Algorithm 1), encapsulation generates a shared secret and encrypts it with the public key (Encaps), and decapsulation retrieves the shared secret with the private key from the ciphertext (Decaps). Due to its QC-MDPC code-based nature, BIKE uses binary polynomial arithmetic operations and the Black-Gray-Flip decoding procedure [14], while random number generation and cryptographic hash functionalities (**H**, **K**, and **L** in Algorithm 1) are implemented by employing SHA-3 and SHAKE.

Contributions

This manuscript provides an overview and a comparison of the existing state-of-art implementations of BIKE, a QC-MDPC code-based post-quantum KEM candidate for standardization in the fourth round of NIST's PQC standardization process.

The goal is to gauge the ability to deploy BIKE on different computing platforms suitable to various real-world use-case scenarios, ranging from low-power embedded systems to desktop-class CPUs and mid-range FPGAs.

2 Related works

The literature contains a variety of proposals that provide complete software and hardware implementations of QC-MDPC code-based post-quantum cryptosystems.

2.1 State-of-the-art software implementations

On the software side, implementations of QC-MDPC code-based cryptosystems participating in the NIST PQC competition were made publicly available and distributed open-source.

Two separate software versions of LEDAcrypt, an early candidate to the NIST's PQC standardization process which was admitted to its third round of evaluation, are available at [4]. They consist of a reference version written in plain C11 and an optimized one that exploits the AVX2 extension for recent Intel Core CPUs.

[2] provides instead the two official software implementations of BIKE, a reference one written in plain C11 and an optimized one that exploits the Intel AVX512 extension. Other works from literature provide software implementations for ISAs other than the Intel x86 one, with [9] targeting Arm Cortex-M4 microcontrollers and [10] introducing support for RISC-V computing platforms. Further additional implementations of BIKE, including a fully portable one, versions optimized for AVX2 and AVX512 instruction set extensions, and implementations optimized for CPUs that support *PCLMULQDQ* and *VPCLMULQDQ* instructions, are also publicly available on Github [1].

The Intel AVX2 instruction set extension and similar ones can indeed significantly boost the performance of binary polynomial arithmetic operations. Intel introduced the *PCLMULQDQ* instruction and the corresponding hardware support in its Westmere architecture to accelerate the AES Galois/Counter Mode (AES-GCM) authenticated encryption algorithm. The *PCLMULQDQ* instruction performs the carry-less multiplication of two 64-bit operands. Similarly, the ARMv8-A architecture provides the *VMULL.P64* instruction, which takes as inputs two 64-bit NEON registers and outputs their product, computing according to binary polynomial multiplication, on a 128-bit NEON register.

The work in [12] leveraged the *VPCLMULQDQ* instruction, which is intended to further accelerate AES-GCM and which is the vectorized extension of *PCLMULQDQ*, to compute multiplications between large-degree binary polynomials, i.e.. polynomials with degree greater than 511. [13] introduced a constant-time algorithm for polynomial inversion, targeting the software implementation of BIKE and based on Fermat's little theorem. The authors optimized the exponentiation operation and further improved performance by means of a source code targeting the latest Intel Ice Lake CPUs, that support the AVX512 and *VPCLMULQDQ* instructions. The optimizations introduced in [12] and [13] are implemented within the Intel AVX2-optimized constant-time implementations of BIKE [1].

2.2 State-of-the-art hardware and hardware-software implementations

On the hardware side, the literature provides a variety of FPGA-based implementations of QC-MDPC code-based cryptosystems.

[17, 29] proposed the implementation of the McEliece cryptosystem with QC-MDPC codes on FPGAs. In particular, [17] targeted a performance-oriented design while [29] focused on a resource-optimized one. [18] discussed a fast implementation of QC-MDPC Niederreiter encryption for FPGAs, outperforming the work in [17] 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.

The authors of BIKE presented a VHDL FPGA-based implementation, targeting Xilinx Artix-7 FPGAs and providing support for the key generation, encryption, and decryption KEM primitives on a unique design [26]. However, the proposed architecture was custom-tailored to smaller FPGA targets, up to Artix-7 100, and it employed the AES and SHA-2 cryptographic functions as random oracles, thus supporting a now obsolete specification of BIKE. The work in [26] provided the first FPGA-based implementation of the BGF decoder, employed a multiplication module that minimized the BRAM usage by parallelizing the computation of a simpler schoolbook multiplication algorithm, rather than applying a more complex one such as Karatsuba's, and implemented binary polynomial inversion by employing a Fermat-based inversion algorithm that is a variant of the algorithm introduced in [19].

The work in [15] presented another FPGA-based implementation of BIKE, split into two components devoted to supporting the client-side (key generation and decapsulation) and server-side (encapsulation) primitives. The client and server cores integrated highly configurable hardware accelerators for binary polynomial multiplication [5, 31] and inversion [16] and BGF decoding [30]. Setting different parameters for the configurable accelerators allowed the authors to implement the client and server cores on FPGAs ranging from Artix-7 35 to Artix-7 200.

Finally, [25] proposed an updated FPGA-based implementation of [26] that employed a Keccak core rather than AES and SHA-2 ones, as specified in the latest version of the BIKE cryptoscheme [3]. In addition, the work in [25] only implements a dense-sparse multiplication module, which exploits the sparse representation of one of the two operands in the binary polynomial multiplication, rather than also a dense-dense one, and implements the extended Euclidean algorithm for binary polynomial inversion rather than the Fermat-based one. The proposed architecture targets Artix-7 FPGAs and the authors listed three instances implementing the whole KEM providing a range of area-performance trade-offs. The smallest one requires less resources than the lightweight one from [26] and provides a more than 3$\times$ speedup, while the largest one takes 3.7ms compared to the 4.8ms of the high-speed one from [26] while also occupying a smaller area.

On the hardware-software (HW/SW) side, [20] proposed a mixed HW/SW approach that made use of three HLS-generated accelerators, each implementing one of the BIKE primitives. The HW/SW approach allowed mixing the usage of hardware acceleration for the most computationally expensive primitives with the software execution of the least complex ones. The proposed solution resulted in different combinations of hardware-implemented and software-executed KEM primitives on three chips from the Xilinx Zynq-7000 heterogeneous SoC family, which feature ARM CPUs coupled with programmable FPGA logic equivalent to the Artix-7 one.

3 Methodology

The evaluation of state-of-the-art BIKE implementations spanned software, hardware, and hardware-software ones. On the software side, it considered 32- and 64-bit architectures, ARM and x86 ISAs, embedded- and desktop-class processors, and plain-C and AVX2-optimized software. On the hardware and hardware-software sides, we compared solutions that were human-designed and HLS-generated, targeting Xilinx FPGAs and heterogeneous SoCs.

3.1 Evaluated software implementations

The software performance analysis considered three implementations of BIKE.

The **reference C99** (**Ref C99** in Section 4) software [2] is the reference implementation from the official BIKE NIST submission and provides a code without any architecture-specific optimization, making it suitable to any target computing platform.

The **additional portable C99** (**CT C99**) software [1], written in plain C99 without any architecture-specific optimization, delivers a constant-time execution and is compatible with both 64-bit Intel and ARM architectures.

The **additional Intel AVX2-optimized** (**CT AVX2**) software [1] provides a faster constant-time implementation on Intel x86-64 CPUs that support the Intel AVX2 instruction set extension, i.e., CPUs from the Intel Haswell generation and later ones.

3.2 Evaluated hardware and hardware-software implementations

The experimental evaluation of hardware and hardware-software solutions considered three different implementations of the BIKE cryptoscheme.

The **Official** hardware implementation [25] delivers a unified design that implements the whole BIKE KEM and executes it in constant time. The authors provide three instances ranging from a lightweight one up to mid-range and high-performance ones. The proposed design, targeting Xilinx FPGAs, is described in SystemVerilog and publicly available online [8].

The **Client-server** hardware implementation [15] consists of two separate architectures devoted to client- (key generation and decapsulation) and server-side (encapsulation) operations of BIKE. The two client and server cores integrate configurable components, whose selection of the different architectural parameters results in instances targeting smaller and larger FPGAs.

The **HLS-based** hardware-software implementation [20] consists of three instances, targeting heterogeneous SoCs, that mix software execution and hardware acceleration, through HLS-generated components, of the BIKE KEM primitives. The instances differ in which primitives are executed in software and in hardware, allowing them to fit on different target chips.

3.3 Target computing platforms

The software implementations were executed on target platforms ranging from low-end ARM-based embedded systems to desktop-class Intel CPUs, while the hardware and hardware-software ones targeted Xilinx Artix-7 FPGAs and Zynq-7000 SoCs, respectively.

Arm Cortex-A9 (**ARM32** in Section 4) is an embedded-class 32-bit processor implementing the ARMv7-A instruction set architecture (ISA). We execute BIKE on an Arm Cortex-A9 dual-core processor featured on a Xilinx Zynq-7000 heterogeneous SoC. The ARM CPU

Table 1 Available FPGA resources on FPGAs from the Xilinx Artix-7 family and SoCs from the Xilinx Zynq-7000 family. Legend: **LUT** look-up tables, **FF** flip-flops, **BRAM** 36kb blocks of block RAM, **DSP** digital signal processing slices.

FPGA/SoC	LUT	FF	BRAM	DSP
Artix-7 12	8000	16000	20	40
Artix-7 15	10400	20800	25	45
Artix-7 25	14600	29200	45	80
Artix-7 35	20800	41600	50	90
Artix-7 50	32600	65200	75	120
Artix-7 75	47200	94400	105	180
Artix-7 100	63400	126800	135	240
Artix-7 200	134600	269200	365	740
Zynq-7000 Z-7010	17600	35200	60	80
Zynq-7000 Z-7015	46200	92400	95	160
Zynq-7000 Z-7020	53200	106400	140	220

has a clock frequency up to 667MHz, and the external memory mounted on the employed Digilent Zedboard development board, which houses the Zynq-7000 chip, is a 512MB DDR3. The BIKE software is executed on top of the Xilinx Petalinux operating system.

Arm Cortex-A53 (**ARM64**) is an embedded-class 64-bit processor implementing the ARMv8-A ISA. In particular, we consider the RP3A0 system-in-package mounted on a Raspberry Pi Zero 2 W, that features a quad-core 64-bit Arm Cortex-A53 processor clocked up to 1GHz and 512MB of SDRAM. We executed BIKE on the Raspberry Pi running the 64-bit Raspberry Pi OS Lite operating system, that is based on Debian 11, and setting a fixed 1GHz clock frequency through Linux *cpupower* tools.

Intel Core i5-10310U (**Intel x86-64**) is a desktop-class 64-bit processor implementing the x86-64 ISA and providing support for the Intel AVX2 extension, running at a clock frequency up to 4.4GHz. The PC mounting the Intel CPU ran the Ubuntu 20.04.3 LTS operating system. Such CPU supports the execution of the AVX2-optimized software version of BIKE.

Xilinx Artix-7 (**A7-xxx**) FPGAs are mid-range, cost-effective FPGA chips which are the suggested target for hardware implementations within the NIST PQC standardization process, which targets FPGAs in order to prevent the adoption of ASIC-specific technology optimizations and thus ensure a fair comparison of the hardware implementations. The look-up table (LUT), flip-flop (FF), block RAM (BRAM), and digital signal processing (DSP) resources available on each FPGA chip from the Xilinx Artix-7 family are listed in Table 1.

Xilinx Zynq-7000 (**Z-70xx**) chips are heterogeneous SoCs that couple an Arm Cortex-A9 dual-core processor with Artix-7 class programmable FPGA logic. The ARM CPU part has a clock frequency up to 667MHz, and the external memory mounted on the employed Digilent Zedboard development board, which houses the Zynq-7000 chip, is a 512MB DDR3. The LUT, FF, BRAM, and DSP resources available on the considered Zynq-7000 SoCs are listed in Table 1. The BIKE software [2] is executed on top of the Xilinx Petalinux operating system and extended with calls to the HLS-generated hardware accelerators.

4 Experimental evaluation

The experimental evaluation of the state-of-the-art implementations of BIKE considers first the software solutions and then the hardware and hardware-software ones. The discussion of the collected software performance results is split into the absolute execution times, to gauge

Table 2 Breakdown of the execution times of BIKE, expressed in milliseconds, for different security levels, architectures, and software implementations. Legend: KeyGen key generation, Encaps encapsulation, Decaps decapsulation, **SL*i*** NIST security level *i*.

	Target CPU, software version, security level							
	ARM32		**ARM64**		**Intel x86-64**			
	Ref C99		**CT C99**		**CT C99**		**CT AVX2**	
KEM primitive	**SL1**	**SL3**	**SL1**	**SL3**	**SL1**	**SL3**	**SL1**	**SL3**
KeyGen	332.34	920.93	21.15	66.97	3.68	11.91	0.20	0.57
Encaps	14.83	40.94	1.99	5.60	0.27	0.77	0.05	0.09
Decaps	464.82	1188.27	33.93	104.65	4.07	12.67	0.81	2.55
Overall KEM	811.98	2150.14	57.06	177.23	8.02	25.35	1.06	3.21

the actual real-world performance of the BIKE cryptoscheme, and the relative execution times, to highlight similarities and differences between the various computing platforms and software implementations. The evaluation of the hardware state-of-the-art solutions is split instead into their performance, expressed as their absolute execution time, and their FPGA resource utilization, expressed in terms of LUT, FF, BRAM, and DSP resources.

4.1 Software performance

The range of computing platforms and software implementations considered in the experimental evaluation resulted in significant differences in terms of absolute performance when executing the BIKE software, as shown by data provided in Table 2. Such performance results were collected by executing BIKE 100 times and averaging the ensuing execution times for each considered CPU and software version.

On the lower end, the **ARM32** 32-bit Arm Cortex-A9 platform, running at 667MHz, provided execution times of 812ms and 2150ms, i.e., in the order of seconds, when executing the **Ref C99** reference implementation with NIST security levels 1 and 3, respectively.

Moving to a more efficient code that made use of 64-bit instructions, i.e., the **CT C99** additional portable implementation, as well as to a more modern ARMv8-A architecture, provided a speedup of more than 10×. The performance on the **ARM64** Arm Cortex-A53 64-bit CPU, also running at a higher 1GHz clock frequency, measured at 57ms and 177ms for AES-128 and -192 security instances of BIKE, respectively.

Executing the same **CT C99** software implementation of BIKE on the **Intel x86-64** CPU resulted in a further speedup of around 7×. The different architecture and the higher clock frequency, in the order of 4GHz, allowed executing BIKE instances with security levels 1 and 3 in 8ms and 25ms, respectively.

Finally, we evaluated the execution, on the same **Intel x86-64** CPU, of the **CT AVX2** software implementation making use of instructions from the Intel AVX2 extension. The execution times of 1.1ms and 3.2ms are around 8× smaller than those obtained by the **CT C99** plain-C99 software, which highlights the effectiveness of those dedicated instructions in a software making wide use of binary polynomial arithmetic.

4.2 Software performance profile

Table 3 details the performance profile of the software execution of BIKE, on the different computing platforms, highlighting the ratio of execution time taken by the main operations comprising the three primitives of the BIKE KEM.

Table 3 Breakdown of the percentage execution times of BIKE for different security levels, architectures, and software implementations. Legend: KeyGen key generation, Encaps encapsulation, Decaps decapsulation, **SL*i*** NIST security level *i*.

		Target CPU, software version, security level							
		ARM32		**ARM64**		**Intel x86-64**			
		Ref C99		**CT C99**		**CT C99**		**CT AVX2**	
KEM primitive	**Operation**	**SL1**	**SL3**	**SL1**	**SL3**	**SL1**	**SL3**	**SL1**	**SL3**
KeyGen	PRNG	0%	0%	1%	1%	0%	0%	1%	1%
	Inversion	39%	41%	34%	35%	43%	44%	17%	17%
	Multiplication	2%	2%	2%	2%	2%	2%	1%	1%
		41%	43%	37%	38%	46%	47%	19%	18%
Encaps	**H** function	0%	0%	1%	1%	1%	1%	2%	1%
	Multiplication	2%	2%	2%	2%	2%	2%	1%	1%
	L function	0%	0%	0%	0%	0%	0%	1%	1%
	K function	0%	0%	0%	0%	0%	0%	1%	0%
		2%	2%	3%	3%	3%	3%	5%	3%
Decaps	Decoding	57%	55%	56%	56%	49%	48%	71%	75%
	L function	0%	0%	0%	0%	0%	0%	1%	1%
	H function	0%	0%	1%	1%	1%	1%	1%	1%
	K function	0%	0%	0%	0%	0%	0%	1%	0%
		57%	55%	59%	59%	51%	50%	76%	79%

On the **ARM32** ARMv7-A platform, the execution of the **Ref C99** reference implementation resulted in a performance profile characterized by binary polynomial inversion and BGF decoding occupying up to 41% and 57% of the KEM execution time, with binary polynomial multiplication taking instead up to 4% overall.

The execution of the **CT C99** additional portable implementation of BIKE on the **ARM64** ARMv8-A CPU highlighted binary polynomial inversion and BGF decoding taking up to 35% and 56% of the execution time.

Executing the same **CT C99** software on the **Intel x86-64** processor saw the KEM execution time being almost equally distributed between inversion and decoding, taking up to 44% and 49%, respectively. Overall, the results are quite similar to ARMv8-A software execution, due to not using any Intel-specific optimization.

On the contrary, the execution of the **CT AVX2** AVX2-optimized software on the same **Intel x86-64** CPU produced quite different results. The decoding procedure takes indeed a larger portion of the KEM execution time, up to 75%, while inversion only takes up to 17%. Notably, AVX2 instructions provide the higher speedup to the operations in binary polynomial arithmetic, namely multiplications and inversions, where the latter is computed as iterated multiplications and exponentiations. Binary polynomial multiplications and inversions end up therefore taking smaller shares of the KEM execution time.

Overall, the obtained results highlight QC-MDPC bit-flipping decoding and binary polynomial inversion as the two operations taking the largest share of the execution time across all considered platforms and software versions, with an aggregate share of the execution time ranging from 89% to 96%. The third largest share of execution time is occupied by binary polynomial multiplications, ranging from 2% to 4%. **H**, **K**, and **L** functions, which are not accelerated by AVX instructions, require a notable share of execution time, 8% and 5% for NIST security levels 1 and 3, respectively, only when executing AVX2-optimized software.

Table 4 Breakdown of the execution times of AES-128 security instances of BIKE, expressed in milliseconds, for different state-of-the-art FPGA-based implementations. Legend: **LW** lightweight, **MR** mid-range, **HP** high-performance instances, * aggregate for key generation and decapsulation.

	Hardware implementation							
	Official			**Client-server**		**HLS-based**		
KEM primitive	**LW**	**MR**	**HP**	**LW**	**HP**	**LW**	**MR**	**HP**
KeyGen	3.79	1.87	1.67	*5.71	*0.58	137.84	332.14	137.84
Encaps	0.44	0.28	0.13	0.03	0.03	14.86	14.86	14.86
Decaps	6.90	4.21	1.89	*5.71	*0.58	464.61	135.48	135.48
Overall KEM	11.14	6.36	3.70	5.74	0.61	617.31	482.48	288.18

4.3 Hardware and hardware-software performance

Table 4 lists the execution times, expressed in milliseconds, of the considered hardware state-of-the-art implementations of BIKE. It provides the execution times of the overall KEM as well as a breakdown at the granularity of KEM primitives for the NIST security level 1 instance of BIKE.

The lightweight, mid-range, and high-performance **Official** constant-time implementations [25] range from 11.14ms to 3.70ms. The lightweight instance is faster than 64-bit ARM software execution, while the high-performance one is more than twice faster than plain-C99 software execution on the Intel CPU but still slower than the AVX2 software executed on the same Intel CPU, which takes instead 1.06ms.

The **Client-server** hardware implementation [15] improves over the performance of the official one, with the smaller instance taking 5.74ms to execute the whole BIKE KEM and the larger one taking instead 0.61ms. The lightweight instance is thus faster than the official mid-range one, while the high-performance instance is more than six times faster than the best-performing official one. Notably, the authors do not provide a breakdown between the execution times of the key generation and decapsulation primitives, thus Table 4 provides their aggregate execution time.

Finally, the **HLS-based** hardware/software solution [20], which mixes software execution with the adoption of HLS-generated accelerators, provides an execution time for the overall KEM comprised between 617.31ms and 288.18ms. While all three instances proposed by the authors outperform the software execution on the ARM32 CPU, with a speedup up to 2.78× for the best-performing one, they are however significantly slower than the software execution on the ARM64 CPU, which takes instead 57.06ms.

The orders of magnitude of difference in the performance between human-designed hardware implementations and HLS-generated ones highlight the difficulty of HLS tools to make an efficient use of FPGA resources, in particular for applications as complex as the BIKE cryptosystem.

4.4 Hardware and hardware-software resource utilization

Table 5 lists the resource utilization, expressed in terms of LUT, FF, BRAM, and DSP resources, of the hardware state-of-the-art implementations of BIKE, and it details the smallest FPGA or SoC that fits the required amount of resources.

The **Official** constant-time implementations [25] require the smallest amount of resources, with the lightweight, mid-range, and high-performance instances fitting respectively on Artix-7 25, 35, and 50 FPGAs. With respect to the resources available on Artix-7 chips, the most relatively used resources are LUTs, which thus concur to defining the smallest FPGA which can fit the BIKE hardware implementation.

Table 5 Resource utilization of AES-128 security instances of BIKE, expressed in terms of LUT, FF, BRAM, and DSP resources, for different state-of-the-art FPGA-based implementations. Legend: **LW** lightweight, **MR** mid-range, **HP** high-performance instances.

	Hardware implementation							
	Official			**Client-server**		**HLS-based**		
Resource	**LW**	**MR**	**HP**	**LW**	**HP**	**LW**	**MR**	**HP**
LUT	12319	19607	25549	51596	217932	13567	37160	50727
FF	3896	5008	5462	29206	97700	11621	38118	49739
BRAM	9	17	34	73.5	632.5	40	90	130
DSP	7	9	13	0	0	0	35	35
Target	A7-25	A7-35	A7-50	A7-100	2×A7-200	Z-7010	Z-7015	Z-7020

The better performance of **Client-server** implementations [15] comes at the cost of a larger amount of FPGA resources. In particular, they implement two separate components, one dedicated to key generation and decapsulation and the other devoted to encapsulation. The smallest instance proposed by the authors requires an Artix-7 50 chip for the client core and an Artix-7 35 FPGA for the server one, while the largest one fits each core on a separate Artix-7 200 chip. Notably, both the client and server cores do not make use of any DSPs.

Finally, the **HLS-based** hardware/software instances [20] target the Zynq-7000 Z-7010, Z-7015, and Z-7020 SoCs. In particular, the lightweight one implements in hardware the lone key generation primitive, while the mid-range one implements only decapsulation and the high-performance one instantiates both the former and latter, resorting to software execution for the lone encapsulation. Although not providing a performance that is comparable to the human-designed accelerators, the HLS-generated accelerators show a significant usage of FPGA resources, in particular of LUT and BRAM ones.

5 Conclusions

This work provided an overview and a comparison of the software, hardware, and hardware-software state-of-art implementations of BIKE.

Performance results highlighted significant differences in terms of software execution times across a variety of computing platforms and software implementations, with the execution of the whole BIKE KEM taking a time in the order of seconds on lower-end embedded-class ARM CPUs and a few milliseconds on desktop-class Intel ones with support for AVX2 dedicated instructions. On the hardware side, the human-designed FPGA-based solutions were shown to outperform the reference plain-C99 software executed on Intel CPUs. The best-performing hardware solution could even outperform the AVX2-optimized software, completing the BIKE execution in 0.61ms compared to the software's 1.06ms. On the contrary, HLS-generated solutions highlighted the difficulty to generate effective hardware accelerators through high-level synthesis for target applications as complex as QC-MDPC code-based cryptosystems. The considered hardware-software solutions were still able to outperform the reference software execution on ARM32 CPUs by almost three times.

References

1 Amazon Web Services – Labs. Additional implementation of bike (bit flipping key encapsulation). `https://github.com/awslabs/bike-kem`, 2020.

2 Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE website. `https://www.bikesuite.org/`, 2017.

3 Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. BIKE: Bit flipping key encapsulation – round 3 submission. `https://bikesuite.org/files/v4.2/BIKE_Spec.2021.09.29.1.pdf`, 2021.

4 Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAcrypt website. `https://www.ledacrypt.org/`, 2017.

5 Alessandro Barenghi, William Fornaciari, Andrea Galimberti, Gerardo Pelosi, and Davide 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)*, pages 739–742, 2019. `doi:10.1109/ICECS46596.2019.8964882`.

6 Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography – PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

7 Daniel J Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, 2017.

8 Chair for Security Engineering @ Ruhr-Universität Bochum. Racingbike: Improved polynomial multiplication and inversion in hardware. `https://github.com/Chair-for-Security-Engineering/RacingBIKE`, 2021.

9 Ming-Shing Chen, Tung Chou, and Markus Krausz. Optimizing bike for the intel haswell and arm cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):97–124, July 2021. `doi:10.46586/tches.v2021.i3.97-124`.

10 Ming-Shing Chen, Tim Güneysu, Markus Krausz, and Jan Philipp Thoma. Carry-less to bike faster. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security*, pages 833–852, Cham, 2022. Springer International Publishing.

11 W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. `doi:10.1109/TIT.1976.1055638`.

12 N. Drucker, S. Gueron, and V. Krasnov. Fast multiplication of binary polynomials with the forthcoming vectorized vpclmulqdq instruction. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 115–119, June 2018. `doi:10.1109/ARITH.2018.8464777`.

13 Nir Drucker, Shay Gueron, and Dusan Kostic. Fast polynomial inversion for post quantum qc-mdpc cryptography. In Shlomi Dolev, Vladimir Kolesnikov, Sachin Lodha, and Gera Weiss, editors, *Cyber Security Cryptography and Machine Learning*, pages 110–127, Cham, 2020. Springer International Publishing. `doi:10.1007/978-3-030-49785-9_8`.

14 Nir Drucker, Shay Gueron, and Dusan Kostic. Qc-mdpc decoders with several shades of gray. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 35–50, Cham, 2020. Springer International Publishing.

15 Andrea Galimberti, Davide Galli, Gabriele Montanaro, William Fornaciari, and Davide Zoni. Fpga implementation of bike for quantum-resistant tls. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 539–547, 2022. `doi:10.1109/DSD57027.2022.00078`.

16 Andrea Galimberti, Gabriele Montanaro, and Davide Zoni. Efficient and scalable fpga design of gf(2m) inversion for post-quantum cryptosystems. *IEEE Transactions on Computers*, 71(12):3295–3307, 2022. `doi:10.1109/TC.2022.3149422`.

17 Stefan Heyse, Ingo von Maurich, and Tim Güneysu. Smaller keys for code-based cryptography: Qc-mdpc mceliece implementations on embedded devices. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, pages 273–292, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

18 Jingwei Hu and Ray C.C. Cheung. Area-time efficient computation of niederreiter encryption on qc-mdpc codes for embedded hardware. *IEEE Transactions on Computers*, 66(8):1313–1325, 2017. doi:10.1109/TC.2017.2672984.

19 Jingwei Hu, Wei Guo, Jizeng Wei, and Ray C. C. Cheung. Fast and generic inversion architectures over GF(2^m) using modified itoh-tsujii algorithms. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(4):367–371, 2015. doi:10.1109/TCSII.2014.2387612.

20 Gabriele Montanaro, Andrea Galimberti, Ernesto Colizzi, and Davide Zoni. Hardware-software co-design of bike with hls-generated accelerators. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4, 2022. doi:10.1109/ICECS202256217.2022.9970992.

21 National Institute of Standards and Technology (NIST) – U.S. Department of Commerce. Post-quantum cryptography. https://csrc.nist.gov/projects/post-quantum-cryptography, 2021.

22 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. doi:10.6028/NIST.IR.8413.

23 National Security Agency. Commercial National Security Algorithm Suite 2.0 (CNSA 2.0) Cybersecurity Advisory (CSA). https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF, 2022.

24 Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

25 Jan Richter-Brockmann, Ming-Shing Chen, Santosh Ghosh, and Tim Güneysu. Racing bike: Improved polynomial multiplication and inversion in hardware. Cryptology ePrint Archive, Paper 2021/1344, 2021. URL: https://eprint.iacr.org/2021/1344.

26 Jan Richter-Brockmann, Johannes Mono, and Tim Güneysu. Folding bike: Scalable hardware implementation for reconfigurable devices. *IEEE Transactions on Computers*, 2021. doi:10.1109/TC.2021.3078294.

27 R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. doi:10.1145/359340.359342.

28 Victor Shoup. A proposal for an iso standard for public key encryption. Cryptology ePrint Archive, Paper 2001/112, 2001. URL: https://eprint.iacr.org/2001/112.

29 Ingo von Maurich and Tim Güneysu. Lightweight code-based cryptography: Qc-mdpc mceliece encryption on reconfigurable devices. In *2014 Design, Automation and Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2014. doi:10.7873/DATE.2014.051.

30 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*, 8:163419–163433, 2020. doi:10.1109/ACCESS.2020.3020262.

31 D. Zoni, A. Galimberti, and W. Fornaciari. Flexible and scalable fpga-oriented design of multipliers for large binary polynomials. *IEEE Access*, 8:75809–75821, 2020. doi:10.1109/ACCESS.2020.2989423.